Exploiting Qualitative User Feedback in Deterministic and Probabilistic Databases

by

Niccolò Meneghetti

August 11, 2016

A dissertation submitted to the
Faculty of the Graduate School of
the University at Buffalo, State University of New York
in partial fulfillment of the requirements for the
degree of

Doctor of Philosophy

Department of Computer Science and Engineering

To my sister Costanza and my parents Chiaretta and Ettore.

ACKNOWLEDGEMENTS

I would like to express my gratitude to my adviser, Dr. Jan Chomicki, for being an excellent mentor and a constant point of reference during my graduate studies. His guidance and support were indispensable for me to achieve my first publication. I am also deeply grateful to my co-adviser Dr. Oliver Kennedy; many parts of this dissertation are a direct result of our collaboration. I would like to thank Dr. Jing Gao for serving in my committee, and for being always kind and encouraging. A special thanks goes to all my co-authors: to Dr. Paolo Ciaccia, who introduced me to the world of academic research, to Ying Yang, who is always passionate about our shared projects, and to Denis Mindolin. I am deeply grateful to Dr. Wolfgang Gatterbauer for his continued support and for investing so much time on our shared research. I would also like to thank Dr. Jinhui Xu for introducing me to the beauties of computational geometry, and Dr. Hung Ngo, whose lectures are a consistent source of inspiration. Last, but not least, I would like to thank all my friends in Buffalo, who made this whole experience much more enjoyable.

TABLE OF CONTENTS

LIST OF TABLES

Table

ABSTRACT

Exploiting Qualitative User Feedback in Deterministic and Probabilistic Databases

by

Niccolò Meneghetti

Committee:

Dr. Jan Chomicki (Chair)

Dr. Oliver Kennedy (Co-Chair)

Dr. Jing Gao

This dissertation consists of two parts. In the first part we investigate the use of qualitative user preferences in the context of deterministic databases. In the second part we study the same problem in the context of probabilistic databases. A qualitative preference is a partial order over a set of query-answers. In deterministic databases, they are used to identify the answers that are most interesting to the user. In the first part of this dissertation we adopt the framework of Prioritized Skylines, a popular approach for preference reasoning. Our contribution focuses on query evaluation: we introduce a novel output-sensitive algorithm, whose worst-case computational complexity depends explicitly on the size of the output. This is a very desirable property in the context of preference queries, as the output is expected to be, on average, only a small fraction of the input. Additionally, we show our algorithm exhibits linear complexity in the average-case scenario. The second part of this dissertation is dedicated to probabilistic databases. In this context, qualitative preferences are used to encode

a set of logical implications between query-answers: if the user states that a query-answer implies another, then the marginal probability of the latter must be as large as the marginal probability of the former. Such information can be used to speed-up the training process of a probabilistic database. We introduce Beta Probabilistic Databases (B-PDBs), a generalization of tuple-independent probabilistic databases, designed to support Bayesian updates and parameter learning in an efficient, scalable way. The key feature of B-PDBs is that they can be trained using samples of query-answers as evidence; this way, the user is relieved from the need of specifying the model's parameters explicitly, in a quantitative fashion. Additionally, B-PDBs can leverage qualitative user-feedback to speed-up the learning process.

CHAPTER I

Introduction

1.1 Background and Motivation

The most common way to query databases is to express users' goals in a *quantitative* fashion, by specifying a set of logical constraints to be satisfied, together with a scoring function that ranks the answers into some desired order. For example, the user of a travel booking website may submit the following query: "*Retrieve all the 4-stars hotels in Cambridge, MA, whose distance to the Charles River is less than 3 miles; sort the results by increasing price and distance to the river*". This classic approach presents several limitations. Without prior knowledge about the content of the database, it is very hard to devise the right set of constraints to identify few interesting query-answers. If the constraints are too strict, the database will likely return an empty answer; if they are too permissive, the database will return too many results. For example: if there are no hotels within 3 miles of the Charles River, the user may still be willing to consider an accommodation that is 3.5 miles away, or a 3-stars hotel. On the other hand, if the distance threshold was set to 100 miles, the database would return almost every hotel in Boston. In both scenarios the user is required to reformulate her queries, possibly several times, until the desired answers are returned without being mixed with spurious results. Such trial-and-error procedure is hardly efficient. Even when the selection constraints are easily found, choosing the

right sorting order is not a trivial task. In the example above, the sorting order is determined by two factors, (1) price and (2) distance to the river, and the former is set to a higher priority than the latter. Therefore, a cheap hotel must be ranked higher than a more expensive one, independently from their respective distance to the river. Such ranking criterion may not be the best way to describe the user's preferences: she may favor an accommodation that is slightly more expensive than another, if it is significantly closer to the river.

In this dissertation we adopt a different paradigm for interacting with databases, where the user, rather that expressing her goals in quantitative terms, by means of selection predicates and scoring functions, specifies her objectives *qualitatively*. Following [19, 20, 51], we define *qualitative preferences* as binary relations over query-answers. Intuitively, a qualitative preference is a simple logical criterion for comparing pairs of query-answers. For example, the user may state: "I prefer hotels that are located near the Charles River, all else being equal or better", meaning that given two hotels h_1 and h_2, if h_1 is closer to the river, it should be preferred to h_2, as long as its price is not greater than the price of h_2, and h_1 has as many stars as h_2. In such case we say h_1 *dominates* h_2. A database can exploit qualitative preferences by filtering away all the answers that are dominated by some other answer [19]. This way, only few results are returned, possibly those that are most interesting to the user. *Skyline queries* [12] are one of the most common instances of preference reasoning. Given a set of preference criteria, we say that an answer h_1 Pareto-dominates an answer h_2 if h_1 dominates h_2 w.r.t. at least one criterion and it is not dominated w.r.t. all the others. Skyline queries return all the answers that are *Pareto-optimal*, i.e. all the answers that are not Pareto-dominated by any other answer.

Qualitative preferences can be leveraged in the context of probabilistic databases too. A probabilistic database [31, 45, 77] is designed to store noisy, uncertain relational data. It defines a probability distribution $\mathbb{P}[\cdot]$ over a collection of *possible*

worlds, a set of deterministic databases that share the same fixed schema. Given a query, probabilistic databases return sets of *possible answers*, annotated with marginal probabilities. The marginal probability of a query-answer is the likelihood of observing such answer when the query is executed over a possible world chosen at random from $\mathbb{P}[\cdot]$. In the context of probabilistic databases, we say that an answer h_1 dominates an answer h_2 if we believe that every possible world that returns h_2 should return h_1 as well. In other words, the statement "h_1 dominates h_2" represents the user's belief that h_2 implies h_1 and, consequently, the marginal probability of the latter should be at least as large as the marginal probability of the former.

1.2 Contributions

This dissertation consists of two parts. In the first part (Chapter II) we investigate the use of qualitative preferences in the context of deterministic databases. In the second part (Chapter III) we study the same problem in the context of probabilistic databases.

In Chapter II we adopt the framework of P-Skyline queries, a generalization of Skyline queries where the user is allowed to prioritize some preference criteria with respect to others. P-skyline queries were originally studied in [51, 64]. Our contribution focuses on query evaluation: we introduce a novel output-sensitive algorithm, whose worst-case computational complexity depends explicitly on the size of the output. This is a very desirable property in the context of p-skyline queries, as the output is expected to be, on average, only a small fraction of the input. Additionally, we prove that our algorithm exhibits linear complexity in the average-case scenario. We conclude the study by presenting extensive experimental results. Most of the content of Chapter II has been published in a SIGMOD paper [63].

In Chapter III we introduce Beta Probabilistic Databases (B-PDBs), a generalization of tuple-independent probabilistic databases [31, 24], designed to support

Bayesian updates and parameter learning in a efficient, scalable way. The key feature of B-PDBs is that they can be trained using samples of query-answers as evidence; this way, the user is relieved from the need of specifying the model's parameters explicitly, in a quantitative fashion. In Chapter III we show how to update the parameters of a B-PDB in response to the observation of a new query-answer. Our procedure supports arbitrary conjunctive query. We prove that multiple updates lead, in the limit, to a maximum likelihood estimate of the model's parameters. Eventually, we show how to leverage user-provided feedback, in the form of a qualitative dominance relationship, to speed-up the learning process.

CHAPTER II

Output-Sensitive Evaluation of P-Skyline Queries

Skylines assume that all attributes are equally important, as each dimension can always be traded off for another. Prioritized skylines (p-skylines) generalize skylines, by allowing the user to prioritize some dimensions w.r.t. others. In this chapter we develop an efficient in-memory divide-and-conquer algorithm for answering p-skyline queries. Our algorithm is output-sensitive*; this is very desirable in the context of preference queries, since the output is expected to be, on average, only a small fraction of the input. We prove that our method is well behaved in both the worst- and the average-case scenarios. We conclude our study with extensive experimental results.*

2.1 Background and Notation

In this chapter we adopt the following typographic conventions: sets of tuples (or operators returning set of tuples) are written as capital letters (like D, B, W or $M(D)$), sets of attributes are written in calligraphic font (like $\mathcal{A}$, $\mathcal{C}$ or $\mathcal{E}$), actual attributes are written in boldface font (for example: $\mathbf{A}_1, \mathbf{A}_2, \ldots$). For readers' convenience, Table A.1 summarizes the most common notations used throughout the chapter. All notational conventions are formally introduced and explained in more details in the following sections.

2.1.1 Preferences

Several frameworks have been proposed for modeling preferences in deterministic databases, many of which are surveyed in [75]. In this chapter we follow the *qualitative* approach [19, 51, 20], as we assume user's wishes are modeled as strict partial orders. We denote by $\mathcal{A} = \{\mathbf{A}_1, \mathbf{A}_2, \ldots\}$ a finite set of attributes defining the schema of the relation returned by a query, and by U the set of all possible tuples over such schema (i.e. the set of all possible query-answers). Without lack of generality, we assume attributes can be either discrete or range over the set of rational numbers. A *preference* is a strict partial order $\succ$ over U, a subset of $U \times U$ being transitive and irreflexive. The assertion $t' \succ t$ (t' *dominates* t) means the user prefers t' over t, i.e. she's always willing to trade the answer t for the answer t' if she is given the chance. Given a relation instance $D \subseteq U$ and a preference $\succ$, a *preference query* retrieves all the maximal elements of the partially ordered set $(D, \succ)$. Following the notation of [54], we denote by $M_\succ(D)$ the set of all these maximal tuples:

$$M_\succ(D) = \{t \in D \mid \nexists\, t' \in D \text{ s.t. } t' \succ t\}$$

Every preference $\succ$ induces an indifference relation ($\sim$): $t' \sim t$ holds whenever $t' \not\succ t$ and $t \not\succ t'$. A preference $\succ$ is a *weak order* when $\sim$ is transitive. A weak order extension of $\succ$ is simply an arbitrary weak order containing $\succ$. Two tuples t' and t are *indistinguishable* with respect to $\succ$ when they agree on all attributes that are relevant for deciding $\succ$. In such case we write $t' \approx t$. We denote by $t' \succeq t$ the fact that t' is either better than or indistinguishable from t. If D and D' are two relation instances, we write $D' \not\succeq D$ when there is no pair of tuples (t', t) in $D' \times D$ such that $t' \succeq t$.

Declaring a preference by enumerating its elements is impractical in all but the most trivial domains. Kießling [51] suggested to define preferences in a constructive

fashion, by iteratively composing single-attribute preferences into more complex ones. From his work we borrow two composition operators, namely the *Pareto accumulation* ($\otimes$) and the *Prioritized accumulation* (&). Denote $\succ_1$ and $\succ_2$ two strict partial orders; their Pareto accumulation $\succ_{1\otimes 2}$ is defined as follows

$$\forall\, t', t \quad t' \succ_{1\otimes 2} t \iff (t' \succ_1 t \ \wedge\ t' \succeq_2 t) \ \vee (t' \succ_2 t \ \wedge\ t' \succeq_1 t)$$

In other words, $t' \succ_{1\otimes 2} t$ holds when t' is better according to one of the preferences $\succ_1$ and $\succ_2$, and better or indistinguishable from t according to the other. Hence, the two preferences are equally important. The prioritized composition of $\succ_1$ and $\succ_2$ is defined as follows

$$\forall\, t', t \quad t' \succ_{1\&2} t \iff t' \succ_1 t \ \vee (t' \approx_1 t \ \wedge\ t' \succ_2 t)$$

Clearly $\succ_{1\&2}$ gives more weight to the first preference, as $\succ_2$ is taken into consideration only when two tuples are indistinguishable w.r.t. $\succ_1$. Both operators are associative and the Pareto accumulation is commutative [51]. Several standard query languages have been extended to support preference constructors like the Pareto- and the Prioritized accumulation, including SQL [53], XPATH [52], and SPARQL [73].

2.1.2 P-Skyline Queries

A *p-expression* [64] is a formula denoting multiple applications of the above operators. P-expressions respect the following grammar:

$$\begin{aligned} \textit{pExpr} &\rightarrow \textit{paretoAcc} \mid \textit{prioritizedAcc} \mid \textit{attribute} \\ \textit{paretoAcc} &\rightarrow \textit{pExpr} \otimes \textit{pExpr} \\ \textit{prioritizedAcc} &\rightarrow \textit{pExpr}\ \&\ \textit{pExpr} \\ \textit{attribute} &\rightarrow \mathbf{A}_1 | \mathbf{A}_2 | \ldots \end{aligned}$$

where all non-terminal symbols are lower-case and each terminal symbol is either a composition operator ($\otimes$ or &) or a single-attribute preference, with the restriction that no attribute should appear more than once. A single-attribute preference (also denoted by $\succ_{\mathbf{A}_i}$) is simply an arbitrary total order defined over the attribute's domain. Without lack of generality, we will assume users rank values in natural order (i.e. they prefer small values to larger ones), unless stated differently. With a little abuse of notation, we will use single-attribute preferences for ranking either tuples or values, depending on the context.

Example 1. *Assume that a dealer is offering the following cars*

id	**P** *(price)*	**M** *(mileage)*	**T** *(transmission)*
t_1	\$ 11,500	50,000 miles	*automatic*
t_2	\$ 11,500	60,000 miles	*manual*
t_3	\$ 12,000	50,000 miles	*manual*
t_4	\$ 12,000	60,000 miles	*automatic*

and that we are looking for a cheap vehicle, with low mileage, possibly with manual shift. Notice that while the first car is Pareto-optimal for price and mileage, if we want a manual transmission we need to give up either on getting the best price or the best mileage. Depending on our priorities, we can model our preferences in different ways. All the following are well-formed and meaningful p-expressions:

1. $\mathbf{P}$
2. $(\mathbf{P} \otimes \mathbf{M})$ & $\mathbf{T}$
3. $(\mathbf{P}$ & $\mathbf{T}) \otimes \mathbf{M}$
4. $\mathbf{M}$ & $\mathbf{T}$ & $\mathbf{P}$

Expression (1) states that we care only about price. If that is the case, we should buy either t_1 or t_2. Expression (2) states that we are looking for cars that are Pareto-optimal w.r.t. price and mileage, and that we take into consideration transmission only to decide between cars that are indistinguishable in terms of price and mileage. In this case t_1 is the best option. Expression (3) is more subtle: we are looking for

cheap cars, with low mileage and manual transmission, but we are not willing to pay an extra price for the manual transmission. In this case we should buy either t_1 or t_2, since t_1 dominates t_3 and t_4. Finally, expression (4) denotes a lexicographic order: amongst the cars with the lowest mileage, we are looking for one with manual transmission, and price is the least of our concerns. In this case we should buy t_3.

Given a p-expression π we denote by $\succ_\pi$ the preference relation defined by it. Notice that $\succ_\pi$ is guaranteed to be a strict partial order [51]. We denote by $\mathcal{V}ar(\pi)$ the set of attributes that appear inside π, i.e. those that are relevant for deciding $\succ_\pi$. Notice that $t' \approx_\pi t$ holds iff t' and t agree on every attribute in $\mathcal{V}ar(\pi)$; in the following we will simply say that t and t' are indistinguishable w.r.t. attributes in $\mathcal{V}ar(\pi)$.

Definition 1. *Given a relation instance $D \subseteq U$ and a p-expression π, a* p-skyline query *returns the set $M_\pi(D)$ of the maximal elements of the poset $(D, \succ_\pi)$.*

The computational complexity of p-skyline queries depends strongly on the size of the input, the size of the output, and the number of attributes that are relevant to decide $\succ_\pi$. Hence, our analysis will take in consideration mainly three parameters: n, the number of tuples in the input, v the number of tuples that belong to the p-skyline, and d, the cardinality of $\mathcal{V}ar(\pi)$.

Every p-expression π implicitly induces a priority order over the attributes in $\mathcal{V}ar(\pi)$. Mindolin and Chomicki [64] modeled these orders using p-graphs, and showed how they relate to the semantics of p-skyline preferences.

Definition 2. *A p-graph Γ_π is a directed acyclic graph having one vertex for each attribute in $\mathcal{V}ar(\pi)$. The set $E(\Gamma_\pi)$ of all edges connecting its vertices is defined recursively as follows:*

- *if π is a single-attribute preference, then $E(\Gamma_\pi) \equiv \emptyset$*
- *if $\pi = \pi_1 \otimes \pi_2$ then $E(\Gamma_\pi) \equiv E(\Gamma_{\pi_1}) \cup E(\Gamma_{\pi_2})$*

- *if* $\pi = \pi_1 \,\&\, \pi_2$ *then* $E(\Gamma_\pi) \equiv E(\Gamma_{\pi_1}) \cup E(\Gamma_{\pi_2}) \cup (\mathcal{V}ar(\pi_1) \times \mathcal{V}ar(\pi_2))$

Intuitively, a p-graph Γ_π *contains an edge from* $\mathbf{A}_i$ *to* $\mathbf{A}_j$ *iff the preference on* $\mathbf{A}_i$ *is more important than the one on* $\mathbf{A}_j$*. Notice that p-graphs are transitive by construction and, since p-expressions do not allow repeated attributes, they are guaranteed to be acyclic. In order to simplify the notation in the following sections we will not use p-graphs directly, but we will refer mostly to their transitive reductions*[1] Γ_π^r *(see Figure 2.1). In relation to* Γ_π^r *we define the following sets of attributes:*

$\mathcal{S}ucc_\pi(\mathbf{A}_i)$	*immediate successors of* $\mathbf{A}_i$
$\mathcal{D}esc_\pi(\mathbf{A}_i)$	*descendants of* $\mathbf{A}_i$
$\mathcal{P}re_\pi(\mathbf{A}_i)$	*immediate predecessors of* $\mathbf{A}_i$
$\mathcal{A}nc_\pi(\mathbf{A}_i)$	*ancestors of* $\mathbf{A}_i$
$\mathcal{R}oots_\pi$	*nodes having no ancestors*

We will denote by $d_{\mathbf{A_i}}$ the depth of $\mathbf{A}_i$, i.e. the length of the longest path in Γ_π^r from any root to $\mathbf{A}_i$ (roots have depth 0).

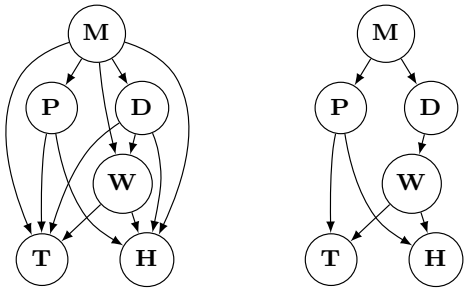
Example 2. *A customer is looking for a low-mileage* ($\mathbf{M}$) *car; amongst barely used models, she is looking for a car that is available near her location* ($\mathbf{D}$) *for a good price* ($\mathbf{P}$)*, possibly still under warranty* ($\mathbf{W}$)*. In order to obtain a comprehensive warranty she is willing to pay more, but not to drive to a distant dealership, since regular maintenance would require her to go there every three months. All else being equal, she prefers cars equipped with heated seats* ($\mathbf{H}$) *and manual transmission* ($\mathbf{T}$)*. Her preferences can be formulated using the following p-expression:*

$$\mathbf{M} \,\&\, ((\mathbf{D}\&\mathbf{W}) \otimes \mathbf{P}) \,\&\, (\mathbf{T} \otimes \mathbf{H})$$

Figure 2.1(a) shows the corresponding p-graph and Figure 2.1(b) its transitive reduction. Notice the p-graph is not a weak order, thus the attributes cannot be simply

[1]Since every p-graph is a finite strict partial order, the transitive reduction Γ_π^r is guaranteed to exist and to be unique: it consists of all edges that form the only available path between their endpoints.

ranked using a scoring function.



(a) Γ_π (b) Γ_π^r

Figure 2.1: (a) the p-graph of the expression $\mathbf{M}$ & $((\mathbf{D}\&\mathbf{W}) \otimes \mathbf{P})$ & $(\mathbf{T} \otimes \mathbf{H})$ and (b) its transitive reduction.

We refer to Figure 2.1(b) to exemplify how the operators $\mathcal{S}\text{ucc}_\pi(\cdot)$, $\mathcal{D}\text{esc}_\pi(\cdot)$, $\mathcal{P}\text{re}_\pi(\cdot)$ and $\mathcal{A}\text{nc}_\pi(\cdot)$ work. It is easy to see that $\mathcal{D}\text{esc}_\pi(\mathbf{D}) = \{\mathbf{W}, \mathbf{T}, \mathbf{H}\}$ while $\mathcal{S}\text{ucc}_\pi(\mathbf{D})$ contains only $\mathbf{W}$. Similarly, $\mathcal{A}\text{nc}_\pi(\mathbf{W})$ consists of $\{\mathbf{M}, \mathbf{D}\}$, but $\mathcal{P}\text{re}_\pi(\mathbf{W})$ contains only $\mathbf{D}$. $\mathcal{R}\text{oots}_\pi$ contains only $\mathbf{M}$.

The following result from [64] highlights the relation between p-graphs and the semantics of p-skylines:

Proposition 1. *[64] Denote by t and t' two distinct tuples, by $\mathcal{B}etter_\pi(t', t)$ the set of attributes where t' is preferred to t, and by $\mathcal{T}op_\pi(t', t)$ the topmost elements in Γ_π^r where t and t' disagree. The following assertions are equivalent:*

1. $t' \succ_\pi t$
2. $\mathcal{B}etter_\pi(t', t) \supseteq \mathcal{T}op_\pi(t', t)$
3. $\mathcal{D}esc_\pi(\mathcal{B}etter_\pi(t', t)) \supseteq \mathcal{B}etter_\pi(t, t')$

In other words, $t' \succ_\pi t$ holds iff the two tuples are distinguishable and every node in the p-graph for which t is preferred has an ancestor for which t' is preferred. We will take advantage of this result in Sections B.1.2 and 2.5.

2.1.3 Skyline Queries

A *skyline query* [12, 21] returns all the query-answers that are Pareto-optimal. A tuple is Pareto-optimal if no other tuple dominates w.r.t. one dimension, without being dominated w.r.t. some other dimension. Skyline queries can be seen as a special case of p-skyline queries: the Pareto-optimality criterion over an arbitrary set of attributes $\{\mathbf{A}_i, \mathbf{A}_j, \ldots, \mathbf{A}_k\}$ can be formulated simply as $\mathbf{A}_i \otimes \mathbf{A}_j \otimes \ldots \otimes \mathbf{A}_k$. From now on we will denote by *sky* the above expression, and by $M_{\text{sky}}(D)$ the result of a skyline query. Clearly, the meaning of this notation will depend on the content of $\mathcal{V}ar(\text{sky})$.

Proposition 2. *[64] Let π and π' be two p-expressions such that $\mathcal{V}ar(\pi) = \mathcal{V}ar(\pi')$. The following containment properties hold*

$$\succ_\pi \subset \succ_{\pi'} \Leftrightarrow E(\Gamma_\pi) \subset E(\Gamma_{\pi'}) \tag{2.1}$$

$$\succ_\pi = \succ_{\pi'} \Leftrightarrow E(\Gamma_\pi) = E(\Gamma_{\pi'}) \tag{2.2}$$

From Proposition 2 we can directly infer that $M_\pi(D) \subseteq M_{\text{sky}}(D)$ as long as $\mathcal{V}ar(\pi) = \mathcal{V}ar(sky)$.

Skyline queries have been very popular in the database community. Over several years a plethora of algorithms have been proposed, including BNL [12], SFS [22], LESS [35], SALSA [6], BBS [68], and many others. For the purpose of this chapter we briefly review BNL, together with its extensions. BNL (block-nested-loop) allocates a fixed-size memory buffer able to store up to k tuples, the *window*, and repeatedly performs a linear scan of the input; during each iteration i each tuple t is compared with all the elements currently in the window. If t is dominated by any of those, it is immediately discarded, otherwise all the elements of the window being dominated by t are discarded and t is added to the window. If there is not enough space, t is written to a temporary file T_i. At the end of each iteration i all the tuples that entered the

window when T_i was empty are removed and added to the final result; the others, if any, are left in the window to be used during the successive iteration $(i+1)$, that will scan the tuples stored in T_i. The process is repeated until no tuples are left in the temporary file.

SFS (sort-filter-skyline) improves BNL with a pre-sorting step; at the very beginning the input is sorted according to a special ranking function, ensuring that no tuple dominates another that precedes it. The resulting algorithm is pipelineable and generally faster than BNL. LESS (Linear Elimination Sort for Skyline [35]) and SALSA (Sort and Limit Skyline algorithm [6]) improve this procedure by applying an elimination filter and an early-stop condition.

Skylines have been studied for several decades in computational geometry, as an instance of the *maximal vector problem* [59, 11, 54]. The first divide and conquer algorithm is due to Kung, Luccio and Preparata [59] and is similar to the one used in this chapter: the general idea is to split the data set in two halves, say B and W, so that no tuple in W is indistinguishable from or dominates any tuple in B ($W \not\succeq_{\text{sky}} B$); then, the skyline of both halves is computed recursively, obtaining $M_{\text{sky}}(B)$ and $M_{\text{sky}}(W)$. The last step is to remove from $M_{\text{sky}}(W)$ all tuples dominated by some element in $M_{\text{sky}}(B)$. This operation is called *screening*. Let W' be the set of tuples from $M_{\text{sky}}(W)$ that survive the screening, the algorithm returns the set $M_{\text{sky}}(B) \cup W'$, containing each and every skyline point. The base case for the recursion is when B and W are small enough to make the computation of the skyline trivial.

In [8] Bentley et al. developed a similar, alternative algorithm, and in [9] proposed a method that is provably fast in the average-case. Kirkpatrick and Seidel [54] were the first to propose an output-sensitive procedure. More recently, [72] developed a divide-and-conquer algorithm that is efficient in external memory, [62] improved the algorithm from Kirkpatrick and Seidel, while several results have been obtained using

the word-RAM model [32, 18, 3]. Following [54] we denote by $C_d(v, n)$ the worst-case complexity of skyline queries, and by $F_d(b, w)$ the worst-case complexity of screening, assuming $b = |B|$ and $w = |W|$. The following complexity results were obtained in [54, 58, 59]. In this chapter we will prove similar results in the context of p-skylines.

Proposition 3. *[54] The following upper-bounds hold on the complexity of the maximal vector problem:*

1. $C_d(v, n) \leq \mathcal{O}(n)$ *for* $d = 1$
2. $C_d(v, n) \leq \mathcal{O}(n \log v)$ *for* $d = 2, 3$
3. $C_d(v, n) \leq \mathcal{O}\left(n \log^{d-2} v\right)$ *for* $d \geq 4$
4. $C_d(v, n) \leq \mathcal{O}(n)$ *when* $v = 1$

Proposition 4. *[54, 58, 59] The following upper-bounds on the complexity of the screening problem hold:*

1. $F_d(b, w) \leq \mathcal{O}(b + w)$ *for* $d = 1$
2. $F_d(b, w) \leq \mathcal{O}((b + w) \log b)$ *for* $d = 2, 3$
3. $F_d(b, w) \leq \mathcal{O}\left((b + w) \log^{d-2} b\right)$ *for* $d \geq 4$
4. $F_d(b, w) \leq \mathcal{O}(w)$ *when* $b = 1$

2.2 Contributions

Output-sensitive algorithms are quite popular in computational geometry. Starting from the classical results on convex hulls by Kirkpatrick and Seidel [55] and Chan [17], output-sensitive solutions have been explored in several problem domains. An output-sensitive algorithm is designed to be efficient when v (the output-size) is a small fraction of n (the input-size). More specifically, its asymptotic complexity should depend explicitly on v, and gracefully degrade to the level of the best known output-*insensitive* algorithms when $v \in \Omega(n)$.

The main contribution of this chapter is to develop a novel output-sensitive algorithm for p-skyline queries. We show the problem is $\mathcal{O}\left(n \log^{d-2} v\right)$ in $d \geq 4$ dimensions, and $\mathcal{O}\left(n \log v\right)$ in two and three dimensions. Hence, our work generalizes the results in [54] to the context of p-skylines. Our solution differs significantly from [54], as we show how to exploit the semantics of prioritized preferences for devising an effective divide-and-conquer strategy. Additionally, we prove the algorithm is $\mathcal{O}\left(n\right)$ in the average case. Our second contribution is a testing framework for p-skyline algorithms. The problem is challenging, since their performance depends on many factors that are related to both preferences and data. With respect to preferences, we show how to sample them uniformly, with the purpose of running unbiased benchmarks. With respect to data, we show how to highlight the effect of data correlation on performance. We conclude our work presenting extensive experimental results on real-life and synthetic data. Apart from the nice asymptotic properties, our algorithm proves to be practical for processing realistically sized data sets. For our evaluations we use data sets with up to one million records and up to 20 attributes. To the best of our knowledge our benchmarks are in line with most of the studies of skyline queries in the literature.

2.3 Output-Sensitive P-Skylines

In this section we present our output-sensitive algorithm for p-skyline queries. We first introduce a simple divide and conquer algorithm, named Dc, showing the problem is $\mathcal{O}\left(n \cdot \log^{d-2} n\right)$. Later we extend it, making it output-sensitive and ensuring an asymptotic worst-case complexity of $\mathcal{O}\left(n \cdot \log^{d-2} v\right)$. Before discussing our algorithms in detail, we need to generalize the concept of screening to the context of p-skylines:

Definition 3. *Given a p-expression π and two relation instances B and W, such that $W \not\succeq_\pi B$,* p-screening *is the problem of detecting all tuples in W that are not*

dominated by any tuple in B, according to $\succ_\pi$.

Extending the notation of [58], we denote by $\left[\begin{smallmatrix} B \\ W \end{smallmatrix}\right]_\pi$ the problem of p-screening B and W, and by $F_d^*(b, w)$ its worst-case complexity[2]. In Appendix B.1.2 we will show $F_d^*(b, w)$ is $\mathcal{O}\left((b + w) \cdot (\log b)^{d-2}\right)$. For the moment we take this result as given. We denote by $C_d^*(v, n)$ the worst-case complexity of p-skylines, where n and v are the input- and the output-size.The complete code of Dc (Algorithm 1) is listed below.

Algorithm 1: DC (divide & conquer)

```
   Input: a p-expression π, a relation instance D_0
   Output: the p-skyline M_π(D_0)
 1 Procedure Dc(π, D_0)
 2 |  return DcRec(π, D_0, Roots_π, ∅)
 3 Procedure DcRec(π, D, C, E)
 4 |  if C = ∅ or |D| ≤ 1 then
 5 |  |  return D
 6 |  select an attribute A from the candidates set C
 7 |  if all tuples in D assign the same value to A then
 8 |  |  E' ← E ∪ {A}
 9 |  |  C' ← C \ {A}
10 |  |  C'' ← C' ∪ {V ∈ Succ_π(A) : Pre_π(V) ⊆ E'}
11 |  |  return DcRec(π, D, C'', E')
12 |  else
13 |  |  (B, W, m*) ← SplitByAttribute(D, A)
14 |  |  B' ← DcRec(π, B, C, E)
15 |  |  W' ← PScreen(π, B', W, C \ {A}, E)
16 |  |  W'' ← DcRec(π, W', C, E)
17 |  |  return B' ∪ W''
18 Procedure SplitByAttribute(D, A)
19 |  Select m* as the median w.r.t. ≻_A in D
20 |  Compute the set B = {t ∈ D | t ≻_A m*}
21 |  Compute the set W = {t ∈ D | m* ≽_A t}
22 |  return (B, W, m*)
```

Similarly to the divide and conquer algorithm by [59], the strategy of Dc is to split the input data set into two chunks, B and W, so that no tuple in W dominates or is indistinguishable from any tuple in B ($W \not\succeq_\pi B$). The algorithm then computes recursively the p-skyline of B, $M_\pi(B)$, and performs the p-screening of W against

[2] As in Section 2.1.1: $d = |\mathcal{V}ar(\pi)|$, $b = |B|$ and $w = |W|$.

$M_\pi(B)$, i.e. it prunes all tuples in W that are dominated by some element of $M_\pi(B)$. Eventually, it recursively computes the p-skyline of the tuples that survived the p-screening, and returns it, together with $M_\pi(B)$, the p-skyline of B.

In order to understand how Dc works, it is important to understand how the input data set is split into B and W: the goal is to ensure that no tuple in B is dominated by (or indistinguishable from) any tuple in W. The general idea is to select an attribute from $\mathcal{V}ar(\pi)$, say $\mathbf{A}$, and compute the median tuple m^* w.r.t. $\succ_{\mathbf{A}}$, over the entire data set; then we can put in B all tuples that assign to $\mathbf{A}$ a better value than the one assigned by m^*, and put in W all the other tuples. If we want to be sure that $W \not\succeq_\pi B$ holds, we need to make sure that the preference on attribute $\mathbf{A}$ is not overridden by some other, higher-priority attribute. Hence, we need to choose $\mathbf{A}$ so that all tuples in both B and W agree w.r.t. all the attributes in $\mathcal{A}\text{nc}_\pi(\mathbf{A})$. In order to choose $\mathbf{A}$ wisely, Dc keeps track of two sets of attributes, namely $\mathcal{E}$ and $\mathcal{C}$, ensuring the following invariants hold:

I1 : If an attribute belongs to $\mathcal{E}$ then no pair of tuples in D can disagree on the value assigned to it. In other words: all tuples in D are indistinguishable w.r.t. all attributes in $\mathcal{E}$.

I2 : An attribute in $\mathcal{A} \setminus \mathcal{E}$ belongs to $\mathcal{C}$ if and only if all its ancestors belong to $\mathcal{E}$.

Clearly, attribute $\mathbf{A}$ is always chosen from $\mathcal{C}$. Let's see how Dc works in more detail: at the beginning $\mathcal{E}$ is empty and $\mathcal{C}$ contains all the root nodes of Γ^r_π; at every iteration the algorithm selects some attribute $\mathbf{A}$ from $\mathcal{C}$ and finds the median tuple m^* in D w.r.t. $\succ_{\mathbf{A}}$. If all tuples in D assign to $\mathbf{A}$ the same value, the algorithm updates $\mathcal{C}$ and $\mathcal{E}$ accordingly and recurs (lines 8-11). Otherwise (lines 13-17) m^* is used to split D into B and W: B contains all the tuples preferred to m^*, W those that are indistinguishable from or dominated by m^*, according to $\succ_{\mathbf{A}}$. Clearly $W \not\succeq_{\mathbf{A}} B$ holds by construction, and $W \not\succeq_\pi B$ is a direct consequence of invariants

I1 and I2. Hence, the algorithm can compute $M_\pi(B)$ recursively (line 14), perform the p-screening $\left[{}^{M_\pi(B)}_{W}\right]_\pi$ (line 15), and compute the p-skyline of the remaining tuples (line 16). The procedure PSCREEN at line 15 will be discussed in Appendix B.1.2.

Example 3. *Let's see how* DC *would determine the p-skyline for the data set of Example 1, with respect to the p-expression* $\pi = (\mathbf{P} \ \& \ \mathbf{T}) \otimes \mathbf{M}$. *The p-graph of* π *contains three nodes and only one edge, from* $\mathbf{P}$ *to* $\mathbf{T}$. *The following diagram shows the invocation trace of the procedure* DCREC. *Each box represents an invocation, its input, output and a short explanation of the actions performed. For the lack of space we omit the invocations that process only one tuple.*

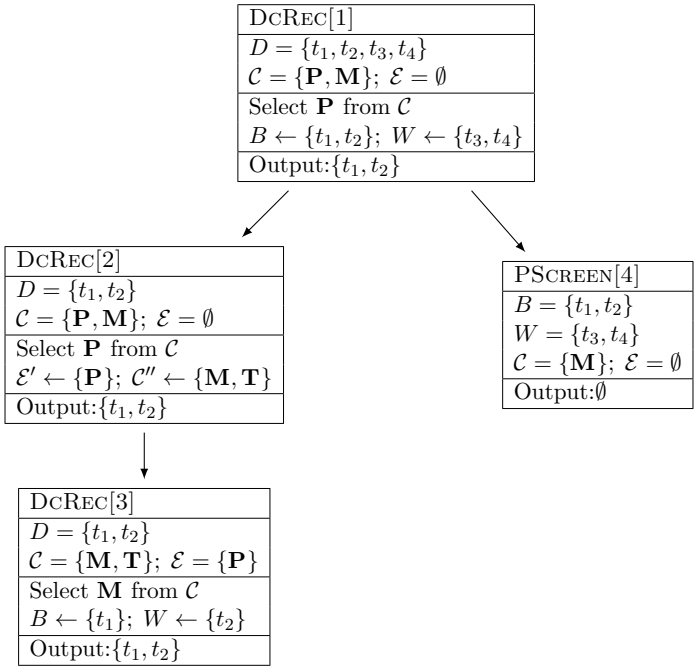


Figure 2.2: Execution of the procedure DCREC.

During the first invocation, the original set of cars is split into two halves, with respect to price. *The algorithm then recurs on the first half,* $\{t_1, t_2\}$, *in order to compute its p-skyline. The second invocation performs no work, except updating* $\mathcal{C}$ *and* $\mathcal{E}$*: attribute* $\mathbf{P}$ *is moved from* $\mathcal{C}$ *to* $\mathcal{E}$, *and attribute* $\mathbf{T}$ *enters* $\mathcal{C}$. *The third invocation computes the p-skyline of* $\{t_1, t_2\}$, *splitting w.r.t.* mileage *(no tuple is pruned). Back to the first invocation of* DCREC, *the procedure* PSCREEN *is used for removing from* $\{t_3, t_4\}$ *all*

tuples dominated by some element in $\{t_1, t_2\}$*. Both* t_3 *and* t_4 *are pruned. Since no tuple survived the screening, the algorithm directly returns* $\{t_1, t_2\}$ *without making an additional recursive call.*

The complexity analysis for Dc (Algorithm 1) is straightforward. If we denote by $T(n)$ its running time and we assume $|B| \simeq |W|$ at every iteration, the following upper bound holds for some fixed constant k_0

$$T(n) \leq k_0 n + 2 \cdot T\left(\frac{n}{2}\right) + F^*_{d-1}\left(\frac{n}{2}, \frac{n}{2}\right)$$

where the linear term $k_0 n$ models the time spent by the SplitByAttribute procedure; notice the p-screening operation at line 14 doesn't need to take attribute **A** into consideration, hence only $d-1$ attributes need to be taken into account. Since we assumed $F^*_d(b, w)$ is $\mathcal{O}\left((b+w) \cdot (\log b)^{d-2}\right)$, we can apply the master theorem [10, 23] and conclude that

$$T(n) \leq \mathcal{O}\left(n \cdot \log^{d-2} n\right)$$

We show now how to make Dc (Algorithm 1) output-sensitive. Before moving to the algorithm, we introduce some basic complexity results for $C^*_d(v, n)$ when $v = 1$ and for $F^*_d(b, w)$ when $b = 1$.

Lemma 1. *Given a relation instance* $D \subseteq U$ *and a p-expression* π*, locating a single, arbitrary element of* $M_\pi(D)$ *takes linear time.*

Proof. We can use an arbitrary weak order extension of $\succ_\pi$ and locate a maximal element $p^* \in D$ in linear time. It is easy to see p^* must belong to $M_\pi(D)$. From now on, we will denote this procedure as PSkylineSP(π, D). □

As a corollary to Lemma 1 we can state that $C^*_d(1, n) \leq \mathcal{O}(n)$. A similar result holds for p-screening:

Lemma 2. $F^*_d(1, w) \leq \mathcal{O}(w)$ *for any positive* d *and* w*.*

Proof. If B contains only one element, then we can easily perform p-screening in linear time: we only need one dominance test for each element of W. From now on, we will refer to this procedure as $\text{PScreenSP}(\pi, B, W)$. □

Now that we have defined the two procedures PScreenSP and PSkylineSP, we can introduce our output-sensitive algorithm, Osdc (listed as Algorithm 2).

Algorithm 2: Osdc (output-sensitive divide & conquer)

Input: a p-expression π, a relation instance D_0
Output: the p-skyline $M_\pi(D_0)$

```
1  Procedure Osdc(π, D_0)
2  |  return OsdcRec(π, D_0, Roots_π, ∅)
3  Procedure OsdcRec(π, D, C, E)
4  |  if C = ∅ or |D| ≤ 1 then
5  |  |  return D
6  |  select an attribute A from the candidates set C
7  |  if all tuples in D assign the same value to A then
8  |  |  E' ← E ∪ {A}
9  |  |  C' ← C \ {A}
10 |  |  C'' ← C' ∪ {V ∈ Succ_π(A) : Pre_π(V) ⊆ E'}
11 |  |  return OsdcRec(π, D, C'', E')
12 |  else
13 |  |  (B, W, m*) ← SplitByAttribute(D, A)
14 |  |  p*  ← PSkylineSP(π, B)
15 |  |  B'  ← PScreenSP(π, {p*}, B \ {p*})
16 |  |  W'  ← PScreenSP(π, {p*}, W)
17 |  |  B'' ← OsdcRec(π, B', C, E)
18 |  |  W'' ← PScreen(π, B'', W', C \ {A}, E)
19 |  |  W''' ← OsdcRec(π, W'', C, E)
20 |  |  return {p*} ∪ B'' ∪ W'''
```

The divide and conquer strategy is similar to the one in Dc (Algorithm 1), except for the look-ahead procedure at lines 14-16: at each recursion the algorithm spends linear time to extract a single p-skyline point p^* and prune from both B and W all tuples dominated by it. As a consequence, if at some point in the execution either B or W contains only a single p-skyline point, then either B' or W' will be empty, and the corresponding recursive call (either line 17 or line 19) will terminate immediately (line 5). Algorithm 2 can be used for proving the following theorem:

Theorem II.1. *$C_d^*(v,n)$ is $\mathcal{O}\left(n \cdot \log^{d-2} v\right)$.*

The proof of Theorem II.1 is given in Appendix B.1.1.

2.4 Average-case Analysis

The average-case performance for regular skyline algorithms has been studied extensively in several papers [35, 9, 11]. The most usual assumption is that the data is distributed so that each attribute is independent from the others (*statistical independence*) and the probability of two tuples agreeing on the same value for the same attribute is negligible (*sparseness*). These two assumptions are usually called, collectively, *component independence* (or CI, as in [11]). Under the CI assumption LESS has been shown to be $\mathcal{O}(n)$ in the average-case, while SFS resulted to be $\mathcal{O}(n \log n)$ [35]. In this section we show how to modify OSDC, in order to ensure a linear average-case complexity. We start by making two key observations:

Observation 1 Over all the preference relations that can be expressed using p-expressions, the skyline relation $\succ_{\text{sky}}$ represents a worst-case scenario for an output-sensitive algorithm like OSDC. This follows directly from the containment property (Proposition 2): it is easy to see that for every $\succ_\pi$ we have that $M_\pi(D) \subseteq M_{\text{sky}}(D)$ holds as long as $\mathcal{V}ar(\pi) = \mathcal{V}ar(sky)$.

Observation 2 Buchta [14] proved that under the CI assumption the expected size of $M_{\text{sky}}(D)$ is $H_{d-1,n}$, the $(d-1)$-th order harmonic of n. Godfrey [34] observed that if we drop the assumption of sparseness the size of $M_{\text{sky}}(D)$ is likely to *decrease*.

Hence, computing regular skylines over data sets respecting the CI assumption is a corner-case scenario for an output-sensitive p-skyline algorithm, in the sense that introducing priorities between attributes or dropping the sparseness assumption would only improve the algorithm's performance. In order to show that our procedure is well-

behaved in the average-case, we will prove it is well-behaved under the assumptions discussed above. More specifically, we will show how to make a simple modification to OSDC and ensure an average performance of $\mathcal{O}(n)$ for skyline queries under the CI assumption. The general idea is to follow a two-stage approach, similar to the one proposed in [9]:

1. During the first phase a linear scan prunes all the points dominated by a virtual tuple t^*, that is chosen so that the probability that no point dominates it is less than $1/n$, and the average number of points not dominated by it is $o(n)$. Such t^* can be chosen using the strategy presented in [9].
2. With probability $(n-1)/n$ the virtual tuple t^* is dominated by a real tuple in D, and so the algorithm can compute the final answer by running OSDC only on the $o(n)$ points that survived the initial linear scan.
3. With probability $1/n$ the final answer needs to be computed by running OSDC over the whole data set D.

It is easy to see that the amortized, average cost of this procedure is $\mathcal{O}(n)$, while the worst-case complexity is still $\mathcal{O}\left(n \log^{d-2} v\right)$.

2.5 P-Skylines in External Memory

In the past scan-based skyline algorithms like BNL, SFS, SALSA or LESS have generated a considerable interest in the database community. While all these algorithms exhibit a sub-optimal worst-case performance of $\mathcal{O}(n^2)$, they are known to be well-behaved in the average-case scenario, and, for larger values of d, to be even more practical than divide-and-conquer algorithms like [11] (we refer the reader to [35] for an exhaustive discussion of the topic). Additionally, scan-based algorithms are easy to embed into existing database systems, as they are easily pipelinable, and support

execution in external memory (where they still exhibit a sub-optimal, quadratic worst-case performance, as discussed in [72]). In this section we show how to adapt two scan-based skyline algorithms, SFS and LESS, in order to support p-skyline queries. Our goal is twofold: on one hand we want to show that OSDC is faster than the scan-based solutions, a part for its nice asymptotic properties (this will be done in the Section 2.6); on the other hand we want to develop a p-skyline algorithm that supports execution in external memory.

Both SFS and LESS sort the input dataset so that no tuple can dominate another preceding it; to achieve a similar result with prioritized preferences, we propose to presort the input w.r.t. the following weak order extension of $\succ_\pi$

$$\succ_\pi^{\text{ext}} = \succ_{\text{sum}_0} \ \& \ \succ_{\text{sum}_1} \ \& \ \ldots \ \& \ \succ_{\text{sum}_{d-1}} \tag{2.3}$$

Each $\succ_{\text{sum}_i}$ is defined as follows:

$$t' \succ_{\text{sum}_i} t \iff \sum_{\mathbf{A} \in \mathcal{V}ar(\pi): d_{\mathbf{A}} = i} t'[\mathbf{A}] < \sum_{\mathbf{A} \in \mathcal{V}ar(\pi): d_{\mathbf{A}} = i} t[\mathbf{A}]$$

That is, $t' \succ_{\text{sum}_i} t$ holds when the sum over all attributes at depth i computed for t' is lower than the same sum computed for t. After the sorting step, if $t' \succ_\pi^{\text{ext}} t$ holds tuple t' is going to be processed before tuple t.

Theorem II.2. *The relation $\succ_\pi^{ext}$ defined above is a weak order extension of $\succ_\pi$.*

Proof. First we want to show that for each pair of tuples (t', t) in U^2, $t' \succ_\pi^{\text{ext}} t$ implies $t \not\succ_\pi t'$. We can denote by i^* the smallest index i such that $(t' \succ_{\text{sum}_{i^*}} t)$. Since the sum over all attributes at depth i^* is smaller for t' rather than for t, there must be at least one attribute $\mathbf{A}$ at depth i^* favoring t' over t. It is easy to see that such $\mathbf{A}$ belongs to $\mathcal{T}op_\pi(t', t)$; from Proposition 1, point 2, we can conclude that $t \not\succ_\pi t'$. We are left to prove that $\succ_\pi^{\text{ext}}$ is a weak order: this follows directly from (2.3), noting

that each $\succ_{\text{sum}_i}$ is a weak order, and the prioritized composition of weak orders is a weak order itself. □

2.6 Experimental Results

To the best of our knowledge there is no published work on measuring p-skyline queries performance. The problem is difficult, since the response time depends on many factors, including the topology of the p-graph and data properties like size, correlation and likelihood of duplicated values.

In the following sections we try to address this issue by proposing a novel p-skyline testing framework. First we show how to sample random p-expressions from a uniform distribution, and how to generate meaningful synthetic data sets. Later we present our experimental results from both real and synthetic data sets.

2.6.1 Sampling random p-expressions

P-expressions can encode a wide variety of preferences: they can represent lexicographic orders, classical skylines, or any combination of the two. In order to keep evaluations fair and unbiased we should not polarize benchmarks on specific preferences. Instead, our goal is to randomly sample p-expressions from a uniform distribution, ensuring all preferences are equally represented.

Given the number of attributes d, sampling a random p-expression means building a random p-graph over d vertices, ensuring that all legal p-graphs have the same probability of being generated. We present a result from [64] to characterize the set of p-graphs we want to sample from.

Theorem II.3. *[64] Given a set of d attributes $\mathcal{A}$, a graph Γ over $\mathcal{A}$ is a p-graph if and only if:*

1. *Γ is transitive and irreflexive.*

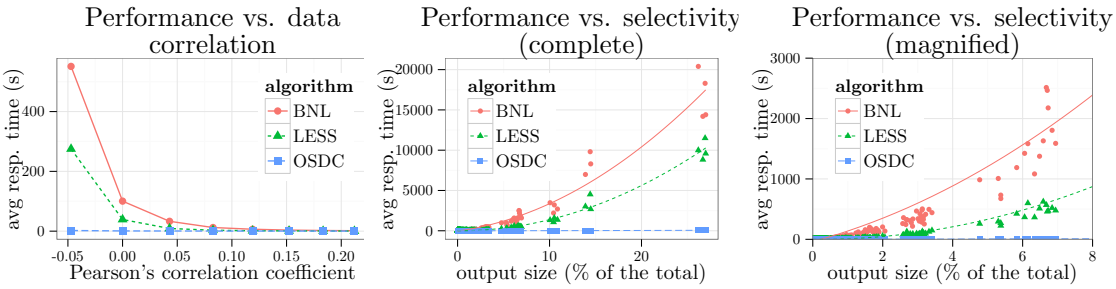


Figure 2.3: The effect of data correlation and query selectivity on performance (synthetic data sets)

2. Γ *respects the* envelope *property:*

$$\forall \mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_3, \mathbf{A}_4 \textit{ all different in } \mathcal{A},\ (\mathbf{A}_1, \mathbf{A}_2) \in \Gamma \wedge (\mathbf{A}_3, \mathbf{A}_4) \in \Gamma \wedge (\mathbf{A}_3, \mathbf{A}_2) \in \Gamma$$
$$\Rightarrow (\mathbf{A}_3, \mathbf{A}_1) \in \Gamma \vee (\mathbf{A}_1, \mathbf{A}_4) \in \Gamma \vee (\mathbf{A}_4, \mathbf{A}_2) \in \Gamma.$$

Iterating over all graphs that respect the above constraints is practical only for small values of d. For larger values we use the following strategy: we convert the constraints into a boolean satisfaction problem and sample from its solutions near-uniformly using SampleSAT [78]. SampleSAT performs a random walk over the possible solutions of a SAT problem, alternating greedy WalkSAT moves with simulated annealing steps. The ratio of the frequencies of the two kinds of steps, denoted by f, determines the trade-off between the uniformity of the sampling and the time spent to obtain it. For our tests on synthetic data we used $f = 0.5$ for generating 200 p-expressions, with d ranging from 5 to 20 attributes.

2.6.2 Synthetic data sets

Several papers, starting from [12], showed how data correlation affects the performance of skyline queries. We wanted to test whether similar considerations apply to p-skylines. Notice the role of correlation in this context is subtle: depending whether two variables have the same priority or not, a correlation between them may have different effects. For this reason we decided to generate synthetic data sets where each pair of dimensions exhibits approximatively the same linear correlation. Let's

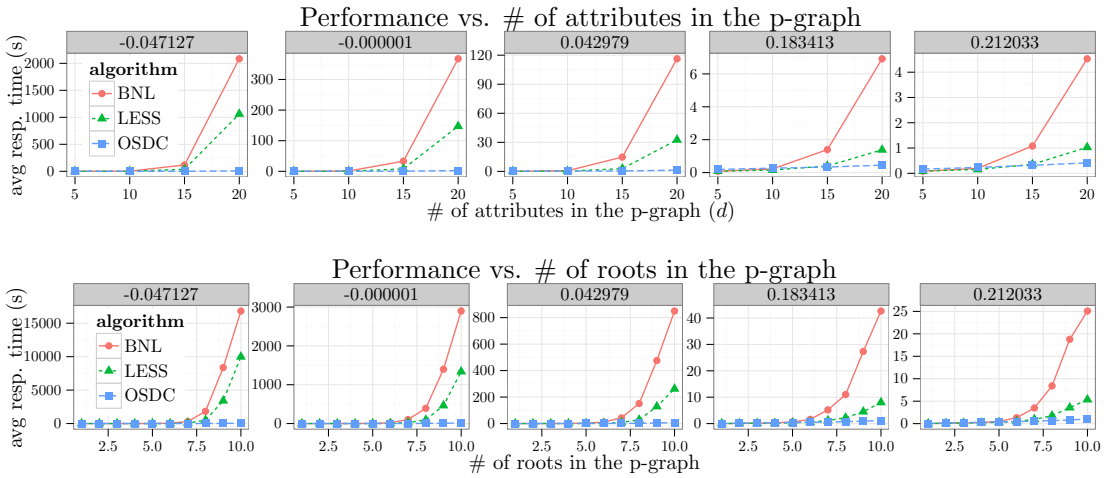


Figure 2.4: The effect of the p-graph's topology on performance (synthetic data sets)

denote by $\mathbf{1}$ a d-dimensional all-ones vector $(1, 1, \ldots, 1)$, and by M a $d \times d$ matrix whose rows form an orthonormal basis for $\mathbb{R}^d$, the first one being parallel with $\mathbf{1}$. Notice that M represents a rotation centered on the origin. Let M_D be a $d \times d$ diagonal matrix, having $(\alpha, 1, \ldots, 1)$ as its main diagonal. We propose to test p-skyline algorithms over a multivariate Gaussian distribution $\mathcal{N}_\alpha$, centered on the origin, with covariance matrix $\Sigma_\alpha = M \times M_D \times M^{-1}$. According to this distribution each pair of distinct dimensions exhibits the same correlation, determined by the parameter α. It is important to notice that $\mathcal{N}_\alpha$, amongst all the distributions where all pairs of variables have the same correlation, is the one with *maximum entropy* given the parameter α.

For our tests we sampled several data sets, varying the value of α; each set contains one million tuples over $d = 20$ attributes. Since p-skylines make sense only when some tuples agree on some attributes, we rounded the data off to four decimal digits of precision, in order to ensure the presence of duplicated values. As a result, the uncorrelated data sets (those with $\alpha = 1$) have approximatively $7,000$ distinct values in each column. We compared the performance of Osdc against Bnl and Less. To keep the comparison fair we implemented an in-memory version of Bnl, setting

the size of the window to be large enough to store the whole input. This way, the algorithm could answer each query with a single iteration. We adapted Less using the strategy discussed in Section 2.5. We ran it using several different thresholds on the size of the elimination filter, ranging between 50 and $10,000$ tuples. For each experiment we report only the fastest response times. In order to avoid any overhead, we precomputed the ranking $\succ_{\pi}^{\text{ext}}$. All the algorithms were implemented using Java, and tested on an Intel Core i7-2600 (3.4 GHz) machine equipped with 8 GB of RAM. We ran all the experiments using the Java Runtime Environment version 1.7.0, limiting the maximum heap size to 4 GB.

On the average we observed Osdc to be significantly faster than Less and Bnl. Here we analyze these results in relation with data correlation, and we study how the topology of p-expressions affects the performance of each algorithm. Figure 2.3 (left) focuses on the effect of data correlation. We average the response time over all queries, and plot it against the observed Pearson's correlation coefficient[3]. The Figure shows that Less and Bnl compete with Osdc in presence of positive data correlation, but their performance decreases quickly on anti-correlated data. Osdc, on the other hand, remains mostly unaffected by data correlation.

In Figure 2.4 we investigate the relation between performance and the topology of p-graphs. We group queries according to the number of attributes (top) and roots (bottom) in their p-graphs, and we aggregate response times w.r.t. data correlation. For lack of space we report only five levels of correlation, the most significant ones. Independently from data correlation, Osdc exhibits a distinct performance advantage on queries with more than 10 attributes, especially if there are more than five roots; Less shows a similar advantage in presence of positive data correlation, while Bnl results are competitive mostly on queries with less than five roots. During our experiments we observed that both p-graph topology and data correlation have a direct

[3]The correlation coefficient was measured after rounding the data sets.

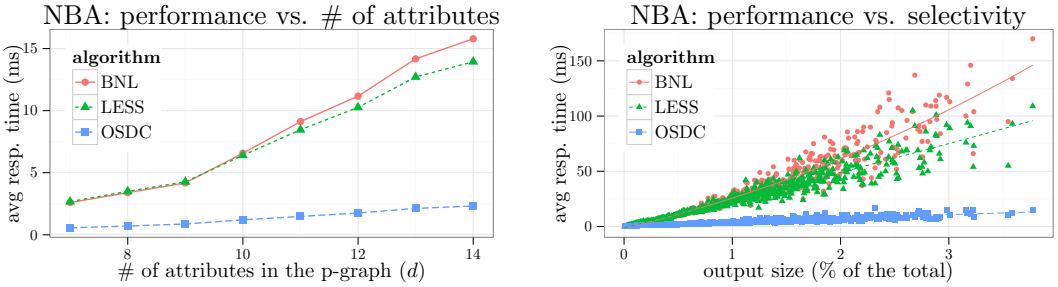


Figure 2.5: NBA data set (21,959 tuples over 14 attributes)

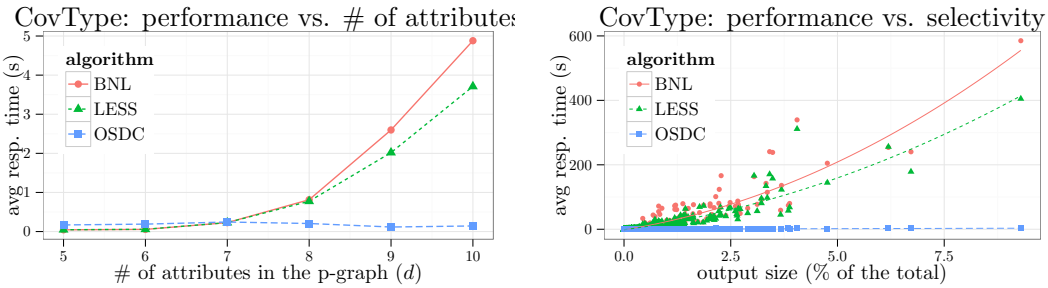


Figure 2.6: CovType data set (581,012 tuples over 10 attributes).

influence on the size of the output: highly-prioritized p-expressions (those with few roots) are likely to produce smaller p-skylines; similarly, positively correlated data is likely to produce smaller result-sets. Therefore, we summarize our results by plotting the average response time against the size of the output (Figure 2.3, on the right). As expected, Osdc and Less show a clear advantage for large result-sets while Bnl remains competitive only for queries returning few tuples. The lines on the graph represent second-order polynomial regressions.

2.6.3 Real data sets

We tested our algorithms over the following real, publicly available data sets:

NBA NBA[4] is a very popular data set for evaluating skyline algorithms. We used the following regular season statistics: *gp*, *minutes*, *pts*, *reb*, *asts*, *stl*, *blk*, *turnover*, *pf*, *fga*, *fta*, *tpa*, *weight*, *height*. After dropping null values, the data set contains

[4]www.databasebasketball.com

21,959 tuples. We generated 8,000 random p-expressions with d ranging from 7 to 14. For this data set we used the assumption that larger values are preferred.

CovType Forest Covertype[5] contains a collection of cartographic observations performed by the US Forest Service and the US Geological Survey. We extracted a data set of 581,012 tuples over 10 attributes. We generated 6,000 random p-expressions with d ranging from 5 to 10. For this data set we used the assumption that smaller values are preferred.

Our results are presented in Figures 2.5 and 2.6. In the graphs on the left response times are aggregated by the number d of attributes in each p-expression. In the plots on the right response times are put in relation with the size of the output. On both data sets our findings confirmed our average-case analysis and the results we obtained from synthetic data: OSDC outperforms LESS and BNL, especially when the output-size is over 1% of the input-size.

[5] archive.ics.uci.edu/ml/datasets/Covertype

CHAPTER III

Beta Probabilistic Databases

Tuple-independent probabilistic databases (TI-PDBs) handle uncertainty by annotating each tuple with a probability parameter; when the user submits a query, the database derives the marginal probabilities of each output-tuple, assuming input-tuples are statistically independent. While query processing in TI-PDBs has been studied extensively, limited research has been dedicated to the problems of updating or deriving the parameters from observations of queries' results. *Addressing this problem is the main focus of this chapter. We introduce* Beta Probabilistic Databases *(B-PDBs), a generalization of TI-PDBs designed to support both (i) belief updating and (ii) parameter learning in a principled and scalable way. The key idea of B-PDBs is to treat each parameter as a latent, Beta-distributed random variable. We show how this simple expedient enables both belief updating and parameter learning in a principled way, without imposing any burden on regular query processing. As a result, B-PDBs can be trained by simply observing query-answers, without requiring the user to provide the model's parameters explicitly.*

3.1 Introduction

Uncertain data arises in numerous settings, including data exchange, ETL, approximate query processing, and more. In the last decade, the challenge of posing

queries over uncertain data – data specified by a probability distribution has received considerable attention from the database community [4, 13, 16, 24, 29, 37, 44, 46, 50, 69, 71, 74]. In short, querying uncertain data is relatively well understood. However, deriving the probability distribution behind a probabilistic database can be significantly harder.

In this chapter, we address the challenge of building a probabilistic database from a noisy, indirect signal. Specifically, we propose a new model for probabilistic databases called Beta-Probabilistic Databases (B-PDBs) that enables principled approaches for deriving or updating the database's distribution. The information used to update or derive the B-PDB may be indirect — a B-PDB can incorporate any information that can be expressed as the output of a query over the database. This information may also be noisy — the information incorporated into a B-PDB may itself be sampled, as in a poll or a vote. Most importantly, B-PDBs are completely backwards compatible with the widely used Tuple-Independent model [31, 13, 24, 25] for probabilistic databases (TI-PDBs), allowing us to freely leverage query processing techniques developed for TI-PDBs, like pRA [31], Monte Carlo simulations [16, 46, 49], anytime approximations [30], dissociations [33], lineage-based methods [38] and more.

In the Tuple-Independent model, a database is a set of n tuples $\{x_1, \ldots, x_n\}$, annotated with independent probabilities $\{\theta_1, \ldots, \theta_n\}$. It represents a standard relational database whose internal state is uncertain; the set of its plausible states (its "possible worlds") consists of the power-set of $\{x_1, \ldots, x_n\}$. The probability of a possible world w is simply the probability of selecting its tuples independently: $\mathbb{P}[w] = \prod_{x_i \in w} \theta_i \cdot \prod_{x_j \notin w} (1 - \theta_j)$. Given a Boolean query q, the probability of q being true is equal to the sum of the probabilities of the possible worlds that satisfy q: $\mathbb{P}[q] = \sum_{w \models q} \mathbb{P}[w]$.

Example 4. *A TI-PDB may be used to encode noisy knowledge about the employment history of a set of candidates:*

R^p			
name	emp	*tid*	$\boldsymbol{\theta}$
Ada	*HP*	x_1	.6
Ada	*IBM*	x_2	.6
Bob	*HP*	x_3	.5

If we want to find out whether the person named Ada ever had an employer, we run the following Boolean query

$$\texttt{exists(select * from R}^p\texttt{ where name='Ada')} \tag{3.1}$$

The answer is expected to be true ("Ada had at least one employer in the past") with probability 0.84 *and false ("Ada never had a job") with probability* 0.16.[1]

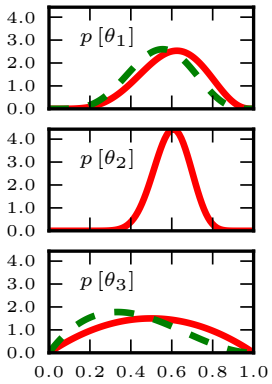
Let's now assume we are given (i) a TI-PDB $\mathcal{D}$ whose parameters are hidden, (ii) a set of Boolean queries $\mathcal{Q} \stackrel{\text{def}}{=} \{q_1, \ldots, q_k\}$, and (iii) a finite set of query-answers sampled from $\mathbb{P}[q]$, for each q in $\mathcal{Q}$. We denote by $\mathcal{E}$ (the "evidence") the whole set of samples and we assume each sample is drawn independently from all the others. We focus mainly on two problems:

1. **Belief updates**: given an initial hypothesis about the hidden parameters of $\mathcal{D}$, we show how to refine such hypothesis as to incorporate the evidence $\mathcal{E}$.

2. **Parameter learning**: we show how to derive a new hypothesis from scratch, relying only on the given evidence.

Without lack of generality, we assume each query is observed exactly s times, and we denote by τ the fraction of positive answers. Therefore, we model the evidence $\mathcal{E}$ as a mapping that associates each query to its observed relative frequency of positive answers: $\mathcal{E} \stackrel{\text{def}}{=} \{(q_1, \tau_1), (q_2, \tau_2), \ldots, (q_k, \tau_k)\}$.

[1]The probabilities follow from $1 - [(1 - 0.6) \cdot (1 - 0.6)] = 0.84$. Also notice that we adopt here a closed-world assumption as is common with TI-PDBs: any missing tuple is assumed to have probability 0.



before observing $\neg(x_1 \vee x_3)$

name	emp	*tid*	a	b	$\langle\boldsymbol{\theta}\rangle$	$h[\boldsymbol{\theta}]$
Ada	HP	x_1	6	4	.6	−0.507
Ada	IBM	x_2	18	12	.6	−1.014
Bob	HP	x_3	2	2	.5	−0.125

after observing $\neg(x_1 \vee x_3)$

name	emp	*tid*	a	b	$\langle\boldsymbol{\theta}\rangle$	$h[\boldsymbol{\theta}]$
Ada	HP	x_1	6	5	.54	−0.528
Ada	IBM	x_2	18	12	.6	−1.014
Bob	HP	x_3	2	3	.4	−0.235

Figure 3.1: A simple B-PDB with three tuples, *before* (solid red) and *after* (dashed green) observing the answer "no" to the query `exists(select * from R where emp='HP')` .

Example 4 (continued)**.** *If we crawled the web and retrieved 25 LinkedIn profiles that are all plausible, equally likely matches for the entity named Ada, and all of them but 4 report some unspecified work experience, then we can associate the relative frequency* $\tau = 0.84$ *to query (3.1), and set* $s = 25$*. Evidence for other queries can be collected in a similar fashion.*

Belief updating is useful when someone wants to improve an already reliable probabilistic model, exploiting some new, previously unseen, evidence. For example, let's assume we trust the information stored in relation R^{p}, but we want to improve our knowledge about Ada's work history. In order to do so, we submit the query "has Ada ever worked for IBM?"[2] to 10 independent data banks. If at least 7 of them answer "yes", then we may want to increase parameter θ_1 in R^{p} (whose initial value is 0.6) as to reflect this additional information. Clearly, the extent of the adjustment will depend on how strong our initial confidence about the value of θ_1 was. Belief updating is about computing these adjustments to prior beliefs in a principled way.

Parameter learning is about using the experts' opinion to build a new probabilistic

[2]In SQL: `exists(select * from R where name='Ada' and emp='IBM')`.

model from scratch. For example: if we are told that the query

$$\mathbf{q} = \texttt{select emp from R where name='Ada'}$$

should return the answer $\{\text{HP}, \text{IBM}\}$ ("Ada worked for both HP and IBM") with relative frequency 0.32 (or some other answer with relative frequency $1-0.32 = 0.68$), and should return the empty set $\emptyset$ ("Ada has not worked before") with relative frequency 0.22 (or some other non-empty answer with relative frequency $1 - 0.22 = 0.78$), then our goal becomes to choose θ_1 and θ_2 so that the model (R^{p}) exhibits the desired marginal probabilities for $\mathbf{q}$. This is achieved either when $(\theta_1, \theta_2) = (0.4, 0.8)$ or when $(\theta_1, \theta_2) = (0.8, 0.4)$.[3]

The key idea behind B-PDBs is to model the parameters $\{\theta_1, \ldots, \theta_n\}$ as Beta-distributed latent random variables. This simple expedient allows us to (i) model *both* our current estimate of a probability and its confidence in a natural way, and (ii) to deploy a principled way to update those estimates in the presence of new evidence. We illustrate with Figure 3.1 (first table) a simple B-PDB, consisting of a single relation, generalizing the TI-relation R^{p} we introduced in Example 4. Notice that for each tuple x_i the parameter θ_i has been replaced by *two* parameters, a_i and b_i. The symbol θ_i is now used to identify a $[0, 1]$-valued random variable, whose probability density is Beta-distributed:

$$p[\theta_i] \stackrel{\text{def}}{=} \theta_i^{a_i - 1} \cdot (1 - \theta_i)^{b_i - 1} \cdot B(a_i, b_i)^{-1} \tag{3.2}$$

Here $B(\cdot, \cdot)$ denotes the Beta function, which serves as a normalizing factor[4] . The three solid-red plots on the left in fig. 3.1 depict the density functions $p[\theta]$ for each

[3]Since $0.32 = 0.8 \cdot 0.4$ and $0.22 = 1 - (1 - 0.8)(1 - 0.6)$. Notice that $\mathbf{q}$ is not a Boolean query. Nonetheless, the example is well defined: it is possible to write a Boolean query to verify whether the answer to $\mathbf{q}$ is $\{\text{HP}, \text{IBM}\}$, and another one to verify whether the answer is $\emptyset$.

[4]Chapter 25 of [48] is a good introduction to Beta distributions.

tuple in the B-PDB. An intuitive way to understand Beta distributions is to think about their parameters a and b as votes. Under this interpretation, the first row in the B-PDB represents a poll where the query "has Ada ever worked for HP?" has received 6 positive answers and 4 negative ones. Similarly, the second row can be seen as a poll where the query "has Ada ever worked for IBM?" has received 18 positive votes and 12 negative ones. While the relative frequency of positive answers is the same for both the polls (0.6), the second poll should be considered more informative, as it involves more votes (30 against 10). Consistently with this intuition, the plots of $p[\theta_1]$ and $p[\theta_2]$ have both a peak on .6, but the former exhibits a higher entropy than the latter.

In B-PDBs the marginal probability of a tuple x_i is determined by the *expected value* of the random variable θ_i:

$$\mathbb{P}[x_i] = \int_0^1 \theta_i \cdot p[\theta_i]\, d\theta_i \tag{3.3}$$

From now on we denote with $\langle\theta_i\rangle$ the expected value of θ_i. It is well known [48] that the integral in eq. (3.3) admits the following closed solution: $\mathbb{P}[x_i] = \langle\theta_i\rangle = a_i/(a_i{+}b_i)$. It follows that B-PDBs are *indistinguishable* from regular TI-PDBs when it comes to query processing: all existing inference techniques, both exact and approximate, that have been proposed in the past for TI-PDBs [25, 30, 33, 49, 66, 67] can be readily applied to B-PDBs. To do so it is sufficient to convert the B-PDB into a TI-DB by computing the expectation $\langle\theta_i\rangle$ for each and every tuple in the database, in polynomial time. Therefore, B-PDBs are a conservative generalization of TI-PDBs that add a principled way to update the parameters, which is the main focus of our chapter. The first table of fig. 3.1 shows the tuples' marginal probabilities inside column $\langle\boldsymbol{\theta}\rangle$. This column is shown for readers' convenience only; it is not explicitly stored in a B-PDB. It is immediate to verify that the given B-PDB is equivalent, in

terms of query processing, to the relation R^{p} introduced in Example 4.

Beyond estimating tuples' probabilities B-PDBs can also evaluate the *confidence* of such estimates. In the remainder of this chapter we adopt the *differential entropy* $h[\theta_i]$ as a metric for the confidence of the B-PDB's estimates of $\mathbb{P}[x_i]$. By definition, the differential entropy of θ_i is:

$$h[\theta_i] \stackrel{\text{def}}{=} \int p[\theta_i] \cdot \log(p[\theta_i]) d\theta_i$$

The above integral admits a well-known [61, 28] closed form[5]. Intuitively, the smaller $h[\theta_i]$ is the higher is the confidence of the estimate of $\mathbb{P}[x_i]$. For readers' convenience fig. 3.1 shows the differential entropies in the column $h[\boldsymbol{\theta}]$[6].

In the following sections we will describe extensively how B-PDBs support belief updates. We introduce the topic here with a simple example. The second table in fig. 3.1 shows the effect of performing a belief update to incorporate the observation of a single, negative answer to the Boolean query

```
exists(select * from R where emp='HP')
```

Intuitively, the observation suggests that neither Ada nor Bob have worked in the past for HP. We react to this new information by adding a negative vote to both the first and the third tuple in the B-PDB. The adjusted probability densities of θ_1 and θ_3 are plotted on the left, in dashed green. Notice that the update has a greater impact on $p[\theta_3]$ rather than on $p[\theta_1]$, consistently with the fact that $h[\theta_1] < h[\theta_3]$. In the general case, belief updates may involve thousands of queries, and affect the parameters of a B-PDB in a non-trivial way.

[5]The closed solution is $h[\theta_i] = \log(B(a_i, b_i)) - [(a_i - 1) \cdot (\psi(a_i) - \psi(a_i + b_i))] - [(b_i - 1) \cdot (\psi(b_i) - \psi(a_i + b_i))$, where $\psi(\cdot)$ denotes the Digamma function.

[6]As before, this information is given for readers' convenience only and does not need to be stored inside the B-PDB.

In the remainder of this chapter, we study (i) belief updates (§3.4) and (ii) parameter learning (§3.5) in great detail; we show how to perform both when we observe answers to an arbitrary set of conjunctive, self-join-free queries.

Our contributions include:

1. **Bayesian belief updates.** Given a B-PDB and a set of queries' results, we show how to incorporate the new evidence into the B-PDB in a *Bayesian* fashion. We analyze the complexity of computing such Bayesian updates and provide efficient algorithms for tractable classes of queries.
2. **Soft Expectation Maximization.** We devise a soft-EM algorithm for computing the maximum likelihood estimate of the parameters $\{\theta_1, \ldots, \theta_n\}$ w.r.t. some given queries' results.
3. **Benchmarks.** We show how the algorithms we propose can be easily embedded into a standard relational engine, so to exploit its optimization features. We test our framework against a standard industry benchmark (TCP-H), annotated with probabilities.

3.2 Background

In this section, we review some background notions and contextualize B-PDBs w.r.t. previous work on probabilistic databases. For the sake of conciseness, we use the following notation: given a real number z we denote by $\overline{z}$ its complement $(1-z)$. When φ is a Boolean expression $\overline{\varphi}$ denotes, as usual, its negation $\neg\varphi$. A comprehensive notation table is given in Appendix A.

3.2.1 Relational Databases

A relational database consists of a finite collection of relations $\{R, S, T, \ldots\}$, over a finite set of n tuples $\{x_1, \ldots, x_n\}$. A conjunctive query $\mathbf{q}$ is a first-order formula in

prenex normal form, respecting the following restrictions: (i) each predicate symbol represents a relation, (ii) all variables are either existentially quantified or quantifier-free, (iii) the formula is negation-free and (iv) disjunction-free. We use capital letters to denote first-order logic variables. For example:

$$\mathbf{q}(Z) = \exists X \; \exists Y \; R(X, Y) \wedge S(X, Z) \tag{3.4}$$

We denote by $\texttt{hvar}(\mathbf{q})$ the set of free ("head-") variables of $\mathbf{q}$, and by $\texttt{evar}(\mathbf{q})$ the set of existentially quantified variables. A conjunctive query is said to be *self-join-free* iff every relation name appears at most once; it is said to be *Boolean* iff there are no free variables. Given a database instance $\mathcal{D}$, every non-Boolean conjunctive query can be seen as a collection of Boolean queries, one for each of the possible grounding of the free variables to values in their *active domain* [2]. In the remainder of this paper we assume queries are always conjunctive and self-join-free. With limited abuse of notation we will denote non-Boolean queries as vectors ($\mathbf{q}$) and Boolean ones as indexed vectors' components (q_j) that range over the groundings of $\mathbf{q}$. Given a database instance $\mathcal{D}$ and a Boolean query q, we denote by $\Phi_{\mathcal{D}}(q)$ the *lineage* [7, 15, 38] of q, a propositional Boolean formula over the alphabet $\{x_1, \ldots, x_n\}$, built according to the following recursive rules

- $\Phi_{\mathcal{D}}(q) = \Phi_{\mathcal{D}}(q'_1) \vee \ldots \vee \Phi_{\mathcal{D}}(q'_k)$ when $q = \exists X \; \mathbf{q}'$, and $\texttt{hvar}(\mathbf{q}') = \{X\}$ and $\{q'_1, \ldots, q'_k\}$ are the groundings of $\mathbf{q}'$ obtained by replacing X with one of the constants in its active domain
- $\Phi_{\mathcal{D}}(q) = \Phi_{\mathcal{D}}(q') \wedge \Phi_{\mathcal{D}}(q'')$ when $q = q' \wedge q''$
- $\Phi_{\mathcal{D}}(q) = x_i$, when q is a ground atom[7]

A lineage expression φ is said to be *read-once* iff each literal appears at most once.

[7] In this section "atoms" are atomic first-order logic formulas. For example, the query from eq. (3.4) contains two atoms, $R(X, Y)$ and $S(X, Y)$. An atom is *ground* when it has no variables: $R(\texttt{'Ada'}, \texttt{'HP'})$.

It is straightforward to extend the definition of lineage to query answers: if φ is the lineage of q then the answer $\top$ has lineage φ, while the answer $\bot$ has lineage $\overline{\varphi}$. In the following we often identify Boolean queries with their lineage. For the sake of compactness we sometimes omit the $\wedge$ symbol in lineage expressions (therefore, x_1x_2 is an abbreviation for $x_1 \wedge x_2$) and use the common Datalog notation to express conjunctive queries; for example: $\mathbf{q}(Z) :- R(X,Y), S(X,Z)$.

Given a variable X and a query $\mathbf{q}$, we denote by $\mathtt{at}(X, \mathbf{q})$ the set of $\mathbf{q}$'s atoms where X appears. Variables that appear in every atom of $\mathbf{q}$ are called *root variables*. We say that a query $\mathbf{q}$ is *hierarchical* iff, for any two existential variables (X, Y), either $\mathtt{at}(X, \mathbf{q}) \subseteq \mathtt{at}(Y, \mathbf{q})$ or $\mathtt{at}(Y, \mathbf{q}) \subseteq \mathtt{at}(X, \mathbf{q})$ or $\mathtt{at}(X, \mathbf{q}) \cap \mathtt{at}(Y, \mathbf{q}) = \emptyset$ holds.

Example 4 (continued)**.** *The query* $\mathbf{q}$ *from eq.* (3.4) *is hierarchical; the set of head-variables* $\mathtt{hvar}(\mathbf{q})$ *contains only* Z*, while* $\mathtt{evar}(\mathbf{q})$ *consists of* $\{X, Y\}$*.* X *is a root variable, but* Y *is not. Let* $\mathcal{D}$ *be a database instance where the relations* R *and* S *are defined as follows:*

R		
name	emp	*tid*
Ada	*HP*	x_1
Ada	*IBM*	x_2
Bob	*HP*	x_3

S		
name	lng	*tid*
Ada	*eng*	x_4
Bob	*eng*	x_5
Bob	*ita*	x_6

Within $\mathcal{D}$ *the active domain of* Z *is* $\{eng, ita\}$*. Therefore query* $\mathbf{q}$ *can be seen as a collection of two Boolean queries:*

$$q_1 :- R(X,Y), S(X, eng). \qquad q_2 :- R(X,Y), S(X, ita).$$

Their lineage expressions are:

$$\Phi_{\mathcal{D}}(q_1) = x_1x_4 \vee x_2x_4 \vee x_3x_5 \qquad \Phi_{\mathcal{D}}(q_2) = x_3x_6$$

The lineage of q_2 is read-once, while the lineage of q_1 is not, as the literal x_4 is used twice (we will later show how to obtain a read-once expression for q_1).

We define *query plans* as sentences respecting the following grammar:

$$P ::= R \mid \pi_X(P') \mid \sigma(P') \mid \ \bowtie [P', P'', \ldots]$$

where R denotes an arbitrary relation name and projections (π), selections (σ) and natural joins ($\bowtie$) have the usual semantics. It is straightforward to extend the notation we use for queries to query plans: if P denotes a plan, then $\texttt{hvar}(P)$ is the set of attributes in its output schema, while $\texttt{evar}(P)$ denotes the set of attributes that are projected-away. In the following we write $\pi_{-X}(P)$ as short form for $\pi_{\texttt{hvar}(P)\setminus\{X\}}(P)$. A plan P is Boolean when $\texttt{hvar}(P)$ is empty. Given a database instance $\mathcal{D}$, a non-Boolean plan P can be seen as a collection of Boolean plans $\{P_1, \ldots, P_k\}$, one for each of its output-tuples. Each plan in $\{P_1, \ldots, P_k\}$ is obtained from P by substituting its head variables by the constants of the respective output-tuple. A query plan always corresponds to exactly one query, but one query may have multiple distinct query plans. Two query plans are logically equivalent if they answer the same query. Given a database instance $\mathcal{D}$ and a Boolean plan P we denote by $\Phi_{\mathcal{D}}(P)$ the lineage of P, a Boolean expression built according to the following recursive rules:

- If P identifies a tuple, say x, then $\Phi_{\mathcal{D}}(P) = x$.
- If $P = P' \bowtie P''$ then $\Phi_{\mathcal{D}}(P) = \Phi_{\mathcal{D}}(P') \wedge \Phi_{\mathcal{D}}(P'')$.
- If $P = \pi_{\emptyset}\,(P')$ then $\Phi_{\mathcal{D}}(P) = \Phi_{\mathcal{D}}(P'_1) \vee \Phi_{\mathcal{D}}(P'_2) \vee \ldots \vee \Phi_{\mathcal{D}}(P'_k)$ assuming that $\{P'_1, P'_2, \ldots, P'_k\}$ are the plans corresponding to the output-tuples of P'.

If plan P answers query q, then $\Phi_{\mathcal{D}}(P)$ is logically equivalent to $\Phi_{\mathcal{D}}(q)$, for every $\mathcal{D}$.

Example 4 (continued)**.** *Both the following query plans*

$$P' = \pi_{-X}(\pi_{-Y}(R) \bowtie S) \qquad P'' = \pi_{-XY}(R \bowtie S)$$

compute the correct answer for query $\mathbf{q}$ *from eq.* (3.4)*, but they produce different lineage expressions:*

P'	
`lng`	$\Phi_{\mathcal{D}}$
eng	$((x_1 \vee x_2)\ x_4) \vee x_3x_5$
ita	x_3x_6

P''	
`lng`	$\Phi_{\mathcal{D}}$
eng	$x_1x_4 \vee x_2x_4 \vee x_3x_5$
ita	x_3x_6

Notice that all the lineage expressions produced by P' *are read-once and logically equivalent to the corresponding lineage expressions of* $\mathbf{q}$ *and* P''*.*

3.2.2 Tuple-independent Probabilistic Databases

A tuple-independent probabilistic database (TI-PDB) is a regular relational database where each tuple represents an independent probabilistic event. Each tuple x_i is associated with a Bernoulli-distributed Boolean random variable, expected to be true with probability θ_i and false with probability $\overline{\theta_i}$. It represents the belief that tuple x_i belongs to the database. In slight abuse of notation we use x_i to denote both a tuple and its associated Boolean random variable; we use the vector notation $\boldsymbol{\theta}$ to denote the whole set of parameters $\{\theta_1, \ldots, \theta_n\}$. Unlike deterministic databases, the state of a TI-PDB is uncertain: the set of its plausible states (its "possible worlds") ranges over the power-set of its tuples. Hence, a possible world consists of a subset of tuples, generated by including each tuple x_i with probability θ_i. A TI-PDB $\mathcal{D}$ defines a probability measure $\mathbb{P}[\cdot]$ over possible worlds and Boolean queries. If w is a possible world, we denote by $w[i]$ a function that returns 1 when tuple x_i belongs to w, and 0 otherwise. The probability of w is $\mathbb{P}[w|\mathcal{D}] \stackrel{\text{def}}{=} \prod_{i:w[i]=1} \theta_i \cdot \prod_{i:w[i]=0} \overline{\theta_i}$, the probability of drawing its tuples independently; if q is a Boolean query, its marginal probability is the sum of all possible worlds where q is satisfied: $\mathbb{P}[q|\mathcal{D}] \stackrel{\text{def}}{=} \sum_{Asst(\models)q} \mathbb{P}[w]$. If φ is a lineage expression, we denote by $\mathbb{P}[\varphi|\mathcal{D}]$ the probability of φ being satisfied, given that each literal x_i is true with probability θ_i and false otherwise. Notice that

$\mathbb{P}[q|\mathcal{D}] = \mathbb{P}[\varphi|\mathcal{D}]$ when $\varphi = \Phi_{\mathcal{D}}(q)$.

TI-PDBs are often associated with *Probabilistic Relational Algebra* (pRA) [31], a generalization of positive relational algebra that consists of three probabilistic operators: independent projection (π^{p}), independent join ($\bowtie^{\mathsf{p}}$) and selection (σ). These operators differ from standard relational algebra in the fact that they associate a score to each output tuple. Let P be the Boolean plan associated with an arbitrary output tuple; its score is computed according to the following recursive rules:

- If P identifies a tuple x_i then $\mathsf{score}(P) = \theta_i$
- If $P = P' \bowtie^{\mathsf{p}} P''$ then $\mathsf{score}(P) = \mathsf{score}(P') \cdot \mathsf{score}(P'')$
- If $P = \sigma(P')$ then $\mathsf{score}(P) = \mathsf{score}(P')$
- If $P = \pi^{\mathsf{p}}_{\emptyset}(P')$ then

$$\mathsf{score}(P) = 1 - [(1 - \mathsf{score}(P'_1)) \cdot \ldots \cdot (1 - \mathsf{score}(P'_k))]$$

  assuming that $\{P'_1, \ldots, P'_k\}$ are the plans corresponding to the output-tuples of P'. For the sake of conciseness we adopt the *independent-or* ($\otimes$) operator:

$$\mathsf{score}(P) \stackrel{\text{def}}{=} \mathsf{score}(P'_1) \otimes \ldots \otimes \mathsf{score}(P'_k) \stackrel{\text{def}}{=} \bigotimes_{i \in \{1,\ldots,k\}} [\mathsf{score}(P'_i)]$$

Let's assume P' and P'' are two plans answering the Boolean queries q' and q'', respectively, and $\mathbb{P}[q'|\mathcal{D}] = \mathsf{score}(P')$ and $\mathbb{P}[q''|\mathcal{D}] = \mathsf{score}(P'')$. Notice that $\mathbb{P}[q' \wedge q''|\mathcal{D}] = \mathsf{score}(P' \bowtie^{\mathsf{p}} P'')$ holds, but only if q' and q'' represent independent events. Similar considerations apply to π^{p}: if P' and P'' are the output-tuples of the plan P, then the equivalence $\mathbb{P}[q' \vee q''|\mathcal{D}] = \mathsf{score}(\pi^{\mathsf{p}}_{\emptyset}(P))$ holds only if q' and q'' represent independent events. The probabilistic independence between q' and q'' is guaranteed when their lineages do not share any literal. We can conclude that an arbitrary pRA plan P computes the correct marginal probabilities only when all its intermediate

results consist of pairwise independent events. A plan respecting such property is said to be *"safe"* and its lineage expressions are guaranteed to be read-once. The following Lemma summarizes a variety of results about probabilistic query processing over TI-PDBs

Lemma 3. *[24, 25, 36, 66] Let* $\mathbf{q}$ *be a self-join-free conjunctive query consisting of* k *Boolean queries* $\{q_1, \ldots, q_k\}$*. The following statements are equivalent:*

1. *Query* $\mathbf{q}$ *is* hierarchical.
2. *For any* $\mathcal{D}$*, query* $\mathbf{q}$ *admits a* safe *pRA plan.*
3. *For any* $\mathcal{D}$ *and* $q_j \in \mathbf{q}$*, the lineage of* q_j *admits a read-once representation.*
4. *For any* $\mathcal{D}$ *and* $q_j \in \mathbf{q}$*, computing* $\mathbb{P}[q_j|\mathcal{D}]$ *takes polynomial time in the size of* $\mathcal{D}$*.*

Deciding any (all) of the above properties (finding a certificate, if any exists) takes polynomial time in the size of $\mathbf{q}$*. If such test fails (i.e. no certificate exists), then anwering* $\mathbf{q}$ *is #P-hard.*

Example 4 (continued)**.** *We can turn the relations* R *and* S *into a TI-PDB by annotating each tuple with a probability, that we store in a dedicated column named* $\boldsymbol{\theta}$*.*

R^p				
name	emp	*tid*	$\boldsymbol{\theta}$	
Ada	*HP*	x_1	0.6	
Ada	*IBM*	x_2	0.6	
Bob	*HP*	x_3	0.5	

S^p			
name	lng	*tid*	$\boldsymbol{\theta}$
Ada	*eng*	x_4	0.4
Bob	*eng*	x_5	0.2
Bob	*ita*	x_6	0.6

We can rewrite the plans P' *and* P'' *in terms of pRA:*

$$P' = \pi^p_{-X}(\pi^p_{-Y}(R^p) \bowtie^p S^p) \qquad P'' = \pi^p_{-XY}(R^p \bowtie^p S^p) \tag{3.5}$$

They both produce the same output-tuples, but different scores:

P'	
lng	*score*
eng	$((\theta_1 \otimes \theta_2)\ \theta_4) \otimes \theta_3\theta_5 = 0.4024$
ita	$\theta_3\theta_6 = 0.3$

P''	
lng	*score*
eng	$\theta_1\theta_4 \otimes \theta_2\theta_4 \otimes \theta_3\theta_5 = 0.48016$
ita	$\theta_3\theta_6 = 0.3$

Notice that plan P' is safe, while P'' is not: the correct value of $\mathbb{P}[q_1|\mathcal{D}]$ is 0.4024*, it is* not 0.48016*.*

In conclusion, pRA is guaranteed to be sound only when dealing with hierarchical queries. Dalvi and Suciu [24] developed a well-known algorithm to identify safe plans for hierarchical queries. Other techniques, like [16, 46, 49, 30], must be used to answer non-hierarchical queries.

3.3 Beta Probabilistic Databases

In this section we introduce *Beta probabilistic databases* (B-PDBs), our new generalization of TI-PDBs, based on the idea of imposing a prior distribution over the parameters $\boldsymbol{\theta}$. In the resulting model, each parameter θ_i becomes an independent random variable, whose probability density function follows a Beta distribution $\mathcal{B}e(a_i, b_i)$ determined by two hyper-parameters, a_i and b_i:

$$p[\theta_i|\mathcal{H}] \stackrel{\text{def}}{=} \mathcal{B}e(a_i, b_i) \tag{3.6}$$

$$p[\boldsymbol{\theta}|\mathcal{H}] \stackrel{\text{def}}{=} \prod_{i=1}^{n} p[\theta_i|\mathcal{H}] \tag{3.7}$$

We denote by $\mathcal{B}e(a, b)$ the probability density function of a Beta distribution:

$$\mathcal{B}e(a, b) \stackrel{\text{def}}{=} \theta^{a-1} \cdot \overline{\theta}^{b-1} \cdot B(a, b)^{-1}$$

We use $\mathbf{a}$ and $\mathbf{b}$ to denote the corresponding n-vectors of hyper-parameters, and $\mathcal{H} \stackrel{\text{def}}{=} (\mathbf{a}, \mathbf{b})$ to denote a Beta-database instance (the relational structure of $\mathcal{H}$ is

assumed to be fixed and, for the sake of conciseness, it is never mentioned explicitly). In terms of graphical models, one can see a B-PDB as a collection of n independent Boolean variables, distributed according to n independent *Beta-Bernoulli compound distributions*:

$$\theta_i \sim \text{Beta}(a_i, b_i) \qquad x_i \sim \text{Bernoulli}(\theta_i)$$

Given an arbitrary function $f(\theta)$ and a distribution $p[\theta]$ we denote by $\langle f \rangle_{p[\theta]}$ the expected value of f, assuming θ is sampled from $p[\theta]$. Just like TI-PDBs, B-PDBs define a probability measure over possible worlds and Boolean queries:

$$\mathbb{P}[x_i|\mathcal{H}] \stackrel{\text{def}}{=} \int_0^1 \theta_i \cdot \mathcal{B}e(a_i, b_i)\, d\theta_i \tag{3.8}$$

$$\mathbb{P}[Asst\,(|)\,\mathcal{H}] \stackrel{\text{def}}{=} \prod_{i=1}^{n} \mathbb{P}[x_i|\mathcal{H}]^{Asst(|)i]} \cdot \overline{\mathbb{P}[x_i|\mathcal{H}]}^{\overline{Asst(|)i]}} \tag{3.9}$$

$$\mathbb{P}[\varphi|\mathcal{H}] \stackrel{\text{def}}{=} \sum_{Asst(:)\ Asst(\models)\varphi} \mathbb{P}[w|\mathcal{H}] \tag{3.10}$$

Equations 3.8, 3.9 and 3.10 denote, respectively, the probability of a literal, the probability of a possible world, and the probability of a Boolean query. Notice that φ may represent the lineage of the answer to a non-Boolean query. For example, if we submit a non-Boolean query $\mathbf{q}$ consisting of three Boolean queries $[\varphi_1, \varphi_2, \varphi_3]$, the probability of observing the answer $[\bot, \top, \bot]$ is $\mathbb{P}[\overline{\varphi}_1\varphi_2\overline{\varphi}_3|\mathcal{H}]$.

In practical terms, B-PDBs differ from TI-PDBs in that each tuple is annotated with *two* $\mathbb{R}^+$-valued parameters, rather than with a single probability measure (compare, for example, the table at the top of Figure 3.1 with the probabilistic relation R^{p} discussed in Example 4). Notice that the marginal probability of x_i can be computed as [48]

$$\mathbb{P}[x_i|\mathcal{H}] = \int_0^1 \theta_i \cdot \mathcal{B}e(a_i, b_i)\, d\theta_i = \langle \theta_i \rangle_{\mathcal{H}} = \tfrac{a_i}{a_i+b_i} \tag{3.11}$$

From eq. (3.11) it follows immediately that vector $\langle \boldsymbol{\theta} \rangle_{\mathcal{H}}$ of expected tuple probabilities

under $\mathcal{H}$, represents the parameters of a TI-PDB that behave identically to the B-PDB $\mathcal{H}$ when it comes to query processing. In other words, the mapping $\mathcal{H} \to \langle \boldsymbol{\theta} \rangle_{\mathcal{H}}$ allows us to immediately re-use all the standard query processing techniques developed for TI-PDBs, like pRA [31], Monte Carlo simulations [16, 46, 49], anytime approximations [30], dissociations [33], lineage-based methods [38] and many others.

Given two queries φ and φ', we denote by $\mathbb{P}[\varphi|\varphi', \mathcal{H}]$ the probability of observing φ being true in a possible world of $\mathcal{H}$ that already satisfies φ':

$$\mathbb{P}[\varphi|\varphi', \mathcal{H}] \stackrel{\text{def}}{=} \frac{\mathbb{P}[\varphi \wedge \varphi'|\mathcal{H}]}{\mathbb{P}[\varphi'|\mathcal{H}]} \tag{3.12}$$

We denote by $p[\boldsymbol{\theta}|\varphi, \mathcal{H}]$ the *posterior* probability density function of $\boldsymbol{\theta}$ w.r.t. φ:

$$p[\boldsymbol{\theta}|\varphi, \mathcal{H}] \stackrel{\text{def}}{=} \frac{p[\boldsymbol{\theta}, \varphi|\mathcal{H}]}{\mathbb{P}[\varphi|\mathcal{H}]} = \frac{\mathbb{P}[\varphi|\boldsymbol{\theta}] \cdot p[\boldsymbol{\theta}|\mathcal{H}]}{\mathbb{P}[\varphi|\mathcal{H}]} \tag{3.13}$$

Notice that $\mathbb{P}[\varphi|\boldsymbol{\theta}]$ represents the probability of observing φ being satisfied by a TI-PDB with parameters $\boldsymbol{\theta}$.

3.3.1 Multiple independent observations

Given a B-PDB $\mathcal{H}$ and a positive integer s, we denote by $\mathcal{H}^s$ the distribution obtained by replicating the model of $\mathcal{H}$ exactly s times. In other words, $\mathcal{H}^s$ represents the distribution of a set of s possible worlds drawn independently from $\mathcal{H}$. Figure 3.2 depicts model $\mathcal{H}^s$ using plate notation, and compares it with the model induced by TI-DBs. Within a model $\mathcal{H}^s$, we denote by $x_{\ell,i}$ and $\theta_{\ell,i}$ the pairs of random variables associated with the i-th tuple of the ℓ-th possible world. Therefore

$$\theta_{\ell,i} \sim \text{Beta}(a_i, b_i) \qquad x_{\ell,i} \sim \text{Bernoulli}(\theta_{\ell,i})$$

We denote by $x_{(\cdot,i)}$ the s-vector $(x_{1,i},\ldots,x_{s,i})$, by $x_{(\ell,\cdot)}$ the n-vector $(x_{\ell,1},\ldots,x_{\ell,n})$ and by $x_{(\cdot,\cdot)}$ the s-by-n matrix containing all the Boolean random variables of the model. We adopt similar conventions for defining the semantics of $\theta_{(\cdot,i)}$, $\theta_{(\ell,\cdot)}$ and $\theta_{(\cdot,\cdot)}$. Given an integer t such that $0 \leq t \leq s$, we denote by $\mathbb{P}[\varphi^t|\mathcal{H}^s]$ the probability of observing a set of s independent possible worlds from $\mathcal{H}$ where φ is satisfied exactly t times:

$$\mathbb{P}[\varphi^t|\mathcal{H}^s] \stackrel{\text{def}}{=} \binom{s}{t} \cdot \mathbb{P}[\varphi|\mathcal{H}]^t \cdot \mathbb{P}[\overline{\varphi}|\mathcal{H}]^{s-t}$$

The posterior probability density of $\theta_{(\cdot,\cdot)}$ w.r.t. $\mathcal{H}^s$ and evidence φ^t is

$$p[\theta_{(\cdot,\cdot)}|\varphi^t,\mathcal{H}^s] \stackrel{\text{def}}{=} \frac{\binom{s}{t}\prod_{\ell=1}^{t}\mathbb{P}[\varphi|\theta_{(\ell,\cdot)}]\cdot\prod_{\ell=t+1}^{s}\mathbb{P}[\overline{\varphi}|\theta_{(\ell,\cdot)}]\cdot\prod_{\ell=1}^{s}p[\theta_{(\ell,\cdot)}|\mathcal{H}]}{\mathbb{P}[\varphi^t|\mathcal{H}^s]}$$

Notice that the above formula generalizes eq. (3.13). Given a positive integer k, we denote by $\mathcal{H}^{s,k}$ the probability distribution obtained by replicating $\mathcal{H}^s$ exactly k times. Equivalently, $\mathcal{H}^{s,k}$ represents the distribution of a set of $s \cdot k$ possible worlds sampled independently from $\mathcal{H}$. We extend our notation accordingly, denoting by $x_{j,\ell,i}$ and $\theta_{j,\ell,i}$ the random variables associated with the $(j \cdot \ell)$-th possible world. Given a set of k distinct Boolean expressions $\{\varphi_1,\ldots,\varphi_k\}$ and k integers $\{t_1,\ldots,t_k\}$ between 0 and s, we denote by $\mathcal{E} = \{\varphi_1^{t_1},\ldots,\varphi_k^{t_k}\}$ the event of observing each φ_j in $\{\varphi_1,\ldots,\varphi_k\}$ being satisfied exactly t_j over the s possible worlds $\{x_{(j,1,\cdot)},\ldots,x_{(j,s,\cdot)}\}$. Its likelihood is

$$\mathbb{P}[\mathcal{E}|\mathcal{H}^{s,k}] \stackrel{\text{def}}{=} \prod_{j=1}^{k}\mathbb{P}[\varphi_j^{t_j}|\mathcal{H}^s]$$

The posterior probability density of $\theta_{(\cdot,\cdot,\cdot)}$ w.r.t. $\mathcal{H}^{s,k}$ and evidence $\mathcal{E} = \{\varphi_1^{t_1},\ldots,\varphi_k^{t_k}\}$ is

$$p[\theta_{(\cdot,\cdot,\cdot)}|\mathcal{E},\mathcal{H}^{s,k}] \stackrel{\text{def}}{=} \prod_{j=1}^{k}p[\theta_{(j,\cdot,\cdot)}|\varphi_j^{t_j},\mathcal{H}^s]$$

Example 5. *Let's assume we have a B-PDB $\mathcal{H}$ with two tuples, x_1 and x_2, and $k = 2$ queries: $\varphi_1 = x_1 \wedge x_2$ and $\varphi_2 = x_1 \vee x_2$. Then:*

- $\mathbb{P}[\varphi_1 \wedge \varphi_2|\mathcal{H}] = \langle\theta_1\rangle_{\mathcal{H}} \cdot \langle\theta_2\rangle_{\mathcal{H}}$
- $\mathbb{P}[\varphi_1|\varphi_2, \mathcal{H}] = (\langle\theta_1\rangle_{\mathcal{H}} \cdot \langle\theta_2\rangle_{\mathcal{H}}) \cdot (\langle\theta_1\rangle_{\mathcal{H}} \otimes \langle\theta_2\rangle_{\mathcal{H}})^{-1}$
- $\mathbb{P}[\varphi_1|\mathcal{H}^2] = \frac{1}{2} \cdot (\langle\theta_1\rangle_{\mathcal{H}} \cdot \langle\theta_2\rangle_{\mathcal{H}}) \cdot (\langle\overline{\theta_1}\rangle_{\mathcal{H}} \otimes \langle\overline{\theta_2}\rangle_{\mathcal{H}})$
- $\mathbb{P}[\varphi_1, \varphi_2|\mathcal{H}^{1,2}] = (\langle\theta_1\rangle_{\mathcal{H}} \cdot \langle\theta_2\rangle_{\mathcal{H}}) \cdot (\langle\theta_1\rangle_{\mathcal{H}} \otimes \langle\theta_2\rangle_{\mathcal{H}})$

From eq. (3.11) it is straightforward to derive the following Lemma:

Lemma 4. *For any arbitrary evidence $\mathcal{E} = \{\varphi_1^{t_1}, \ldots, \varphi_k^{t_k}\}$ the likelihood function $\mathbb{P}[\mathcal{E}|\mathcal{H}^{s,k}]$ respects the following properties:*

1. *If $\mathcal{H}_a$ and $\mathcal{H}_b$ are two B-PDBs such that $\langle\boldsymbol{\theta}\rangle_{\mathcal{H}_a} = \langle\boldsymbol{\theta}\rangle_{\mathcal{H}_b}$ then $\mathbb{P}[\mathcal{E}|\mathcal{H}_a] = \mathbb{P}[\mathcal{E}|\mathcal{H}_b]$.*

2. *Let $\mathcal{H}$ be a B-PDB and $\mathcal{D}$ a TI-PDB with parameters $\boldsymbol{\theta}^*$: if $\boldsymbol{\theta}^* = \langle\boldsymbol{\theta}\rangle_{\mathcal{H}}$ then $\mathbb{P}[\mathcal{E}|\mathcal{H}^{s,k}] = \mathbb{P}[\mathcal{E}|\mathcal{D}^{s,k}]$, where $\mathcal{D}^{s,k}$ denotes the distribution obtained by drawing $s \cdot k$ independent samples from $\mathcal{D}$.*

The next two sections are dedicated to two specific operations supported by B-PDBs, that involve the computation of the posterior $p[\theta_{(\cdot,\cdot,\cdot)}|\mathcal{E}, \mathcal{H}^{s,k}]$: *belief updating* and *maximum likelihood estimation*. We introduce their formal definition here:

Definition 4 (Belief updating)**.** *Given a B-PDB $\mathcal{H}$ and an evidence event $\mathcal{E} = \{\varphi_1^{t_1}, \ldots, \varphi_k^{t_k}\}$, belief updating is the process of identifying the B-PDB $\hat{\mathcal{H}}$ that minimizes the relative entropy between the posterior $p[\theta_{(\cdot,\cdot,\cdot)}|\mathcal{E}, \mathcal{H}^{s,k}]$ and the prior $p[\theta_{(\cdot,\cdot,\cdot)}|\hat{\mathcal{H}}^{s,k}]$. Belief updating is discussed in §3.4.*

Definition 5 (Maximum likelihood estimation)**.** *Given some evidence $\mathcal{E} = \{\varphi_1^{t_1}, \ldots, \varphi_k^{t_k}\}$, maximum likelihood estimation is the problem of identifying a local maximum of the likelihood function $\mathbb{P}[\mathcal{E}|\mathcal{H}^{s,k}]$. Maximum likelihood estimation is discussed in §3.5.*

3.4 Belief Updating

The goal of belief updating is to adjust the parameters **a** and **b** so to incorporate some new, previously unseen, evidence. If $\mathcal{H}$ denotes the current state of the our

B-PDB and $\mathcal{E} = \{\varphi_1^{t_1}, \ldots, \varphi_k^{t_k}\}$ is the new evidence we observe, our goal is to identify a new state $\hat{\mathcal{H}} \stackrel{\text{def}}{=} (\hat{\mathbf{a}}, \hat{\mathbf{b}})$ that minimizes the relative entropy between $p[\theta_{(\cdot,\cdot,\cdot)}|\mathcal{E}, \mathcal{H}^{s,k}]$ and $p[\theta_{(\cdot,\cdot,\cdot)}|\hat{\mathcal{H}}]$. For the sake of clarity, we start introducing belief updates from the simple case where $s = k = 1$ and $\mathcal{E} = \{\varphi\}$. Later we will extend our discussion to the general case.

3.4.1 Simple case: $s = k = 1$

When $s = k = 1$ holds and $\mathcal{E} = \{\varphi\}$, $\hat{\mathcal{H}}$ is supposed to minimize the relative entropy between the posterior $p[\boldsymbol{\theta}|\varphi, \mathcal{H}]$ and the prior $p[\boldsymbol{\theta}|\hat{\mathcal{H}}]$. Since every tuple of $\hat{\mathcal{H}}$ is independent from the others, it is sufficient to minimize the relative entropy between $p[\theta_i|\varphi, \mathcal{H}]$ and $p[\theta_i|\hat{\mathcal{H}}]$ for each and every tuple. The first step in this direction is to compute the marginal posterior $p[\theta_i|\varphi, \mathcal{H}]$. The following theorem addresses this specific problem.

Theorem III.1. *The marginal posterior probability of random variable θ_i, given hypothesis $\mathcal{H}$ and evidence $\mathcal{E} = \{\varphi\}$, can be computed as follows:*

$$p[\theta_i|\varphi, \mathcal{H}] = \mathbb{P}[x_i|\varphi, \mathcal{H}] \cdot \mathcal{B}e(a_i + 1, b_i) + \mathbb{P}[\overline{x_i}|\varphi, \mathcal{H}] \cdot \mathcal{B}e(a_i, b_i + 1)$$

Proof. The proof is given in **??**. □

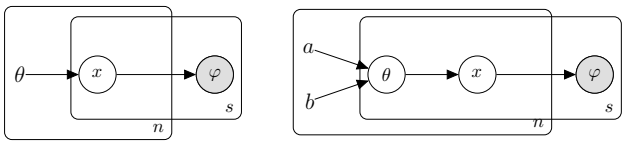


Figure 3.2: Comparison between a regular TI-PDB (left) and a B-PDB (right) when $k = 1$, using plate notation. TI-PDBs associate each tuple x_i with a single Boolean, Bernoulli-distributed random variable, whose probability mass function depends on a single parameter (θ_i). B-PDBs associate each tuple with *two* random variables: θ_i, which is $[0, 1]$-valued and Beta-distributed with hyper-parameters (a_i, b_i), and x_i, which is Boolean-valued and Bernoulli-distributed. For both TI-PDBs and B-PDBs the evidence consists of observed query answers, that here are modeled by the observable variable φ.

Theorem III.1 states that $p[\theta_i|\varphi, \mathcal{H}]$ is a mixture of Beta distributions. It is well known that the Beta distribution is the conjugate prior of the Bernoulli distribution. Consequently, the posterior of a Beta-Bernoulli compound distribution is guaranteed to be Beta-distributed. Within a B-PDB we can exploit such property whenever we can infer the value of some random variable x_i from the evidence; this is formalized by the following Corollary:

Corollary 1. *In a B-PDB $\mathcal{H}$ the conjugancy property holds if and only if the random variable x_i is fully observable from the evidence:*

$$p[\theta_i|\varphi, \mathcal{H}] = \begin{cases} \mathcal{B}e(a_i + 1, b_i) & \textit{if } \varphi \models x_i \\ \mathcal{B}e(a_i, b_i + 1) & \textit{if } \varphi \models \overline{x_i} \end{cases}$$

Corollary 1 suggests a very intuitive interpretation of the marginal posterior $p[\theta_i|\varphi, \mathcal{H}]$: it can be seen as a random process in which we first make a guess about the value taken by x_i in the evidence and then we select the appropriate conjugate prior. Notice that the complexity of computing $p[\theta_i|\varphi, \mathcal{H}]$ depends on the complexity of computing conditional probabilities in the form $\mathbb{P}[x|\varphi, \mathcal{H}]$. This problem is discussed extensively in §3.6; for the moment we just observe that it is $\#\mathcal{P}$-complete in the general case.

Now that we know how to compute the marginal posterior $p[\theta_i|\varphi, \mathcal{H}]$, we can move our attention to the problem of computing an update $\mathcal{H} \to \hat{\mathcal{H}}$ that minimizes the relative entropy between $p[\theta_i|\varphi, \mathcal{H}]$ and the marginal prior $p[\theta_i|\hat{\mathcal{H}}] \stackrel{\text{def}}{=} \mathcal{B}e(\hat{a}_i, \hat{b}_i)$. We denote such measure with $\mathsf{KL}_i^{\mathsf{div}}$:

$$\mathsf{KL}_i^{\mathsf{div}} = \int_0^1 p[\theta_i|\varphi, \mathcal{H}] \cdot \log\left(\frac{p[\theta_i|\varphi, \mathcal{H}]}{p[\theta_i|\hat{\mathcal{H}}]}\right) d\theta_i$$

Notice that $p[\theta_i|\hat{\mathcal{H}}]$ belongs to the exponential family, and its sufficient statistics are $\log \theta_i$ and $\log \overline{\theta_i}$. Therefore, if we want to minimize the relative entropy $\mathsf{KL}_{\mathsf{div}}$ we have

to choose $(\hat{a}_i, \hat{b}_i)$ so that the expected value of $(\log\theta_i, \log\overline{\theta}_i)$ w.r.t. $\mathbb{P}[\theta_i|\hat{\mathcal{H}}]$ matches the expected value of the same statistics computed w.r.t. $\mathbb{P}[\theta|\varphi, \mathcal{H}]$. This well-known criterion is formalized in the following definition and justified in Proposition 1[8]:

Definition 6. *We denote by* $\mathsf{bfit}(a_i, b_i, \varphi)$ *the pair of parameters* $(\hat{a}_i^*, \hat{b}_i^*)$ *satisfying the following two equations:*

$$\begin{cases} \int_0^1 \mathcal{B}e(\hat{a}_i^*, \hat{b}_i^*)\log\theta_i \; d\theta_i = \int_0^1 p[\theta_i|\varphi, \mathcal{H}]\log\theta_i \; d\theta_i \\ \int_0^1 \mathcal{B}e(\hat{a}_i^*, \hat{b}_i^*)\log\overline{\theta_i} \; d\theta_i = \int_0^1 p[\theta_i|\varphi, \mathcal{H}]\log\overline{\theta_i} \; d\theta_i \end{cases} \tag{3.14}$$

or, in terms of expectations:

$$\begin{cases} \langle\log\theta_i\rangle_{\mathcal{B}e(\hat{a}_i^*, \hat{b}_i^*)} = \langle\log\theta_i\rangle_{p[\theta_i|\varphi, \mathcal{H}]} \\ \langle\log\overline{\theta}_i\rangle_{\mathcal{B}e(\hat{a}_i^*, \hat{b}_i^*)} = \langle\log\overline{\theta}_i\rangle_{p[\theta_i|\mathcal{E}, \mathcal{H}]} \end{cases} \tag{3.15}$$

Proposition 1. *When* $(\hat{a}_i, \hat{b}_i) = (\hat{a}_i^*, \hat{b}_i^*) = \mathsf{bfit}(a_i, b_i, \varphi)$ *the relative entropy between the posterior* $p[\theta_i|\varphi, \mathcal{H}]$ *and the Beta distribution* $\mathbb{P}[\theta_i|\hat{\mathcal{H}}]$ *is minimized.*

[8] A similar, more general result is proved in [42] for all the distributions in the exponential family.

Proof. The relative entropy between $p[\theta_i|\varphi,\mathcal{H}]$ and $p[\theta_i|\hat{\mathcal{H}}]$ can be expressed as follows:

$$\begin{aligned}
\mathsf{KL}_i^{\mathsf{div}} &= \int_0^1 p[\theta_i|\varphi,\mathcal{H}]\cdot\log\left(\frac{p[\theta_i|\varphi,\mathcal{H}]}{p[\theta_i|\hat{\mathcal{H}}]}\right)d\theta_i \\
&= h\left[p[\theta_i|\varphi,\mathcal{H}]\right] - \int_0^1 p[\theta_i|\varphi,\mathcal{H}]\cdot\log\left(p[\theta_i|\hat{\mathcal{H}}]\right)d\theta_i \\
&= h\left[p[\theta_i|\varphi,\mathcal{H}]\right] + \log(B(\hat{a}_i,\hat{b}_i)) - \int_0^1 p[\theta_i|\varphi,\mathcal{H}]\cdot\log\left(\theta_i^{\hat{a}_i-1}\cdot\overline{\theta_i}^{\hat{b}_i-1}\right)d\theta_i \\
&= h\left[p[\theta_i|\varphi,\mathcal{H}]\right] + \log(B(\hat{a}_i,\hat{b}_i)) - \int_0^1 p[\theta_i|\varphi,\mathcal{H}]\cdot(\hat{a}_i-1)\cdot\log\theta_i\,d\theta_i + \\
&\quad - \int_0^1 p[\theta_i|\varphi,\mathcal{H}]\cdot(\hat{b}_i-1)\cdot\log\overline{\theta_i}\,d\theta_i \\
&= h\left[p[\theta_i|\varphi,\mathcal{H}]\right] + \log(B(\hat{a}_i,\hat{b}_i)) - (\hat{a}_i-1)\cdot\langle\log\theta_i\rangle_{p[\theta_i|\varphi,\mathcal{H}]} - (\hat{b}_i-1)\cdot\langle\log\overline{\theta_i}\rangle_{p[\theta_i|\varphi,\mathcal{H}]}
\end{aligned}$$

The gradient of $\mathsf{KL}_i^{\mathsf{div}}$ w.r.t. $(\hat{a}_i,\hat{b}_i)$ is:

$$\begin{aligned}
\nabla\mathsf{KL}_i^{\mathsf{div}} &= \begin{bmatrix} \frac{\partial\mathsf{KL}_i^{\mathsf{div}}}{\partial\hat{a}_i} \\ \\ \frac{\partial\mathsf{KL}_i^{\mathsf{div}}}{\partial\hat{b}_i} \end{bmatrix} = \begin{bmatrix} [\psi(\hat{a}_i)-\psi(\hat{a}_i+\hat{b}_i)] - \langle\log\theta_i\rangle_{p[\theta_i|\varphi,\mathcal{H}]} \\ \\ [\psi(\hat{b}_i)-\psi(\hat{a}_i+\hat{b}_i)] - \langle\log\overline{\theta_i}\rangle_{p[\theta_i|\varphi,\mathcal{H}]} \end{bmatrix} \\
&= \begin{bmatrix} \langle\log\theta_i\rangle_{p[\theta_i|\hat{\mathcal{H}}]} - \langle\log\theta_i\rangle_{p[\theta_i|\varphi,\mathcal{H}]} \\ \\ \langle\log\overline{\theta_i}\rangle_{p[\theta_i|\hat{\mathcal{H}}]} - \langle\log\overline{\theta_i}\rangle_{p[\theta_i|\varphi,\mathcal{H}]} \end{bmatrix}
\end{aligned}$$

The Hessian of $\mathsf{KL}_i^{\mathsf{div}}$ w.r.t. $(\hat{a}_i, \hat{b}_i)$ is positive semidefinite:

$$\mathrm{Hess}[\mathsf{KL}_i^{\mathsf{div}}] = \nabla\nabla\mathsf{KL}_i^{\mathsf{div}} = \begin{bmatrix} \psi'(\hat{a}_i) - \psi'(\hat{a}_i + \hat{b}_i) & -\psi'(\hat{a}_i + \hat{b}_i) \\ -\psi'(\hat{a}_i + \hat{b}_i) & \psi'(\hat{b}_i) - \psi'(\hat{a}_i + \hat{b}_i) \end{bmatrix}$$

Where $\psi'(\cdot)$ denotes the Trigamma function. The thesis follows immediately. □

Finally, we observe that the relative entropy between $p[\boldsymbol{\theta}|\varphi, \mathcal{H}]$ and $p[\boldsymbol{\theta}|\hat{\mathcal{H}}]$, that we denote with $\mathsf{KL}^{\mathsf{div}}$, is simply the sum of the tuple-wise KL divergences:

$$\mathsf{KL}^{\mathsf{div}} = \int .. \int p[\boldsymbol{\theta}|\varphi, \mathcal{H}] \cdot \log\left(\frac{p[\boldsymbol{\theta}|\varphi, \mathcal{H}]}{p[\boldsymbol{\theta}|\hat{\mathcal{H}}]}\right) d\boldsymbol{\theta} = \sum_{i=1}^{n} \mathsf{KL}_i^{\mathsf{div}}$$

3.4.2 General case

First we address the case where $k = 1$, $s > 1$ and $\mathcal{E} = \{\varphi^t\}$. Under these assumptions the goal of belief updating is to minimize the KL divergence between $p[\theta_{(\cdot,\cdot)}|\varphi^t, \mathcal{H}^s]$ and $p[\theta_{(\cdot,\cdot)}|\hat{\mathcal{H}}^s]$. Since the tuples of $\hat{\mathcal{H}}$ are pairwise independent, it is sufficient to minimize the relative entropy between $p[\theta_{(\cdot,i)}|\varphi^t, \mathcal{H}^s]$ and $p[\theta_{(\cdot,i)}|\hat{\mathcal{H}}^s]$ for every i in $\{1, \ldots, n\}$. As usual $\theta_{(\cdot,i)}$ denotes the vector $(\theta_{1,i}, \ldots, \theta_{s,i})$. Its probability density w.r.t. $\hat{\mathcal{H}}^s$ is

$$p[\theta_{(\cdot,i)}|\hat{\mathcal{H}}^s] = \prod_{\ell=1}^{s} p[\theta_{\ell,i}|\hat{\mathcal{H}}] \tag{3.16}$$

while its posterior density w.r.t. $\mathcal{H}^s$ and evidence $\{\varphi^t\}$ is

$$p[\theta_{(\cdot,i)}|\varphi^t, \mathcal{H}^s] = \prod_{\ell=1}^{t} p[\theta_{\ell,i}|\varphi, \mathcal{H}] \cdot \prod_{\ell=t+1}^{s} p[\theta_{\ell,i}|\overline{\varphi}, \mathcal{H}] \tag{3.17}$$

where $p[\theta_{\ell,i}|\varphi,\mathcal{H}]$ and $p[\theta_{\ell,i}|\overline{\varphi},\mathcal{H}]$ are computed according to Theorem III.1. We now redefine $\mathsf{KL}_i^{\mathsf{div}}$ as the relative entropy between $p[\theta_{(\cdot,i)}|\varphi^t,\mathcal{H}^s]$ and $p[\theta_{(\cdot,i)}|\hat{\mathcal{H}}^s]$:

$$\mathsf{KL}_i^{\mathsf{div}} = \int_0^1 .. \int_0^1 p[\theta_{(\cdot,i)}|\varphi^t,\mathcal{H}^s] \log\left(\frac{p[\theta_{(\cdot,i)}|\varphi^t,\mathcal{H}^s]}{p[\theta_{(\cdot,i)}|\hat{\mathcal{H}}^s]}\right) d\theta_{(\cdot,i)}$$

Definition 7. *We denote by* $\mathsf{bfit}(a_i, b_i, \{\varphi^t\}, s)$ *the pair of parameters* $(\hat{a}_i^*, \hat{b}_i^*)$ *satisfying the following two equations:*

$$\begin{cases} \langle \log \theta_i \rangle_{\mathcal{B}e(\hat{a}_i^*,\hat{b}_i^*)} = \frac{t}{s}\langle \log \theta_i \rangle_{p[\theta_i|\varphi,\mathcal{H}]} + \frac{s-t}{s}\langle \log \theta_i \rangle_{p[\theta_i|\overline{\varphi},\mathcal{H}]} \\ \langle \log \overline{\theta}_i \rangle_{\mathcal{B}e(\hat{a}_i^*,\hat{b}_i^*)} = \frac{t}{s}\langle \log \overline{\theta}_i \rangle_{p[\theta_i|\varphi,\mathcal{H}]} + \frac{s-t}{s}\langle \log \overline{\theta}_i \rangle_{p[\theta_i|\overline{\varphi},\mathcal{H}]} \end{cases}$$

Proposition 2. *When* $(\hat{a}_i, \hat{b}_i) = (\hat{a}_i^*, \hat{b}_i^*) = \mathsf{bfit}(a_i, b_i, \{\varphi^t\}, s)$ *the relative entropy between* $p[\theta_{(\cdot,i)}|\varphi^t,\mathcal{H}^s]$ *and* $p[\theta_{(\cdot,i)}|\hat{\mathcal{H}}^s]$ *is minimized.*

Proof.

$$\begin{aligned} \mathsf{KL}_i^{\mathsf{div}} &= \int_0^1 .. \int_0^1 p[\theta_{(\cdot,i)}|\varphi^t,\mathcal{H}^s] \log\left(\frac{p[\theta_{(\cdot,i)}|\varphi^t,\mathcal{H}^s]}{p[\theta_{(\cdot,i)}|\hat{\mathcal{H}}^s]}\right) d\theta_{(\cdot,i)} \\ &= \sum_{\ell=1}^{t}\left[\int_0^1 p[\theta_{\ell,i}|\varphi,\mathcal{H}] \log\left(\frac{p[\theta_{\ell,i}|\varphi,\mathcal{H}]}{p[\theta_{\ell,i}|\mathcal{H}]}\right) d\theta_{\ell,i}\right] + \\ &+ \sum_{\ell=t+1}^{s}\left[\int_0^1 p[\theta_{\ell,i}|\overline{\varphi},\mathcal{H}] \log\left(\frac{p[\theta_{\ell,i}|\overline{\varphi},\mathcal{H}]}{p[\theta_{\ell,i}|\mathcal{H}]}\right) d\theta_{\ell,i}\right] \\ &= h\left[p[\Theta|\varphi^t,\mathcal{H}^s]\right] + s\cdot\log(B(\hat{a}_i,\hat{b}_i)) + \\ &- (\hat{a}_i - 1)\cdot\left[t\cdot\langle\log\theta_i\rangle_{p[\theta_i|\varphi,\mathcal{H}]} + (s-t)\langle\log\theta_i\rangle_{p[\theta_i|\overline{\varphi},\mathcal{H}]}\right] \\ &- (\hat{b}_i - 1)\cdot\left[t\cdot\langle\log\overline{\theta}_i\rangle_{p[\theta_i|\varphi,\mathcal{H}]} + (s-t)\langle\log\overline{\theta}_i\rangle_{p[\theta_i|\overline{\varphi},\mathcal{H}]}\right] \end{aligned}$$

The gradient $\nabla \mathsf{KL}_i^{\mathsf{div}}$ is zero when

$$\begin{cases} \langle \log \theta_i \rangle_{p[\theta_i|\hat{\mathcal{H}}]} = \frac{t}{s} \langle \log \theta_i \rangle_{p[\theta_i|\varphi,\mathcal{H}]} + \frac{s-t}{s} \langle \log \theta_i \rangle_{p[\theta_i|\overline{\varphi},\mathcal{H}]} \\ \langle \log \overline{\theta}_i \rangle_{p[\theta_i|\hat{\mathcal{H}}]} = \frac{t}{s} \langle \log \overline{\theta}_i \rangle_{p[\theta_i|\varphi,\mathcal{H}]} + \frac{s-t}{s} \langle \log \overline{\theta}_i \rangle_{p[\theta_i|\overline{\varphi},\mathcal{H}]} \end{cases}$$

□

We can finally address the general case where $k > 1$, $s > 1$ and $\mathcal{E} = \{\varphi_1^{t_1}, \ldots, \varphi_k^{t_k}\}$. Under these assumptions the goal of belief updating is to minimize the KL divergence between $p[\theta_{(\cdot,\cdot,\cdot)}|\mathcal{E}, \mathcal{H}^{s,k}]$ and $p[\theta_{(\cdot,\cdot,\cdot)}|\hat{\mathcal{H}}^{s,k}]$, and is achieved by minimizing the KL divergence between $p[\theta_{(\cdot,\cdot,i)}|\mathcal{E}, \mathcal{H}^{s,k}]$ and $p[\theta_{(\cdot,\cdot,i)}|\hat{\mathcal{H}}^{s,k}]$, for every i in $\{1, \ldots, n\}$, where

$$p[\theta_{(\cdot,\cdot,i)}|\hat{\mathcal{H}}^{s,k}] = \prod_{j=1}^{k} p[\theta_{(j,\cdot,i)}|\hat{\mathcal{H}}^{s}] \tag{3.18}$$

$$p[\theta_{(\cdot,\cdot,i)}|\mathcal{E}, \mathcal{H}^{s,k}] = \prod_{j=1}^{k} p[\theta_{(j,\cdot,i)}|\varphi_j^{t_j}, \mathcal{H}^{s}] \tag{3.19}$$

Therefore, we can redefine $\mathsf{KL}_i^{\mathsf{div}}$ as the relative entropy between $p[\theta_{(\cdot,\cdot,i)}|\mathcal{E}, \mathcal{H}^{s,k}]$ and $p[\theta_{(\cdot,\cdot,i)}|\hat{\mathcal{H}}^{s,k}]$:

$$\mathsf{KL}_i^{\mathsf{div}} = \int_0^1 .. \int_0^1 p[\theta_{(\cdot,\cdot,i)}|\mathcal{E}, \mathcal{H}^{s,k}] \log \left(\frac{p[\theta_{(\cdot,\cdot,i)}|\mathcal{E}, \mathcal{H}^{s,k}]}{p[\theta_{(\cdot,\cdot,i)}|\hat{\mathcal{H}}^{s,k}]} \right) d\theta_{(\cdot,\cdot,i)}$$

Definition 8. *We denote by* $\mathsf{bfit}(a_i, b_i, \mathcal{E}, s, k)$ *the pair of parameters* $(\hat{a}_i^*, \hat{b}_i^*)$ *satisfying the following two equations:*

$$\begin{cases} \langle \log \theta_i \rangle_{\hat{\mathcal{H}}^*} = \sum_j \frac{t_j}{ks} \langle \log \theta_i \rangle_{p[\theta_i|\varphi_j,\mathcal{H}]} + \frac{s-t_j}{ks} \langle \log \theta_i \rangle_{p[\theta_i|\overline{\varphi}_j,\mathcal{H}]} \\ \langle \log \overline{\theta}_i \rangle_{\hat{\mathcal{H}}^*} = \sum_j \frac{t_j}{ks} \langle \log \overline{\theta}_i \rangle_{p[\theta_i|\varphi_j,\mathcal{H}]} + \frac{s-t_j}{ks} \langle \log \overline{\theta}_i \rangle_{p[\theta_i|\overline{\varphi}_j,\mathcal{H}]} \end{cases}$$

Proposition 3. *When* $(\hat{a}_i, \hat{b}_i) = \mathsf{bfit}(a_i, b_i, \mathcal{E}, s, k)$ *the relative entropy between* $p[\theta_{(\cdot,\cdot,i)}|\mathcal{E}, \mathcal{H}^{s,k}]$ *and* $p[\theta_{(\cdot,\cdot,i)}|\hat{\mathcal{H}}^{s,k}]$ *is minimized.*

The proof of Proposition 3 mimics the one of Proposition 2. For the sake of conciseness we omit it. Before introducing our belief update algorithm, we observe that the equations from Definition 8 can be rewritten as follows:

$$\begin{cases} [\psi(\hat{a}_i^*) - \psi(\hat{a}_i^* + \hat{b}_i^*)] - [\psi(a_i) - \psi(a_i + b_i)] = \frac{r}{a_i} - \frac{1}{a_i + b_i} \\ [\psi(\hat{b}_i^*) - \psi(\hat{a}_i^* + \hat{b}_i^*)] - [\psi(b_i) - \psi(a_i + b_i)] = \frac{\overline{r}}{b_i} - \frac{1}{a_i + b_i} \end{cases} \tag{3.20}$$

where $r = \frac{1}{k} \sum_{j=1}^{k} \left[\frac{t_j}{s} \mathbb{P}[x_i | \varphi_j, \mathcal{H}] + \frac{s - t_j}{s} \mathbb{P}[x_i | \overline{\varphi}_j, \mathcal{H}] \right]$ and $\psi(\cdot)$ denotes the Digamma function. From now on we denote by $\mathsf{bu}(a_i, b_i, r)$ the values $(\hat{a}^*, \hat{b}^*)$ that satisfy eq. (3.20).

We finally introduce Algorithm 3, that exploits Propositions 1 to 3 to perform s belief updates, in response to arbitrary evidence $\mathcal{E}$.

Algorithm 3: Belief Update

Data: Model $\mathcal{H}$, evidence $\mathcal{E} = \{\varphi_1^{t_1}, \ldots, \varphi_k^{t_k}\}$

for $j \in \{1, \ldots, k\}$ **do**
  $\tau_j \leftarrow t_j / s$;
for $\ell \in \{1, \ldots, s\}$ **do**
  for $i \in \{1, \ldots, n\}$ **do**
    $r_i \leftarrow \sum_{j=1}^{k} \frac{1}{k} (\tau_j \mathbb{P}[x_i | \varphi_j, \mathcal{H}] + \overline{\tau_j} \mathbb{P}[x_i | \overline{\varphi_j}, \mathcal{H}])$
    $(\hat{a}_i^*, \hat{b}_i^*) \leftarrow \mathsf{bu}(a_i, b_i, r_i)$
  for $i \in \{1, \ldots, n\}$ **do**
    $a_i \leftarrow \hat{a}_i^*$
    $b_i \leftarrow \hat{b}_i^*$

Interestingly, Algorithm 3 allows us to update a B-PDB in an *incremental* fashion: if the evidence is provided as a stream of query-answers, dynamically changing over time both in terms of queries and observed relative frequencies, a B-PDB can incorporate such information by performing a new belief update every time a new chunk of evidence becomes available. The idea of performing repeated Bayesian updates is discussed in detail in the next Section.

3.5 Parameter Learning (MLE)

In this section we show how to exploit our belief update procedures to identify a local maximum of the likelihood function $\mathbb{P}[\mathcal{E}|\mathcal{H}^{s,k}]$. Our approach relies on the observation that belief updates can only increase the likelihood $\mathbb{P}[\mathcal{E}|\mathcal{H}^{s,k}]$; it is immediate to derive a soft-EM [26, 41] algorithm that performs repeated belief updates until convergence. First we show how a single belief update $\mathcal{H} \to \hat{\mathcal{H}}^*$ affects $\mathbb{P}[\mathcal{E}|\mathcal{H}^{s,k}]$:

$$
\begin{aligned}
\mathsf{KL}^{\mathsf{div}} &= \int .. \int p[\theta_{(\cdot,\cdot,\cdot)}|\mathcal{E},\mathcal{H}^{s,k}] \log \left[\frac{p[\theta_{(\cdot,\cdot,\cdot)}|\mathcal{E},\mathcal{H}^{s,k}]}{p[\theta_{(\cdot,\cdot,\cdot)}|\hat{\mathcal{H}}^{s,k}]} \right] d\theta_{(\cdot,\cdot,\cdot)} \\
&= h\left[p[\theta_{(\cdot,\cdot,\cdot)}|\mathcal{E},\mathcal{H}^{s,k}]\right] - \int .. \int p[\theta_{(\cdot,\cdot,\cdot)}|\mathcal{E},\mathcal{H}^{s,k}] \log p[\theta_{(\cdot,\cdot,\cdot)}|\hat{\mathcal{H}}^{s,k}] d\theta_{(\cdot,\cdot,\cdot)} \\
&= h\left[p[\theta_{(\cdot,\cdot,\cdot)}|\mathcal{E},\mathcal{H}^{s,k}]\right] - \int .. \int p[\theta_{(\cdot,\cdot,\cdot)}|\mathcal{E},\mathcal{H}^{s,k}] \log \tfrac{p[\mathcal{E},\theta_{(\cdot,\cdot,\cdot)}|\hat{\mathcal{H}}^{s,k}]}{p[\mathcal{E}|\theta_{(\cdot,\cdot,\cdot)}]} d\theta_{(\cdot,\cdot,\cdot)} \\
&= h\left[p[\theta_{(\cdot,\cdot,\cdot)}|\mathcal{E},\mathcal{H}^{s,k}]\right] + \langle p[\mathcal{E}|\theta_{(\cdot,\cdot,\cdot)}] \rangle_{p[\theta_{(\cdot,\cdot,\cdot)}|\mathcal{E},\mathcal{H}^{s,k}]} - \langle \log p[\mathcal{E},\theta_{(\cdot,\cdot,\cdot)}|\hat{\mathcal{H}}^{s,k}] \rangle_{\log p[\theta_{(\cdot,\cdot,\cdot)}|\mathcal{E},\mathcal{H}^{s,k}]}
\end{aligned}
$$

Notice that the pair $(\mathbf{a}^*, \mathbf{b}^*)$ from definition 8 is the value of $\hat{\mathcal{H}}^{s,k}$ that maximizes the quantity

$$\langle \log p[\mathcal{E},\theta_{(\cdot,\cdot,\cdot)}|\hat{\mathcal{H}}^{s,k}] \rangle_{\log p[\boldsymbol{\theta}|\mathcal{E},\mathcal{H}^{s,k}]}$$

which is a lower bound of the log-likelihood $\log \mathbb{P}[\mathcal{E}|\hat{\mathcal{H}}^{s,k}]$. Therefore, it is possible to apply the considerations from [65] to justify several variants of the EM algorithm. In the following we provide the pseudo-code of the classic, fully Bayesian, soft-EM (here named Algorithm 4). Intuitively, the "E-step" consists of the computation of the posterior $p[\theta_{(\cdot,\cdot,\cdot)}|\mathcal{E},\mathcal{H}^{s,k}]$, while the "M-step" consists of the belief update $\mathcal{H} \to \hat{\mathcal{H}}^*$.

Example 5 (continued)**.** *Let's assume we are given with a B-PDB with two tuples, x_1 and x_2, and $k = 2$ queries: $\varphi_1 = x_1 \wedge x_2$ and $\varphi_2 = x_1 \vee x_2$. The initial state of the database, $\mathcal{H}_0$, is*

$$a_1 = 1 \qquad a_2 = 1 \qquad b_1 = 3 \qquad b_2 = 3$$

Query φ_1 is observed to be satisfied $t_1 = 32$ times over $s = 100$ samples, while φ_2

Algorithm 4: Greedy-MLE

Data: Model $\mathcal{H}$, evidence $\mathcal{E} = \{\varphi_1^{t_1}, \ldots, \varphi_k^{t_k}\}$

1 **for** $j \in \{1, \ldots, k\}$ **do**
2 $\quad \tau_j \leftarrow t_j / s$;
3 **repeat**
4 $\quad$ **for** $i \in \{1, \ldots, n\}$ **do**
5 $\quad\quad r_i \leftarrow \sum_{j=1}^{k} \frac{1}{k}(\tau_j \mathbb{P}[x_i|\varphi_j, \mathcal{H}] + \overline{\tau_j}\mathbb{P}[x_i|\overline{\varphi_j}, \mathcal{H}])$
6 $\quad\quad (\hat{a}_i^*, \hat{b}_i^*) \leftarrow \mathsf{bu}(a_i, b_i, r_i)$
7 $\quad$ **for** $i \in \{1, \ldots, n\}$ **do**
8 $\quad\quad a_i \leftarrow \hat{a}_i^*$
9 $\quad\quad b_i \leftarrow \hat{b}_i^*$
10 **until convergence**;

is observed to be true $t_2 = 88$ *times. Hence* $\mathcal{E} = \{\varphi_1^{t_1}, \varphi_2^{t_2}\}$*, and the target marginal probabilities for* φ_1 *and* φ_2 *are, respectively,* $\tau_1 = 0.32$ *and* $\tau_2 = 0.88$*. Given an arbitrary B-PDB* $\mathcal{H}$*, the likelihood of observing* $\mathcal{E}$ *being generated by* $\mathcal{H}$ *is*

$$\mathbb{P}[\mathcal{E}|\mathcal{H}^{100,2}] = \binom{100}{32}\mathbb{P}[\varphi_1|\mathcal{H}]^{32}\mathbb{P}[\overline{\varphi}_1|\mathcal{H}]^{68} \cdot \binom{100}{88}\mathbb{P}[\varphi_2|\mathcal{H}]^{88}\mathbb{P}[\overline{\varphi}_2|\mathcal{H}]^{12}$$

The likelihood is maximized when $\mathbb{P}[\varphi_1|\mathcal{H}] = \tau_1$ *and* $\mathbb{P}[\varphi_2|\mathcal{H}] = \tau_2$*. There are two values of* $\boldsymbol{\theta}$ *that satisfy these conditions: either* $\boldsymbol{\theta} = (0.8, 0.4)$*, or* $\boldsymbol{\theta} = (0.4, 0.8)$*. Figure 3.3 shows how algorithm 4 converges to one of the optimal values for* $\boldsymbol{\theta}$*: the green dots represent the state of the database (in terms of* $\langle\theta_1\rangle_{\mathcal{H}}$ *and* $\langle\theta_2\rangle_{\mathcal{H}}$*) after each iteration of the cycle at lines 3-10. The starting point is* $\langle\boldsymbol{\theta}\rangle_{\mathcal{H}} = (0.25, 0.25)$*.*

3.6 Computing conditional probabilities

Computing conditional probabilities in the form $\mathbb{P}[x_i|\varphi_j, \mathcal{H}]$ is a central requirement for both Algorithm 3 and Algorithm 4. As discussed in §3.3, $\mathbb{P}[x_i|\varphi_j, \mathcal{H}]$ denotes the probability of observing tuple x_i being present in a possible world sampled from $\mathcal{H}$ that satisfies φ_j. In this Section we study the computational complexity of deriving such probability. The following Theorem states that the dichotomy identified by [25]

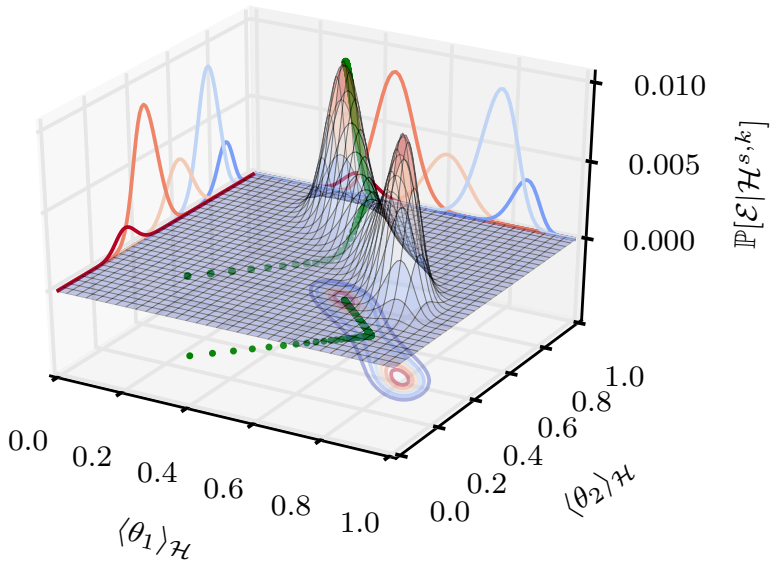


Figure 3.3: Convergence of Algorithm 4 towards a maximum likelihood estimate of (θ_1, θ_2).

(see Lemma 3) also applies to the computation of conditional probabilities in the form $\mathbb{P}[x_i|\varphi_j, \mathcal{H}]$.

Theorem III.2. *Let φ_j represents the lineage of a Boolean conjunctive query, and x_i be one of its literals. When φ_j is read-once, computing the conditional probabilities $\mathbb{P}[x_i|\varphi_j, \mathcal{H}]$ (or $\mathbb{P}[x_i|\overline{\varphi_j}, \mathcal{H}]$) takes polynomial time in the worst case. When φ_j is not read-once, the same problem becomes $\#\mathcal{P}$-complete.*

Proof. The first assertion is proven by observing that

$$\mathbb{P}[x_i|\varphi_j, \mathcal{H}] = \mathbb{P}[\varphi_j|x_i, \mathcal{H}] \cdot \mathbb{P}[x_i|\mathcal{H}]/\mathbb{P}[\varphi_j|\mathcal{H}] \tag{3.21}$$

Notice that $\mathbb{P}[\varphi_j|x_i, \mathcal{H}]$ represents the marginal probability of the formula obtained by replacing x_i with $\top$ in φ_j. We denote by $(\varphi_j|_{x_i})$ such formula, therefore $\mathbb{P}[(\varphi_j|_{x_i})|\mathcal{H}] = \mathbb{P}[\varphi_j|x_i, \mathcal{H}]$. If φ_j is read-once then so is $(\varphi_j|_{x_i})$, therefore computing the RHS of eq. (3.21) takes polynomial time. We prove the second assertion by reduction: let φ_j

be a non-read-once expression, and $\{x_1, \ldots, x_n\}$ the literals appearing in it; we want to reduce the problem of computing $\mathbb{P}[\varphi_j|\mathcal{H}]$ to the problem of computing conditional probabilities in the form $\mathbb{P}[x|\varphi, \mathcal{H}]$. From 3.21 it is immediate to obtain

$$\mathbb{P}[\varphi_j|\mathcal{H}] = \mathbb{P}[x_i|\mathcal{H}] \cdot \mathbb{P}[(\varphi_j|_{x_i})|\mathcal{H}]/\mathbb{P}[x_i|\varphi_j, \mathcal{H}] \tag{3.22}$$

Since Equation 3.22 holds for any literal in $\{x_1, \ldots, x_n\}$, we can apply it $n-1$ times and obtain:

$$\mathbb{P}[\varphi_j|\mathcal{H}] = \frac{\mathbb{P}[x_1|\mathcal{H}]}{\mathbb{P}[x_1|\varphi_j,\mathcal{H}]} \cdot \frac{\mathbb{P}[x_2|\mathcal{H}]}{\mathbb{P}[x_2|(\varphi_j|_{x_1}),\mathcal{H}]} \cdot \ldots \cdot \frac{\mathbb{P}[x_{n-1}|\mathcal{H}]}{\mathbb{P}[x_{n-1}|(\varphi_j|_{x_1..x_{n-2}}),\mathcal{H}]} \cdot \mathbb{P}[(\varphi_j|_{x_1..x_{n-1}})|\mathcal{H}]$$

Notice that the last factor consists of the probability of a read-once boolean formula, as the expression $(\varphi_j|_{x_1..x_{n-1}})$ depends only on the literal x_n. Therefore, if we have an oracle able to compute conditional probabilities in the form $\mathbb{P}[x|\varphi, \mathcal{H}]$, we can compute $\mathbb{P}[\varphi_j|\mathcal{H}]$ in polynomial time, by making $(n-1)$ calls. □

From Lemma 3 and Theorem III.2 it follows immediately that computing a single Bayesian update, to incorporate the answer to a hierarchical query, takes polynomial time in data-size. In the next Section we adapt the well-known algorithm by Dalvi and Suciu [24] to the goal of computing Bayesian updates extensionally, by means of "CP-plans".

3.6.1 CP-plans: Extensional Evaluation of $\mathbb{P}[x_i|\varphi_j, \mathcal{H}]$ for Safe Queries

Let $\mathbf{q} = [\varphi_1, .., \varphi_k]$ be a non-Boolean hierarchical query. Our goal is to compute $\mathbb{P}[x_i|\varphi_j, \mathcal{H}]$ for every Boolean query φ_j in $\{\varphi_1, .., \varphi_k\}$ and every literal x_i appearing in φ_j. We first show how to compute $\mathbb{P}[\varphi_j|\mathcal{H}]$ and $\mathbb{P}[\varphi_j|x_i, \mathcal{H}]$ for every φ_j and x_i, extensionally. Once $\mathbb{P}[\varphi_j|\mathcal{H}]$ and $\mathbb{P}[\varphi_j|x_i, \mathcal{H}]$ are known, it is immediate to obtain $\mathbb{P}[x_i|\varphi_j, \mathcal{H}]$ by eq. (3.21). A plan performing such computation is called "*CP-plan*".

In order to represent CP-plans compactly, we introduce a simple extension of pRA. In our algebra, a CP-plan (P^{cp}) is a sentence respecting the following grammar:

$$P^{\mathsf{cp}} ::= CP(R_0^{\mathsf{p}}) \mid \pi^{\mathsf{c}}_X(P') \mid \sigma^{\mathsf{c}}(P') \mid \bowtie^{\mathsf{c}} [P', P'', \ldots]$$

R_0^{p} represents an arbitrary TI-relation, where each tuple has a unique identifier $\mathtt{tid}$ and is associated with a marginal probability $\mathtt{p}$. Let's assume $\mathbf{A}$ is a key for R_0^{p}, consisting of all the attributes except for $\mathtt{tid}$ and $\mathtt{p}$. The operator $CP(R^{\mathsf{p}})$ turns a TI-relation R_0^{p} into a pair $(R^{\mathsf{p}}, R^{\mathsf{cp}})$, where

$$R^{\mathsf{p}}(\mathbf{A}, \mathtt{p}) \stackrel{\text{def}}{=} \pi_{\mathbf{A},\mathtt{p}}(R_0^{\mathsf{p}})$$
$$R^{\mathsf{cp}}(\mathbf{A}, \mathtt{cp}, \mathtt{lt}) \stackrel{\text{def}}{=} \pi_{R^{\mathsf{p}}.\mathbf{A},1,R^{\mathsf{p}}.\mathtt{tid}}(R^{\mathsf{p}})$$

Intuitively, R^{p} is obtained from R_0^{p} by projecting-away $\mathtt{tid}$. R^{cp} associates each tuple x of R^{p} with the conditional probability $\mathbb{P}[x|x]$, which is, by definition, equal to 1. All the other operators of our algebra process pairs of relations in the form $(R^{\mathsf{p}}, R^{\mathsf{cp}})$. Let $\mathbf{B}$ be a strict subset of $\mathbf{A}$. If we apply the projection operator $\pi^{\mathsf{c}}_{\mathbf{B}}$ to the pair $(R^{\mathsf{p}}, R^{\mathsf{cp}})$, we obtain a pair of relations $(Q^{\mathsf{p}}, Q^{\mathsf{cp}})$, defined as follows:

$$Q^{\mathsf{p}}(\mathbf{B}, \mathtt{p}) \stackrel{\text{def}}{=} \pi_{\mathbf{B},(1-\Pi_{\mathsf{agg}}(\overline{R^{\mathsf{p}}.\mathtt{p}}))}(R^{\mathsf{p}})$$
$$Q^{\mathsf{cp}}(\mathbf{B}, \mathtt{cp}, \mathtt{lt}) \stackrel{\text{def}}{=} \pi_{\mathbf{A},\mathsf{cpexp},R^{\mathsf{cp}}.\mathtt{lt}}[(R^{\mathsf{p}} \bowtie_{\mathbf{A}} R^{\mathsf{cp}}) \bowtie_{\mathbf{B}} Q^{\mathsf{p}}]$$

Where $\Pi_{\mathsf{agg}}(\cdot)$ denotes the aggregate product and $\mathsf{cpexp} \stackrel{\text{def}}{=} 1 - (\overline{Q^{\mathsf{p}}.\mathtt{p}} \cdot \overline{R^{\mathsf{cp}}.\mathtt{cp}}/\overline{R^{\mathsf{p}}.\mathtt{p}})$. The selection operator (σ^{c}) simply applies the selection predicate to both R^{p} and R^{cp}. Therefore, the statement $(Q^{\mathsf{p}}, Q^{\mathsf{cp}}) = \sigma^{\mathsf{c}}(R^{\mathsf{p}}, R^{\mathsf{cp}})$ is equivalent to the following RA

plan:

$$Q^{\mathsf{p}}(\mathbf{A}, \mathtt{p}) \stackrel{\text{def}}{=} \sigma(R^{\mathsf{p}})$$

$$Q^{\mathsf{cp}}(\mathbf{A}, \mathtt{cp}, \mathtt{lt}) \stackrel{\text{def}}{=} \sigma(R^{\mathsf{cp}})$$

Let's now assume we are given a collection of m relation pairs $\{(R_1^{\mathsf{p}}, R_1^{\mathsf{cp}}), .., (R_m^{\mathsf{p}}, R_m^{\mathsf{cp}})\}$ and $\mathbf{A}_i = \mathtt{hvar}(R_i^{\mathsf{p}}) \setminus \{\mathtt{p}\}$. Let's define $\mathbf{A} = \cup_{i=1}^{m} \mathbf{A}_i$. The statement $(Q^{\mathsf{p}}, Q^{\mathsf{cp}}) = \bowtie^{\mathsf{c}} [(R_1^{\mathsf{p}}, R_1^{\mathsf{cp}}), .., (R_m^{\mathsf{p}}, R_m^{\mathsf{cp}})]$ is equivalent to the following RA plan:

$$Q^{\mathsf{p}}(\mathbf{A}, \mathtt{p}) \stackrel{\text{def}}{=} \pi_{\mathbf{A}, (\Pi_{i=1}^{m} R_i^{\mathsf{p}}.\mathtt{p})} [\bowtie [R_1^{\mathsf{p}}, .., R_m^{\mathsf{p}}]]$$

$$V_i(\mathbf{A}, \mathtt{cp}, \mathtt{lt}) \stackrel{\text{def}}{=} \pi_{\mathbf{A}, \mathtt{cpexp}, R_i^{\mathsf{cp}}.\mathtt{lt}} [\bowtie_{\mathbf{A}_i} [Q^{\mathsf{p}}, R_i^{\mathsf{p}}, R_i^{\mathsf{cp}}]] \; \forall i \in \{1..m\}$$

$$Q^{\mathsf{cp}}(\mathbf{A}, \mathtt{cp}, \mathtt{lt}) \stackrel{\text{def}}{=} \uplus_{i=1}^{m} V_i$$

Here $\mathtt{cpexp} \stackrel{\text{def}}{=} (Q^{\mathsf{p}}.\mathtt{p} \cdot R_i^{\mathsf{cp}}.\mathtt{cp} / R_i^{\mathsf{p}}.\mathtt{p})$. Now that we have defined all the operators of our algebra, we can show how to build a CP-plan for a given hierarchical query. The method we propose (Algorithm 5) is a straightforward adaptation of procedure developed by Dalvi and Suciu [24] for constructing safe plans. Their algorithm is known to be sound and complete; Algorithm 5 inherits both properties.

Algorithm 5: SafeCpPlan

Data: Hierarchical query $\mathbf{q}(..) :- R(..), S(..), \ldots$

1 **if** $\mathit{evar}(\mathbf{q}) = \emptyset$ **then**
2 | **return** $CP(R) \bowtie^{\mathsf{c}} CP(S) \bowtie^{\mathsf{c}} \ldots$
3 **else if** $\mathbf{q} :- \mathbf{q}', \mathbf{q}''$ *and* $\mathit{evar}(\mathbf{q}') \cap \mathit{evar}(\mathbf{q}'') = \emptyset$ **then**
4 | **return** SafeCpPlan($\mathbf{q}'$) $\bowtie^{\mathsf{c}}$ SafeCpPlan($\mathbf{q}''$)
5 **else if** $X \in \mathit{evar}(\mathbf{q})$ *is a root variable* **then**
6 | **return** π^{c}_{-X}(SafeCpPlan($\mathbf{q}'(X, \mathtt{hvar}(\mathbf{q})) :- R(..), S(..), \ldots)$)

Let $(Q^{\mathsf{p}}, Q^{\mathsf{cp}})$ be the result of a CP-plan generated by Algorithm 5: if φ_j is the lineage of a tuple in Q^{p} and its literals are $\{x_1, .., x_m\}$, then Q^{cp} is guaranteed to contain m copies of such tuple, each copy being associated with a conditional probability

$\mathbb{P}[\varphi_j|x_i]$, for every x_i in $\{x_1, .., x_m\}$.

Example 4 (continued)**.** *If* $\mathbf{q}$ *denotes the hierarchical query from eq.* (3.4)*, then SafeCpPlan(*$\mathbf{q}$*) returns the following plan:*

$$(Q^{\mathsf{p}}, Q^{\mathsf{pc}}) = \pi^{\mathsf{c}}_{-X}(\pi^{\mathsf{c}}_{-Y}(CP(R)) \bowtie^{\mathsf{c}} CP(S))$$

Notice that Q^{p} *is equivalent to the TI-relation returned by plan* P' *in eq.* (3.5)*.*

3.7 Handling Qualitative Feedback

In this section we show how B-PDBs can leverage user-provided qualitative feedback to speed-up the learning process. In the context of probabilistic databases, we define *qualitative feedback* as a partial order ($\succeq$) over query-answers. By definition, $\succeq$ denotes a binary relation over Boolean queries that is reflexive, antisymmetric and transitive. Therefore, if q, q' and q'' denote arbitrary Boolean queries, the following holds:

1. $q \succeq q$ (*reflexivity*)
2. if $q \succeq q'$ and $q' \succeq q$, then q and q' are logically equivalent (*antisymmetry*)
3. if $q \succeq q'$ and $q' \succeq q''$, then if $q \succeq q''$ (*transitivity*)

The assertion "$q \succeq q'$" ("q *dominates* q'") represents the user belief that every possible world w that satisfies q' should satisfy q as well. Therefore, it represents the user belief that the Boolean query $\overline{q'} \vee q$ should return $\top$ with probability 1. A dominance relationship $\succeq$ can be seen as a collection of statements in the form "$q \succeq q'$". A relation $\succeq$ is *satisfiable* when there is at least one possible world that satisfies it. Under reasonable syntactic restrictions, it is possible to ensure that $\succeq$ is satisfiable.

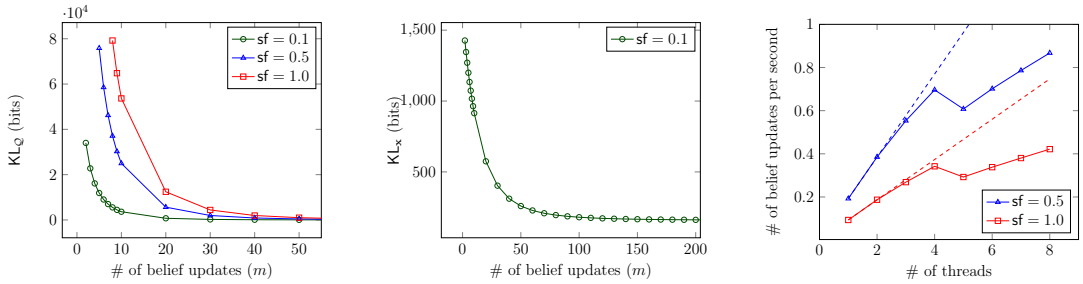
Proposition 5. *If* $\mathcal{Q} = \{q_1, \ldots, q_k\}$ *is a collection of Boolean queries whose lineage formulas are pair-wise independent, then any arbitrary dominance relationship* $\succeq$ *over*

$\mathcal{Q}$ is satisfiable, as long as $\succeq$ is a reflexive, antisymmetric and transitive (i.e. a partial order).

It is straightforward to extend Algorithms 3 and 4 as to handle qualitative user-feedback that respects the constraints of Proposition 5. The general idea is to augment the evidence with the additional information implied by $\succeq$. Let q be an arbitrary query in $\mathcal{Q}$; let $\mathcal{Q}'$ be the subset of queries in $\mathcal{Q}$ that are dominated by q, and $\mathcal{Q}''$ be the subset of queries in $\mathcal{Q}$ that dominate q. If we observe a positive answer to q, then the Bayesian update of $\mathcal{H}$ should be computed w.r.t. to the augmented query $q \wedge (\wedge_{q_i \in \mathcal{Q}''} q_i)$. If we observe a negative answer to q, then the Bayesian update should be computed w.r.t. to the augmented query $\overline{q} \wedge (\wedge_{q_i \in \mathcal{Q}'} \overline{q_i})$.

3.8 Experiments

The goal of this section is to demonstrate that a B-PDB can effectively learn its parameters. In our experiments we use a TI-PDB with known parameters as ground truth and a fixed set of queries to generate a stream of query-answers. We then incrementally update a B-PDB accordingly, and observe the trajectory taken by its internal state. From now on we denote with $\mathcal{T}$ the parameters of the ground-truth TI-PDB, with $\mathcal{Q}$ a fixed set of conjunctive hierarchical queries, with $\mathcal{H}_0$ the initial state of the B-PDB, and with $\mathcal{H}_m$ its state after m belief updates. Our goal is to provide experimental validation of the following claims: (1) if m is large enough, the distribution $\mathbb{P}[q|\mathcal{H}_m]$ will converge towards $\mathbb{P}[q|\mathcal{T}]$ for every $q \in \mathcal{Q}$, independently from the initial state $\mathcal{H}_0$; (2) if $\mathcal{H}_0$ mirrors the probabilities of $\mathcal{T}$, except for a limited amount of noise, with the right evidence expectation maximization may remove such noise, so that $\langle \boldsymbol{\theta} \rangle_{\mathcal{H}_m}$ converges towards $\mathcal{T}$ when m is large enough; (3) the algorithms presented in this chapter are scalable as they can be parallelized effectively.



(a) Experiment 1 (b) Experiment 2 (c) Experiment 3

Figure 3.4

We verify claim (1) and (2) by measuring the following metrics:

$$\mathsf{KL}_{\mathcal{Q}} \stackrel{\text{def}}{=} \sum_{\varphi \in \mathcal{Q}} \mathbb{P}[\varphi|\mathcal{T}] \log \left[\frac{\mathbb{P}[\varphi|\mathcal{T}]}{\mathbb{P}[\varphi|\mathcal{H}_m]} \right] + \mathbb{P}[\overline{\varphi}|\mathcal{T}] \log \left[\frac{\mathbb{P}[\overline{\varphi}|\mathcal{T}]}{\mathbb{P}[\overline{\varphi}|\mathcal{H}_m]} \right]$$

$$\mathsf{KL}_{\mathbf{x}} \stackrel{\text{def}}{=} \sum_{i=1}^{n} \mathbb{P}[x_i|\mathcal{T}] \log \left[\frac{\mathbb{P}[x_i|\mathcal{T}]}{\mathbb{P}[x_i|\mathcal{H}_m]} \right] + \mathbb{P}[\overline{x_i}|\mathcal{T}] \log \left[\frac{\mathbb{P}[\overline{x_i}|\mathcal{T}]}{\mathbb{P}[\overline{x_i}|\mathcal{H}_m]} \right]$$

Intuitively, $\mathsf{KL}_{\mathcal{Q}}$ represents the expected number of bits required to encode the difference in query-answers over $\mathcal{Q}$ between $\mathcal{T}$ and $\mathcal{H}$. $\mathsf{KL}_{\mathbf{x}}$ is the KL divergence between $\mathcal{T}$ and $\mathcal{H}$. Notice that when the former equals zero, $\mathcal{H}_m$ and $\mathcal{T}$ are guaranteed to exhibit the same marginal probabilities for every query in $\mathcal{Q}$. When the latter equals zero, $\mathcal{H}_m$ and $\mathcal{T}$ will exhibit the same marginal probabilities *for any arbitrary query*.[9] We verify claim (3) by observing the actual response times of our algorithms as we increasingly parallelize the execution.

Data and Queries. We use the TPC-H `dbgen` [1] utility to generate a dataset of 1 GB. We obtain $\mathcal{T}$ by annotating each tuple with a random probability chosen uniformly between 0.1 and 0.6. The query-set $\mathcal{Q}$ consists of queries Q_3, Q_4 and Q_6 from the TPC-H benchmark. The choice of queries mirrors queries used to evaluate prior work on PDBs [5, 24].

[9] In other words: $\langle \boldsymbol{\theta} \rangle_{\mathcal{H}_m} = \mathcal{T}$.

Implementation. Our prototype implementation of B-PDBs consists of two components, a *query planner* and an *execution engine*. The query planner implements Algorithm 5: it is responsible for generating a CP-plan and translating it into a set of materialized SQL views, one pair of views for each step in the plan. Rather than computing marginal and conditional probabilities (columns `p` and `cp`), the planner associates a unique identifier to each intermediate tuple in the plan. The views are then used to compile an execution plan, a compact representation of the CP-plan, consisting only of the identifiers and their dependencies, without the relational data. This is processed by the execution engine, that is responsible for the actual computation of the probabilities (both marginal and conditional) associated with each identifier, and for computing the Bayesian updates, as described in Algorithm 3 and 4. This decoupled architecture ensures that the SQL optimizer is called only once per query, even if the same query is used over many iterations of Algorithm 4.

Experiment 1. We initialize $\mathcal{H}_0$ setting all its parameters a_i and b_i to 1, so that all tuples in the database have the same uninformative probability 0.5. We then run Algorithm 4 for 500 iterations;[10] we repeat the test over several variants of the TPC-H dataset, ranging the scale factor (sf) between 0.1 and 1.0.[11] Figure 3.4a depicts the observed $\mathsf{KL}_{\mathcal{Q}}$ against the number of belief updates (m). Independently from the scaling factor, $\mathsf{KL}_{\mathcal{Q}}$ converges towards 0 in less than 300 iterations. One hundred iterations are sufficient to drop $\mathsf{KL}_{\mathcal{Q}}$ below 100 bits.

Experiment 2. We first initialize $\mathcal{H}_0$ as a low-entropy copy of $\mathcal{T}$, ensuring that $\langle \boldsymbol{\theta} \rangle_{\mathcal{H}} = \mathcal{T}$ and $a_i + b_i = 10^6$, for every tuple x_i in $\mathcal{H}$. We then add noise to 1% of the tuples of $\mathcal{H}_0$, chosen at random, setting their parameters to a high-entropy state ($a := b := 1$). Since the lineage of the TPC-H queries Q_3, Q_4 and Q_6 does not cover all the tuples in the dataset, we augment $\mathcal{Q}$ with 12 "group-by" queries, devoid of any selection predicate. We then run Algorithm 4 for 500 iterations, measuring the value

[10] Each iteration process the same, fixed set of query-answers from $\mathcal{T}$.

[11] A scale factor of 1.0 corresponds to about 1 GB of data.

of $\mathsf{KL}_{\mathbf{x}}$ after each belief update. Figure 3.4b reports our findings: after 230 updates $\mathsf{KL}_{\mathbf{x}}$ drops below 160 bits, and remains mostly constant afterwards. As expected, the ability of EM to reconstruct the whole set of parameters of $\mathcal{T}$ is not guaranteed, and depends strongly on the evidence available. Nonetheless, given the right evidence, EM can filter most of the noise introduced in $\mathcal{H}_0$. The experiment also highlights the importance of the entropy h in B-PDBs: it can be used as a marker for parameters that need updating and as a mean for providing implicit evidence.

Experiment 3. We repeat the computation from Experiment 1 on a multi-core machine, varying the number of threads between 1 and 8. Figure 3.4c shows the resulting average processing speed, measured in number of belief updates per second. The plot reports the average processing time of the Execution Engine, without including the time spent on the initial planning phase. The dashed lines indicate the ideal speedup. Up to 4 threads we observed the speedup to be almost perfect, and to slightly deteriorate afterwards. We believe this is due in part to the fact that TPC-H query Q_4 consists of exactly four Boolean queries, all having very large lineage.

3.9 Related Work

Stoyanovich et al. [76] derive probability distributions for the missing parts of incomplete databases, using the complete parts as evidence. Dylla and Theobald [27] study the problem of deriving the parameters of a TI-PDB from a set of Boolean queries, labeled with their marginal probabilities. They prove the problem is $\#\mathcal{P}$-hard in the general case, and provide a sound criterion to identify problem instances that admit a solution. Rather than computing a maximum-likelihood estimate of the parameters, like we advocate in this chapter, they propose to derive them by direct minimization of the mean squared error. Their approach does not consider Bayesian updates. Parameter learning has been proposed in the context of Probabilistic Logic Programming, either by minimizing the mean squared error [39] or by

maximum likelihood estimation [40]. It is also a central feature for many knowledge-based model construction (KBMC) frameworks, including Probabilistic Relational Models [57], Markov Logic [70], Multi-entity Bayesian Networks [60] and many others. All the above approaches rely on probabilistic models that are significantly more sophisticated than TI-PDBs, but without the complexity guarantees provided by the dichotomy theorem [25]. Koch and Olteanu [56] were the first to address the problem of conditioning in probabilistic databases. Their work relies on U-databases, while ours focuses on TI-PDBs and hierarchical queries.

CHAPTER IV

Conclusions and Future Work

4.1 Conclusions

In this dissertation we investigated the use of qualitative feedback in combination with deterministic and probabilistic databases. In the context of deterministic databases, we advanced the state-of-the-art by introducing a novel evaluation algorithm for prioritized skylines. Our work shows that the complexity of answering prioritized skyline queries is $\mathcal{O}\left(n \log^{d-2} v\right)$ in the worst-case scenario, and $\mathcal{O}\left(n\right)$ in the average-case, where n is the number of tuples, v is the output-size and d is the number of dimensions. Since its worst-case complexity depends explicitly on v, our algorithm is *output-sensitive*. This is a very desirable property in the context of preference queries, since the output-size is expected to be very small. Beyond its asymptotic complexity, our algorithm proved to be fast and effective w.r.t. to both synthetic and real-world benchmarks.

In the context of probabilistic databases we introduced B-PDBs, a novel framework that generalizes the popular tuple-independent model, and supports belief updating and parameter learning in a principled, efficient way. To the best of our knowledge, we are the first to design a probabilistic database that can be trained through the observation of query-answers, in an incremental fashion. Our approach relieves the user from the need of specifying the model's parameters explicitly, defin-

ing a precise probability for each and every tuple. Additionally, we have shown how B-PDBs' belief updates can leverage user-provided qualitative feedback to speed-up the convergence towards a maximum likelihood estimate of the parameters.

4.2 Future Work

Our research suggests several directions for future work. In the context of prioritized skyline, it would be interesting to develop a *distribution-sweep* algorithm for p-screening, following the approach proposed by [43, 72] in the context of skyline queries. The resulting algorithm would generalize the results presented in Chapter II to the external memory model. A second direction of future investigation could be the study of prioritized skyline queries in the context of noisy/uncertain data, in the spirit of [47].

In the context of probabilistic databases, we believe that B-PDBs could be extended in several ways. First, it would be interesting to study the behavior of Bayesian updates in combination with approximate inference algorithms, such as dissociations [33], anytime-approximations [30] and Monte-Carlo simulations [49]. Secondly, it would be interesting to drop the tuple independence assumption, adopting a more general probabilistic model where tuples' correlations are allowed, as in *pc-tables* [38, 77]. The resulting framework would replace the Beta priors with Dirichlet distributions. Thirdly, we would like to generalize B-PDBs as to allow the adoption of Beta-Binomial distributions. Fourthly, we would like to experiment the execution of CP-plans over distributed Map/Reduce clusters.

APPENDICES

APPENDIX A

Nomenclature

A.1 Chapter II

Symbol	Meaning
$\mathcal{A}$	a relation schema
D	a relation instance
$\mathbf{A}_1, \mathbf{A}_2, \ldots$	attributes
$t_1, t_2, \ldots$	tuples
π	a p-expression
$\succ_\pi$	the strict partial order induced by π
$M_{\text{sky}}(D)$	the skyline of D
$M_\pi(D)$	the p-skyline of D, w.r.t. π
n	the size of the input (# of tuples)
v	the size of the output (# of tuples)
d	the number of relevant attributes
$W \not\succeq_\pi B$	no tuple in W is better than (or indistinguishable from) any tuple in B
Γ_π, Γ_π^r	the p-graph of π and its trans. reduction
$\mathcal{V}ar(\pi)$	the attributes appearing in π
$\mathcal{S}\text{ucc}_\pi(\mathbf{A}_i)$	immediate successors of $\mathbf{A}_i$ in Γ_π^r
$\mathcal{D}\text{esc}_\pi(\mathbf{A}_i)$	descendants of $\mathbf{A}_i$ in Γ_π^r
$\mathcal{P}\text{re}_\pi(\mathbf{A}_i)$	immediate predecessors of $\mathbf{A}_i$ in Γ_π^r
$\mathcal{A}\text{nc}_\pi(\mathbf{A}_i)$	ancestors of $\mathbf{A}_i$ in Γ_π^r
$\mathcal{R}\text{oots}_\pi$	attributes having no ancestors in Γ_π^r
$\mathcal{B}\text{etter}_\pi(t', t)$	attributes where t' is preferred to t
$\mathcal{T}op_\pi(t', t)$	topmost attributes in Γ_π^r where t and t' disagree
$d_{\mathbf{A_i}}$	the depth of $\mathbf{A}_i$, the length of the longest path in Γ_π^r from any root to $\mathbf{A}_i$
$C_d(v, n)$	worst-case complexity of skyline queries
$F_d(b, w)$	w.c. complexity of screening queries
$C_d^*(v, n)$	w.c. complexity of p-skyline queries
$F_d^*(b, w)$	w.c. complexity of p-screening queries

Table A.1: Nomenclature of Chapter II

A.2 Chapter III

Symbol	Meaning
$\mathcal{H}$	Beta Probabilistic DB
$\mathcal{D}$	Tuple-independent Probabilistic DB
$p[\cdot]$	Probability density function
$\mathbb{P}[\cdot]$	Probability measure
$\langle f(\theta) \rangle_{p[\theta]}$	Expected value of $f(\theta)$ when $\theta \sim p[\cdot]$
$h[\cdot]$	differential entropy
$\mathcal{E}$	Evidence
$\mathcal{B}e(a_i, b_i)$	P.d.f. of a Beta distribution
a, b	Parameters of the Beta distribution
$B(\cdot)$	Beta function
$\Gamma(\cdot)$	Gamma function
$\psi(\cdot)$	Digamma function
$\psi'(\cdot)$	Trigamma function
w	Possible world
$x_1, \ldots, x_n$	Tuples / Boolean random variables
$\theta_1, \ldots, \theta_n$	Tuples' marginal probabilities
$\boldsymbol{\theta}$	Vector $(\theta_1, \ldots, \theta_n)$
$\overline{\theta_i}$	Abbreviation for $(1 - \theta_i)$
$q_1, \ldots, q_k$	Conjunctive queries
$\varphi_1, \ldots, \varphi_k$	Lineage formulas
$\overline{\varphi_j}$	Abbreviation for $\neg\varphi_j$
t_j	Observed frequency of positive answers to φ_j
τ_j	Observed relative freq. of positive answers to φ_j
k	Number of queries
s	Number of samples per query
n	Number of tuples
$R, S, T, \ldots$	Relations' names
$X, Y, Z, \ldots$	First-order logic variables
$\mathsf{ADom}(\cdot)$	Active domain
$\texttt{hvar}(q_j)$	The head variables of query q_j
$\texttt{evar}(q_j)$	The existential variables of query q_j
$\mathcal{T}$	Ground-truth

Table A.2: Nomenclature of Chapter III

APPENDIX B

Proofs

B.1 Chapter II

B.1.1 Proof of Theorem II.1

Proof. If we run OSDC (Algorithm 2), the following upper bound holds on $C_d^*(v,n)$, for some fixed constant k_0

$$C_d^*(v,n) \leq k_0 n + C_d^*\left(v', \frac{n}{2}\right) + C_d^*\left(v'', \frac{n}{2}\right) + F_{d-1}^*\left(v', \frac{n}{2}\right)$$

where $v' + v'' + 1 = v$. The linear term $k_0 n$ models the time spent at lines 13-16, the second and the third terms the recursive calls at lines 17 and 19, while the last term is the p-screening operation at line 18. In order to keep track of the partitioning of v into smaller chunks during the recursion, we need to introduce some additional notational conventions. Let's denote by $v_{\ell,j}$ the size of the $j-th$ portion of v obtained at depth ℓ into the recursion. We can organize the different $v_{\ell,j}$ variables into a binary tree, as in Figure B.1. Clearly, only a finite subtree rooted in $v_{0,0}$ will cover the variables having

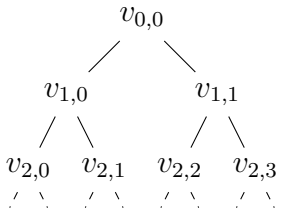


Figure B.1

a strictly positive value. The root $v_{0,0}$ is equal to v and, for all ℓ and j, $v_{\ell,j}$ is either $1 + v_{\ell+1,2j} + v_{\ell+1,2j+1}$ or zero. Notice there are exactly v assignments to (ℓ, j) such that $v_{\ell,j} > 0$, and the recursion proceeds up to a level $\ell_{\max}$ such that $v_{\ell_{max}+1,j} = 0$ for all j. Using this notation:

$$C_d^*(v,n) \leq \sum_{\ell=0}^{\ell_{\max}} \sum_{\{j: v_{\ell,j}>0\}} k_0 \frac{n}{2^\ell} + F_{d-1}^*\left(v_{\ell+1,2j}, \frac{n}{2^{\ell+1}}\right)$$

We don't know $\ell_{\max}$ a priori; depending on the input it could be anywhere between $\log v$ and $\min(v, \log n)$. On the other hand, we do know the above sum has exactly v terms, and that their cost increases when the value of ℓ decreases. Therefore, in the worst-case scenario only the first $\log v$ levels of recursion are used, i.e. $\ell_{\max} = \log v$.

$$C_d^*(v,n) \leq \sum_{\ell=0}^{\log v} \left[k_0 n + 2^\ell \cdot F_{d-1}^* \left(v_{\ell+1,2j}, \frac{n}{2^{\ell+1}} \right) \right]$$

Assuming $F_d^*(b,w)$ is $\mathcal{O}\left((b+w)\cdot(\log b)^{d-2}\right)$, there is a constant k_1 such that

$$C_d^*(v,n) \leq k_0 n \log v + k_1 n \cdot \sum_{\ell=0}^{\log v - 1} \left(\log\left(v_{\ell+1,2j}\right)\right)^{d-3}$$

Since in the worst-case scenario only the first $\log v$ levels of recursion are used, we have that $v_{\ell,j} \leq v/2^\ell \leq n/2^\ell$, hence

$$C_d^*(v,n) \leq k_0 n \log v + k_1 n \cdot \sum_{\ell=0}^{\log v - 1} \left(\log\left(v\right) - (\ell+1)\right)^{d-3}$$

If we set $h = \log\left(v\right) - (\ell+1)$, we obtain

$$C_d^*(v,n) \leq k_0 n \log v + k_1 n \cdot \sum_{h=0}^{\log v - 1} h^{d-3}$$

We can conclude that $C_d^*(v,n) \leq \mathcal{O}\left(n \cdot (\log v)^{d-2}\right)$ □

B.1.2 Complexity of P-Screening

We can now analyze the complexity of p-screening. We start from the simple case $d \leq 3$. When $d = 1$ the problem coincides with regular screening, and it is linear. When $d = 2$ or $d = 3$ the following lemmas apply

Lemma 5. *$F_2^*(b,w) \leq \mathcal{O}\left((b+w)\log b\right)$ for any $b > 1$ and $w > 0$.*

Proof. A p-expression with only two attributes can be either a lexicographical order or a regular skyline. In the first case p-screening takes $\mathcal{O}(b+w)$ time, in the second it takes $\mathcal{O}((b+w)\log b)$ time, as per Proposition 4. □

Lemma 6. *$F_3^*(b,w) \leq \mathcal{O}((b+w)\log b)$ for any $b > 1$ and $w > 0$.*

Proof. A p-expression π over three attributes can come in five possible forms:

Case 1 When $\pi = \mathbf{A}_1 \otimes \mathbf{A}_2 \otimes \mathbf{A}_3$ p-screening consists of a regular screening in three dimensions: its complexity is $\mathcal{O}((b+w)\cdot\log b)$, as per Proposition 4.

Case 2 When $\pi = \mathbf{A}_1$ & $\mathbf{A}_2$ & $\mathbf{A}_3$, p-screening can be done in $\mathcal{O}(b+w)$ time, since π represents a lexicographical order. In $\mathcal{O}(b)$ time we can find a maximal element p^* in B, in additional $\mathcal{O}(w)$ time we can check for each $t \in W$ whether $p^* \succ_\pi t$ holds.

Case 3 When $\pi = \mathbf{A}_1$ & $(\mathbf{A}_2 \otimes \mathbf{A}_3)$ we can proceed as follows: let a_1^* be the best value for attribute $\mathbf{A}_1$ amongst all tuples in B, let W_b, W_e and W_w contain all the tuples in W assigning to $\mathbf{A}_1$ a value respectively better, equal and worse than a_1^*; we have that

$$\begin{bmatrix} B \\ W \end{bmatrix}_\pi = W_b \cup \begin{bmatrix} M_{\mathbf{A}_1}(B) \\ W_e \end{bmatrix}_{\mathbf{A}_2 \otimes \mathbf{A}_3}$$

Since $M_{\mathbf{A}_1}(B)$, W_b, W_e and W_w can be computed in linear time, the overall complexity is dominated by the two-dimensional screening, that takes $\mathcal{O}((b+w)\log b)$, as per Proposition 4.

Case 4 When $\pi = (\mathbf{A}_1 \otimes \mathbf{A}_2)$ & $\mathbf{A}_3$ we can proceed as follows: first we compute the set

$$W' = \begin{bmatrix} B \\ W \end{bmatrix}_{\mathbf{A}_1 \otimes \mathbf{A}_2}$$

in $\mathcal{O}((b+w)\log b)$ time, then we sort B by the lexicographic order $\succ_{\mathbf{A}_1 \& \mathbf{A}_2 \& \mathbf{A}_3}$; given an assignment to $\mathbf{A}_1$ and $\mathbf{A}_2$ finding the best value for $\mathbf{A}_3$ in B takes only

$\mathcal{O}(\log b)$, therefore pruning all the remaining dominated points from W' takes $\mathcal{O}(w \log b)$.

Case 5 When $\pi = (\mathbf{A}_1 \,\&\, \mathbf{A}_2) \otimes \mathbf{A}_3$ we can proceed as follows: let k be the number of distinct values for $\mathbf{A}_1$ in B, we can partition both B and W into k subsets, such that $B_k \not\succeq_{\mathbf{A}_1} B_{k-1} \not\succeq_{\mathbf{A}_1} \ldots \not\succeq_{\mathbf{A}_1} B_1$ and[1] $B_{i+1} \not\succeq_{\mathbf{A}_1} W_i \not\succeq_{\mathbf{A}_1} B_{i-1}$, for each i in $\{1, \ldots, k\}$; then, for each i and j in $\{1, \ldots, k\}$ such that $i \leq j$ we perform the screening of W_j against B_i. When $i = j$ the screening is 2-dimensional, and takes $\mathcal{O}(|B_i \cup W_j| \log |B_i|)$ time; when $i < j$ the screening is 1-dimensional and takes $\mathcal{O}(|B_i \cup W_j|)$ time. All the 1-dimensional screenings take $\mathcal{O}(b + w)$ overall, while the 2-dimensional ones take $\mathcal{O}((b + w) \log b)$.

□

Algorithm 6 (PScreen) shows how to perform p-screening in $\mathcal{O}\left((b + w) \log^{d-2} b\right)$ when $d \geq 3$. The algorithm is inspired by the one proposed by Kung, Luccio and Preparata [59]: in order to perform the p-screening $\left[\begin{smallmatrix} B \\ W \end{smallmatrix}\right]_\pi$, we split B in two halves, namely B_b and B_w, ensuring that no tuple in B_w dominates or is indistinguishable from any tuple in B_b ($B_w \not\succeq_\pi B_b$). We can obtain B_b and B_w with the same strategy we used in Dc: we select an attribute $\mathbf{A}$, making sure that all tuples in *both* B *and* W agree w.r.t all the attributes in $\mathcal{A}\text{nc}_\pi(\mathbf{A})$. Then we find the median tuple m^* w.r.t. $\succ_{\mathbf{A}}$ and split B accordingly. Clearly $B_w \not\succeq_\pi B_b$ holds, as all tuples in B_b are better than all tuples in B_w w.r.t. $\succ_{\mathbf{A}}$, while the preference on $\mathbf{A}$ is not overridden by other higher-priority attributes. Then we proceed splitting W into three chunks: the set W_b of tuples being preferred to m^* (according to $\succ_{\mathbf{A}}$), those being indistinguishable from m^* (W_e) and those being dominated (W_w). Since we chose $\mathbf{A}$ so that all tuples in *both* B and W agree w.r.t. all attributes in $\mathcal{A}\text{nc}_\pi(\mathbf{A})$, we are guaranteed that no tuple in B_w can dominate (or be indistinguishable from) any tuple in W_b[2]. Hence, the

[1] Here we assume $B_0 = B_{k+1} = \emptyset$

[2] Notice that for every pair of tuples (t', t) in $B_w \times W_b$ attribute $\mathbf{A}$ belongs to both $\mathcal{T}op_\pi(t', t)$

Algorithm 6: PScreen

Input: a p-expression π, two relation instances, B and W, s.t. $W \not\succeq_\pi B$
Output: the set W' of tuples in W that are not dominated by any tuple in B

Procedure PScreen*(π, B, W)*
  return PScreenRec$(\pi, B, W, \mathcal{R}\text{oots}_\pi, \emptyset)$
Procedure PScreenRec*(π, B, W, $\mathcal{C}$, $\mathcal{E}$)*
  if $\mathcal{C}$ *is empty* **then**
    return $\emptyset$
  else if $|B| = 1$ **then**
    return PScreenSP(π, B, W)
  else if $|\mathcal{C} \cup \mathcal{D}esc_\pi(\mathcal{C})| \leq 3$ **then**
    apply Lemma 6 and **return**
  select an attribute $\mathbf{A}$ from the candidates set $\mathcal{C}$
  if *all tuples in B assign the value a to* $\mathbf{A}$ **then**
    $(W_b, W_e, W_w) \leftarrow$ SplitByValue$(W, \mathbf{A}, a)$
    $\mathcal{C}' \leftarrow \mathcal{C} \setminus \{\mathbf{A}\}$
    $W'_w \leftarrow$ PScreenRec$(\pi, B, W_w, \mathcal{C}', \mathcal{E})$
    $\mathcal{E}' \leftarrow \mathcal{E} \cup \{\mathbf{A}\}$
    $\mathcal{C}'' \leftarrow \mathcal{C}' \cup \{\mathbf{V} \in \mathcal{S}\text{ucc}_\pi(\mathbf{A}) : \mathcal{P}\text{re}_\pi(\mathbf{V}) \subseteq \mathcal{E}'\}$
    $W'_e \leftarrow$ PScreenRec$(\pi, B, W_e, \mathcal{C}'', \mathcal{E}')$
    return $W'_w \cup W'_e \cup W_b$
  else
    $(B_b, B_w, m^*) \leftarrow$ SplitByAttribute$(B, \mathbf{A})$
    $(W_b, W_e, W_w) \leftarrow$ SplitByValue$(W, \mathbf{A}, m^*)$
    $W'_b \leftarrow$ PScreenRec$(\pi, B_b, W_b, \mathcal{C}, \mathcal{E})$
    $W'_w \leftarrow$ PScreenRec$(\pi, B_w, W_w \cup W_e, \mathcal{C}, \mathcal{E})$
    $W''_w \leftarrow$ PScreenRec$(\pi, B_b, W'_w, \mathcal{C} \setminus \{\mathbf{A}\}, \mathcal{E})$
    return $W'_b \cup W''_w$
Procedure SplitByValue*(D, $\mathbf{A}$, m^*)*
  Compute the set $D_b = \{t \in D \mid t[\mathbf{A}] \succ_\mathbf{A} m^*\}$
  Compute the set $D_e = \{t \in D \mid t[\mathbf{A}] \approx_\mathbf{A} m^*\}$
  Compute the set $D_w = \{t \in D \mid m^* \succ_\mathbf{A} t[\mathbf{A}]\}$
  return (D_b, D_e, D_w)

problem of p-screening B and W is reduced to following three smaller sub-problems, as depicted in Figure B.2:

$$\text{(i)} \begin{bmatrix} B_b \\ W_b \end{bmatrix}_\pi \quad \text{(ii)} \begin{bmatrix} B_b \\ W_w \cup W_e \end{bmatrix}_\pi \quad \text{(iii)} \begin{bmatrix} B_w \\ W_w \cup W_e \end{bmatrix}_\pi$$

and $\mathcal{B}\text{etter}_\pi(t, t')$. From Proposition 1 we can conclude that $B_w \not\succeq_\pi W_b$ holds.

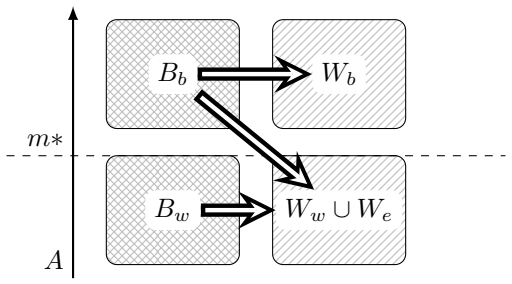


Figure B.2: Dividing p-screening into simpler subproblems. Each box represents a set of tuples, each arrow a p-screening operation.

We can solve these three sub-problems by recursion. The recursion will stop when we run out of attributes ($d \leq 3$) or tuples ($|B| \leq 1$). Notice that for sub-problem (ii) we do not need to take attribute **A** into consideration, hence, for that recursion branch, we reduced the dimensionality by one unit. In order to choose attribute **A** properly, PScreen adopts the same strategy of Dc: it keeps track of two sets of attributes, $\mathcal{C}$ and $\mathcal{E}$, ensuring the following invariants:

I1 : If an attribute belongs to $\mathcal{E}$ then no pair of tuples in $B \cup W$ can disagree on the value assigned to it. That is, all tuples in $B \cup W$ are indistinguishable with respect to all attributes in $\mathcal{E}$.

I2 : An attribute in $\mathcal{A} \setminus \mathcal{E}$ belongs to $\mathcal{C}$ if and only if all its ancestors belong to $\mathcal{E}$.

We can now analyze the pseudo-code of PScreen in more detail. Lines 20-25 define the core logic of the recursion, lines 4-18 handle the base-cases, while lines 26-30 contain auxiliary methods. The algorithm takes as input two sets of tuples, B and W, such that $W \not\succeq_\pi B$. At each iteration the algorithm selects an attribute **A** from $\mathcal{C}$ (line 10), and tests whether the tuples in B are distinguishable w.r.t. **A**. If they are, the block at lines 20-25 is executed, otherwise the one at lines 12-18. Let's analyze the first case. At line 20 B is split into B_b and B_w, at line 21 W is split into W_b, W_e and W_w. At lines 22, 23 and 24 the algorithm recurs three times, in order to solve the three sub-problems discussed above. We can now analyze the base-cases: when $\mathcal{C}$ is left empty (line 4), all tuples in W are dominated by all tuples in B, so the

algorithm returns the empty set; when B contains only one element, the algorithm applies the procedure from Lemma 2 (lines 6-7); when only three attributes are left to be processed, the algorithm applies Lemma 6. When it is not possible to split B, since all its tuples agree on some value a for $\mathbf{A}$, the algorithm splits W into three sets (W_b, W_e and W_w) containing tuples that are respectively better, indistinguishable from and worse than a, w.r.t. $\succ_{\mathbf{A}}$. Then it proceeds to solve $\left[\begin{smallmatrix} B \\ W_w \end{smallmatrix}\right]_\pi$ and $\left[\begin{smallmatrix} B \\ W_e \end{smallmatrix}\right]_\pi$. Notice $\mathcal{C}$ and $\mathcal{E}$ are updated at lines 13, 15 and 16 in order to satisfy the invariants I1 and I2.

Theorem B.1. *$F_d^*(b+w) \leq \mathcal{O}\left((b+w)\log^{d-2} b\right)$ for any $d > 3$, b and w.*

Proof. We show that the above upper bound holds if we run Algorithm 6 to perform p-screening. The proof is by induction on d, and will use the simplifying assumptions that (i) $b = 2^m$ for some positive m and (ii) the splitting operation at line 19 divides B into two almost equally sized halves. Since we proved the base case $d = 3$ in Lemma 6, the rest of the proof has a similar structure to the one proposed in [58, 59]. During each recursion Algorithm 6 splits the set W in two parts; using a notation similar to the one presented in Figure B.1, from now on we will denote by $w_{\ell,j}$ the size of the j-th portion of W obtained at depth ℓ in the recursion. Clearly $w_{0,0} = |W|$, and for each ℓ in $\{0, 1, \ldots, \log b - 1\}$ the sum $\sum_{j=1}^{2^\ell} w_{\ell,j}$ is equal to $|W|$. It is easy to see the following upper bound holds on F_d^*, for some fixed constant k_0

$$F_d^*(b, w_{0,0}) \leq k_0(b + w_{0,0}) + F_{d-1}^*\left(b, w_{0,0}\right) + F_d^*\left(\frac{b}{2}, w_{1,0}\right) + F_d^*\left(\frac{b}{2}, w_{1,1}\right) \qquad \text{(B.1)}$$

the linear term $k_0(b + w_{0,0})$ models the time spent for finding the median m^* and split B and W accordingly (lines 20 and 21), and the other three terms model the recursive calls at lines 22, 23, and 24. Inequality B.1 holds during each recursion,

therefore

$$\begin{aligned} F_d^*\left(\frac{b}{2^\ell}, w_{\ell,j}\right) &\leq k_0\left(\frac{b}{2^\ell} + w_{\ell,j}\right) + F_{d-1}^*\left(\frac{b}{2^\ell}, w_{\ell,j}\right) \\ &+ F_d^*\left(\frac{b}{2^{\ell+1}}, w_{\ell+1,2j}\right) \\ &+ F_d^*\left(\frac{b}{2^{\ell+1}}, w_{\ell+1,2j+1}\right) \end{aligned} \tag{B.2}$$

We can apply (B.2) to the last two terms of (B.1) and repeat the operation until we obtain the following

$$F_d^*(b,w) \leq \sum_{\hat{\ell}=0}^{\log b-1} \sum_{k=1}^{2^{\hat{\ell}}} k_0\left(\frac{b}{2^{\hat{\ell}}} + w_{\hat{\ell},k}\right) + \sum_{\hat{\ell}=0}^{\log b-1} \sum_{k=1}^{2^{\hat{\ell}}} F_{d-1}^*\left(\frac{b}{2^{\hat{\ell}}}, w_{\hat{\ell},k}\right) + \sum_{k=0}^{2^{\log b-1}} F_d^*(1, w_{\log b-1,k})$$

By inductive hypothesis, we can assume

$$F_{d-1}^*(b,w) \leq \mathcal{O}\left((b+w)(\log b)^{d-3}\right)$$

Also, from Lemma 2 we know $F_d^*(1,w)$ is $\mathcal{O}(w)$. Therefore, there exist constants k_0, k_1 and k_2 such that, for every b and w, the following holds

$$F_d^*(b,w) \leq k_0 \cdot (b+w)\log b + k_1 \cdot \sum_{\hat{\ell}=0}^{\log b-1} \sum_{j=1}^{2^{\hat{\ell}}} \left(\frac{b}{2^\ell} + w_{\hat{\ell},j}\right) \cdot \left(\log \frac{b}{2^\ell}\right)^{d-3} + k_2 \cdot w$$

Solving the sum over j, we get

$$F_d^*(b,w) \leq k_0 \cdot (b+w)\log b + k_1 \cdot (b+w) \cdot \sum_{\hat{\ell}=0}^{\log b-1} \left(\log b - \hat{\ell}\right)^{d-3} + k_2 \cdot w$$

If we set $h = \log b - \hat{\ell}$, we obtain

$$F_d^*(b,w) \leq k_0 \cdot (b+w)\log b + k_1 \cdot (b+w) \cdot \sum_{h=1}^{\log b} h^{d-3} + k_2 \cdot w$$

Since $\sum_{h=1}^{\log b} h^{d-3}$ is $\mathcal{O}\left((\log b)^{d-2}\right)$, it follows that

$$F_d^*(b,w) \;\leq\; \mathcal{O}\left((b+w)\cdot(\log b)^{d-2}\right)$$

This concludes our proof. □

B.2 Chapter III

B.2.1 Proof of Theorem III.1

Proof. From the Bayes' rule we have that

$$p[\boldsymbol{\theta}|\varphi,\mathcal{H}] = \frac{p[\boldsymbol{\theta},\varphi|\mathcal{H}]}{\mathbb{P}[\varphi|\mathcal{H}]} = \frac{\mathbb{P}[\varphi|\boldsymbol{\theta}]\cdot p[\boldsymbol{\theta}|\mathcal{H}]}{\mathbb{P}[\varphi|\mathcal{H}]}$$

Where

$$\begin{aligned}
\mathbb{P}[\varphi|\boldsymbol{\theta}] &= \sum_{w:w\models\varphi}\mathbb{P}[w|\boldsymbol{\theta}] = \sum_{w:w\models\varphi}\left[\prod_{i=1}^{n}\theta_i^{w[i]}\cdot\overline{\theta}_i^{\overline{w[i]}}\right]\\
p[\boldsymbol{\theta}|\mathcal{H}] &= \prod_{i=1}^{n}p[\theta_i|\mathcal{H}] = \prod_{i=1}^{n}\frac{\theta_i^{(a_i-1)}\cdot\overline{\theta}_i^{(b_i-1)}}{B(a_i,b_i)}\\
\mathbb{P}[\varphi|\mathcal{H}] &= \sum_{w:w\models\varphi}\left[\prod_{i=1}^{n}\left(\frac{a_i}{a_i+b_i}\right)^{w[i]}\cdot\left(\frac{b_i}{a_i+b_i}\right)^{\overline{w[i]}}\right]
\end{aligned}$$

Hence

$$p[\boldsymbol{\theta}|\varphi,\mathcal{H}] = \frac{\mathbb{P}[\varphi|\boldsymbol{\theta}]\cdot p[\boldsymbol{\theta}|\mathcal{H}]}{\mathbb{P}[\varphi|\mathcal{H}]} = \tfrac{1}{\mathbb{P}[\varphi|\mathcal{H}]}\cdot\sum_{w:w\models\varphi}(\mathbb{P}[w|\boldsymbol{\theta}]\cdot p[\boldsymbol{\theta}|\mathcal{H}])$$

First we want to prove that

$$\mathbb{P}[w|\boldsymbol{\theta}]\cdot p[\boldsymbol{\theta}|\mathcal{H}] = \mathbb{P}[w|\mathcal{H}]\cdot\prod_{i=1}^{n}\mathcal{B}e(a_i+w[i],b_i+\overline{w[i]})$$

Expanding $\mathbb{P}[w|\boldsymbol{\theta}]$ and $p[\boldsymbol{\theta}|\mathcal{H}]$ in the LHS we obtain

$$\begin{aligned}
\mathbb{P}[w|\boldsymbol{\theta}] \cdot p[\boldsymbol{\theta}|\mathcal{H}] &= \prod_{i=1}^{n} \theta_i^{w[i]} \cdot \overline{\theta_i}^{\overline{w[i]}} \cdot \prod_{i=1}^{n} \frac{\theta_i^{(a_i-1)} \cdot \overline{\theta_i}^{(b_i-1)}}{B(a_i, b_i)} \\
&= \prod_{i=1}^{n} \frac{\theta_i^{(a_i+w[i])-1} \cdot \overline{\theta_i}^{(b_i+\overline{w[i]})-1}}{B(a_i, b_i)} \\
&= \prod_{i=1}^{n} \mathcal{B}e(a_i + w[i], b_i + \overline{w[i]}) \cdot \frac{B(a_i + w[i], b_i + \overline{w[i]})}{B(a_i, b_i)} \\
&= \prod_{i=1}^{n} \mathcal{B}e(a_i + w[i], b_i + \overline{w[i]}) \cdot \frac{\Gamma(a_i + w[i]) \cdot \Gamma(b_i + \overline{w[i]}) \cdot \Gamma(a_i + b_i)}{\Gamma(a_i) \cdot \Gamma(b_i) \cdot \Gamma(a_i + b_i + 1)} \\
&= \prod_{i=1}^{n} \mathcal{B}e(a_i + w[i], b_i + \overline{w[i]}) \cdot \frac{(a_i)^{w[i]} \cdot (b_i)^{\overline{w[i]}} \cdot \Gamma(a_i + b_i)}{\Gamma(a_i + b_i + 1)} \\
&= \prod_{i=1}^{n} \mathcal{B}e(a_i + w[i], b_i + \overline{w[i]}) \cdot \frac{(a_i)^{w[i]} \cdot (b_i)^{\overline{w[i]}}}{a_i + b_i} \\
&= \prod_{i=1}^{n} \mathcal{B}e(a_i + w[i], b_i + \overline{w[i]}) \cdot \left(\frac{a_i}{a_i + b_i}\right)^{w[i]} \cdot \left(\frac{b_i}{a_i + b_i}\right)^{\overline{w[i]}} \\
&= \mathbb{P}[w|\mathcal{H}] \cdot \prod_{i=1}^{n} \mathcal{B}e(a_i + w[i], b_i + \overline{w[i]})
\end{aligned}$$

Therefore we can express the posterior $p[\boldsymbol{\theta}|\varphi, \mathcal{H}]$ as follows:

$$\begin{aligned}
p[\boldsymbol{\theta}|\varphi, \mathcal{H}] &= \tfrac{1}{\mathbb{P}[\varphi|\mathcal{H}]} \cdot \sum_{w:w\models\varphi} \mathbb{P}[w|\mathcal{H}] \cdot \prod_{i=1}^{n} \mathcal{B}e(a_i + w[i], b_i + \overline{w[i]}) \\
&= \sum_{w:w\models\varphi} \mathbb{P}[w|\varphi, \mathcal{H}] \cdot \left(\prod_{i=1}^{n} \mathcal{B}e(a_i + w[i], b_i + \overline{w[i]})\right)
\end{aligned}$$

Notice that $\sum_{w:w\models\varphi} \mathbb{P}[w|\varphi, \mathcal{H}] = 1$, therefore the posterior $p[\boldsymbol{\theta}|\varphi, \mathcal{H}]$ is a convex combination (a *mixture*) of products of Beta distributions. When we marginalize it w.r.t.

the parameter θ_i, we obtain the following:

$$
\begin{aligned}
p[\theta_i|\varphi,\mathcal{H}] &= \int_0^1 .. \int_0^1 p[\boldsymbol{\theta}|\varphi,\mathcal{H}]\ d\theta_1..d\theta_{i-1}d\theta_{i+1}..d\theta_m \\
&= \sum_{w:w\models\varphi} \mathbb{P}[w|\varphi,\mathcal{H}]\cdot\mathcal{B}e(a_i+w[i],b_i+\overline{w[i]})\cdot \\
&\quad \cdot \int_0^1 .. \int_0^1 \prod_{j\in\{1,..,m\}\setminus\{i\}} \mathcal{B}e(a_j+w[j],b_j+\overline{w[j]})\ d\theta_1..d\theta_{i-1}d\theta_{i+1}..d\theta_m \\
&= \sum_{w:w\models\varphi} \mathbb{P}[w|\varphi,\mathcal{H}]\cdot\mathcal{B}e(a_i+w[i],b_i+\overline{w[i]})\cdot 1 \\
&= \mathbb{P}[x_i|\varphi,\mathcal{H}]\cdot\mathcal{B}e(a_i+1,b_i)+\mathbb{P}[\overline{x_i}|\varphi,\mathcal{H}]\cdot\mathcal{B}e(a_i,b_i+1)
\end{aligned}
$$

This concludes our proof. □

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] TPC-H benchmark. `http://www.tpc.org/tpch/`.

[2] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[3] P. Afshani. Fast computation of output-sensitive maxima in a word RAM. In C. Chekuri, editor, *SODA*, pages 1414–1423. SIAM, 2014.

[4] P. Agrawal, O. Benjelloun, A. D. Sarma, C. Hayworth, S. U. Nabar, T. Sugihara, and J. Widom. Trio: A system for data, uncertainty, and lineage. In *Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006*, pages 1151–1154, 2006.

[5] L. Antova, T. Jansen, C. Koch, and D. Olteanu. Fast and simple relational processing of uncertain data. In *Proceedings of the 24th International Conference on Data Engineering, ICDE 2008, April 7-12, 2008, Cancún, México*, pages 983–992, 2008.

[6] I. Bartolini, P. Ciaccia, and M. Patella. Efficient sort-based skyline evaluation. *ACM Trans. Database Syst.*, 33(4):31:1–31:49, Dec. 2008.

[7] O. Benjelloun, A. D. Sarma, A. Y. Halevy, and J. Widom. ULDBs: Databases with uncertainty and lineage. In *Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006*, pages 953–964, 2006.

[8] J. L. Bentley. Multidimensional divide-and-conquer. *Commun. ACM*, 23(4):214–229, 1980.

[9] J. L. Bentley, K. L. Clarkson, and D. B. Levine. Fast linear expected-time algorithms for computing maxima and convex hulls. In D. S. Johnson, editor, *SODA*, pages 179–187. SIAM, 1990.

[10] J. L. Bentley, D. Haken, and J. B. Saxe. A general method for solving divide-and-conquer recurrences. *SIGACT News*, 12(3):36–44, Sept. 1980.

[11] J. L. Bentley, H. T. Kung, M. Schkolnick, and C. D. Thompson. On the average number of maxima in a set of vectors and applications. *J. ACM*, 25(4):536–543, 1978.

[12] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *Proceedings of the 17th International Conference on Data Engineering*, pages 421–430, 2001.

[13] J. Boulos, N. N. Dalvi, B. Mandhani, S. Mathur, C. Ré, and D. Suciu. MYSTIQ: a system for finding more answers by using probabilities. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005*, pages 891–893, 2005.

[14] C. Buchta. On the average number of maxima in a set of vectors. *Inf. Process. Lett.*, 33(2):63–65, 1989.

[15] P. Buneman, S. Khanna, and W. C. Tan. Why and where: A characterization of data provenance. In *Database Theory - ICDT 2001, 8th International Conference, London, UK, January 4-6, 2001, Proceedings.*, pages 316–330, 2001.

[16] Z. Cai, Z. Vagena, L. L. Perez, S. Arumugam, P. J. Haas, and C. M. Jermaine. Simulation of database-valued Markov chains using SimSQL. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013*, pages 637–648, 2013.

[17] T. M. Chan. Optimal output-sensitive convex hull algorithms in two and three dimensions. *Discrete & Computational Geometry*, 16(4):361–368, 1996.

[18] T. M. Chan, K. G. Larsen, and M. Pătraşcu. Orthogonal range searching on the RAM, revisited. In *Proceedings of the Twenty-seventh Annual Symposium on Computational Geometry*, SoCG '11, pages 1–10, New York, NY, USA, 2011. ACM.

[19] J. Chomicki. Querying with intrinsic preferences. In C. S. Jensen, K. G. Jeffery, J. Pokorný, S. Saltenis, E. Bertino, K. Böhm, and M. Jarke, editors, *EDBT*, volume 2287 of *Lecture Notes in Computer Science*, pages 34–51. Springer, 2002.

[20] J. Chomicki. Preference formulas in relational queries. *ACM Trans. Database Syst.*, 28(4):427–466, Dec. 2003.

[21] J. Chomicki, P. Ciaccia, and N. Meneghetti. Skyline queries, front and back. *SIGMOD Record*, 42(3):6–18, 2013.

[22] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with presorting. In *ICDE*, pages 717–719, 2003.

[23] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms (3. ed.)*. MIT Press, 2009.

[24] N. N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *(e)Proceedings of the Thirtieth International Conference on Very Large Data Bases, Toronto, Canada, August 31 - September 3 2004*, pages 864–875, 2004.

[25] N. N. Dalvi and D. Suciu. The dichotomy of probabilistic inference for unions of conjunctive queries. *J. ACM*, 59(6):30, 2012.

[26] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *JRSS-B*, pages 1–38, 1977.

[27] M. Dylla and M. Theobald. Learning tuple probabilities in probabilistic databases. Technical report, Max-Planck-Institut für Informatik, 2014.

[28] A. Erdélyi, W. Magnus, F. Oberhettinger, and F. G. Tricomi. *Tables of Integral Transforms: Vol.: 1.* McGraw-Hill Book Company, Incorporated, 1954.

[29] R. Fink, A. Hogue, D. Olteanu, and S. Rath. SPROUT^2: a squared query engine for uncertain web data. In *SIGMOD*, 2011.

[30] R. Fink, J. Huang, and D. Olteanu. Anytime approximation in probabilistic databases. *VLDB J.*, 22(6):823–848, 2013.

[31] N. Fuhr and T. Rölleke. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM TOIS*, 15(1):32–66, 1997.

[32] H. N. Gabow, J. L. Bentley, and R. E. Tarjan. Scaling and related techniques for geometry problems. In R. A. DeMillo, editor, *STOC*, pages 135–143. ACM, 1984.

[33] W. Gatterbauer and D. Suciu. Oblivious bounds on the probability of boolean functions. *ACM TODS*, 39(1):5, 2014.

[34] P. Godfrey. Skyline cardinality for relational processing. In D. Seipel and J. M. T. Torres, editors, *FoIKS*, volume 2942 of *Lecture Notes in Computer Science*, pages 78–97. Springer, 2004.

[35] P. Godfrey, R. Shipley, and J. Gryz. Maximal vector computation in large data sets. In K. Böhm, C. S. Jensen, L. M. Haas, M. L. Kersten, P.-Å. Larson, and B. C. Ooi, editors, *VLDB*, pages 229–240. ACM, 2005.

[36] M. C. Golumbic, A. Mintz, and U. Rotics. Factoring and recognition of read-once functions using cographs and normality and the readability of functions associated with partial k-trees. *DAM*, 154(10):1465–1477, 2006.

[37] T. J. Green, G. Karvounarakis, Z. G. Ives, and V. Tannen. Provenance in ORCHESTRA. *IEEE Data Eng. Bull.*, 33(3):9–16, 2010.

[38] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In L. Libkin, editor, *PODS*. ACM, 2007.

[39] B. Gutmann, A. Kimmig, K. Kersting, and L. D. Raedt. Parameter learning in probabilistic databases: A least squares approach. In *ECML/PKDD*, pages 473–488, 2008.

[40] B. Gutmann, I. Thon, and L. D. Raedt. Learning the parameters of probabilistic logic programs from interpretations. In *ECML/PKDD*, pages 581–596, 2011.

[41] H. Hartley. Maximum likelihood estimation from incomplete data. *Biometrics*, 14(2):174–194, 1958.

[42] R. Herbrich. Minimising the Kullback–Leibler divergence. Technical report, Microsoft Research, 2005.

[43] X. Hu, C. Sheng, Y. Tao, Y. Yang, and S. Zhou. Output-sensitive skyline algorithms in external memory. In S. Khanna, editor, *SODA*, pages 887–900. SIAM, 2013.

[44] J. Huang, L. Antova, C. Koch, and D. Olteanu. MayBMS: a probabilistic database management system. In *SIGMOD*, 2009.

[45] T. Imielinski and W. L. Jr. On representing incomplete information in a relational data base. In *Very Large Data Bases, 7th International Conference, September 9-11, 1981, Cannes, France, Proceedings*, pages 388–397. IEEE Computer Society, 1981.

[46] R. Jampani, F. Xu, M. Wu, L. L. Perez, C. M. Jermaine, and P. J. Haas. MCDB: a monte carlo approach to managing uncertain data. In *SIGMOD*, 2008.

[47] B. Jiang, J. Pei, X. Lin, and Y. Yuan. Probabilistic skylines on uncertain data: model and bounding-pruning-refining methods. *J. Intell. Inf. Syst.*, 38(1):1–39, 2012.

[48] N. L. Johnson, S. Kotz, and N. Balakrishnan. Continuous univariate distributions, vol. 2, 1995.

[49] R. M. Karp, M. Luby, and N. Madras. Monte-carlo approximation algorithms for enumeration problems. *J. Algorithms*, 10(3):429–448, 1989.

[50] O. Kennedy and C. Koch. PIP: A database system for great and small expectations. In *ICDE*, 2010.

[51] W. Kießling. Foundations of preferences in database systems. In *VLDB*, pages 311–322. Morgan Kaufmann, 2002.

[52] W. Kießling, B. Hafenrichter, S. Fischer, and S. Holland. Preference XPATH: A query language for e-commerce. In H. U. Buhl, A. Huther, and B. Reitwiesner, editors, *Wirtschaftsinformatik*, page 32. Physica Verlag / Springer, 2001.

[53] W. Kießling and G. Köstler. Preference SQL - design, implementation, experiences. In *VLDB*, pages 990–1001. Morgan Kaufmann, 2002.

[54] D. G. Kirkpatrick and R. Seidel. Output-size sensitive algorithms for finding maximal vectors. In *Symposium on Computational Geometry*, pages 89–96, 1985.

[55] D. G. Kirkpatrick and R. Seidel. The ultimate planar convex hull algorithm? *SIAM J. Comput.*, 15(1):287–299, 1986.

[56] C. Koch and D. Olteanu. Conditioning probabilistic databases. *PVLDB*, 1(1):313–325, 2008.

[57] D. Koller. Probabilistic relational models. In *ILP-99*, pages 3–13, 1999.

[58] H. T. Kung. On the computational complexity of finding the maxima of a set of vectors. In *SWAT (FOCS)*, pages 117–121, 1974.

[59] H. T. Kung, F. Luccio, and F. P. Preparata. On finding the maxima of a set of vectors. *J. ACM*, 22(4):469–476, 1975.

[60] K. B. Laskey. MEBN: A language for first-order bayesian knowledge bases. *Artif. Intell.*, 172(2-3):140–178, 2008.

[61] A. C. G. V. Lazo and P. N. Rathie. On the entropy of continuous probability distributions (corresp.). *IEEE TOIT*, 24(1):120–122, 1978.

[62] J. Liu, L. Xiong, and X. Xu. Faster output-sensitive skyline computation algorithm. *Inf. Process. Lett.*, 114(12):710–713, 2014.

[63] N. Meneghetti, D. Mindolin, P. Ciaccia, and J. Chomicki. Output-sensitive evaluation of prioritized skyline queries. In T. Sellis, S. B. Davidson, and Z. Ives, editors, *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, pages 1955–1967. ACM, 2015.

[64] D. Mindolin and J. Chomicki. Preference elicitation in prioritized skyline queries. In *VLDB J.*, pages 157–182, 2011.

[65] R. M. Neal and G. E. Hinton. A view of the EM algorithm that justifies incremental, sparse, and other variants. In *Learning in graphical models*, pages 355–368. Springer, 1998.

[66] D. Olteanu and J. Huang. Using OBDDs for efficient query evaluation on probabilistic databases. In *SUM*, 2008.

[67] D. Olteanu, J. Huang, and C. Koch. Approximate confidence computation in probabilistic databases. In *ICDE*, 2010.

[68] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. *ACM Trans. Database Syst.*, 30(1):41–82, 2005.

[69] L. D. Raedt, A. Kimmig, and H. Toivonen. ProbLog: A probabilistic prolog and its application in link discovery. In *IJCAI*, pages 2462–2467, 2007.

[70] M. Richardson and P. M. Domingos. Markov logic networks. *Machine Learning*, 62(1-2):107–136, 2006.

[71] P. Sen, A. Deshpande, and L. Getoor. PrDB: managing and exploiting rich correlations in probabilistic databases. *VLDB J.*, 18(5):1065–1090, 2009.

[72] C. Sheng and Y. Tao. Worst-case i/o-efficient skyline algorithms. *ACM Transactions on Database Systems (TODS)*, 37(4):26, 2012.

[73] W. Siberski, J. Z. Pan, and U. Thaden. Querying the semantic web with preferences. In I. F. Cruz, S. Decker, D. Allemang, C. Preist, D. Schwabe, P. Mika, M. Uschold, and L. Aroyo, editors, *International Semantic Web Conference*, volume 4273 of *Lecture Notes in Computer Science*, pages 612–624. Springer, 2006.

[74] S. Singh, C. Mayfield, S. Mittal, S. Prabhakar, S. E. Hambrusch, and R. Shah. Orion 2.0: native support for uncertain data. In *SIGMOD*, 2008.

[75] K. Stefanidis, G. Koutrika, and E. Pitoura. A survey on representation, composition and application of preferences in database systems. *ACM Trans. Database Syst.*, 36(3):19:1–19:45, Aug. 2011.

[76] J. Stoyanovich, S. B. Davidson, T. Milo, and V. Tannen. Deriving probabilistic databases with inference ensembles. In *ICDE*, pages 303–314, 2011.

[77] D. Suciu, D. Olteanu, C. Ré, and C. Koch. *Probabilistic Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.

[78] W. Wei, J. Erenrich, and B. Selman. Towards efficient sampling: Exploiting random walk strategies. In D. L. McGuinness and G. Ferguson, editors, *AAAI*, pages 670–676. AAAI Press / The MIT Press, 2004.