The VLDB Journal
DOI 10.1007/s00778-013-0310-5

REGULAR PAPER

Anytime approximation in probabilistic databases

Robert Fink · Jiewen Huang · Dan Olteanu

Received: 18 June 2012 / Revised: 24 February 2013 / Accepted: 27 February 2013

Abstract This article describes an approximation algorithm for computing the probability of propositional formulas over discrete random variables. It incrementally refines lower and upper bounds on the probability of the formulas until the desired absolute or relative error guarantee is reached. This algorithm is used by the SPROUT query engine to approximate the probabilities of results to relational algebra queries on expressive probabilistic databases.

Keywords Probabilistic databases · Query evaluation · Anytime approximation · Model-based approximation

1 Introduction

The problem of query evaluation in probabilistic databases has received tremendous attention in recent years [53].

The probabilistic database formalism considered in this article is that of pc-tables, where each input tuple is annotated by a propositional formula over independent discrete random variables [53]. Such formulas define the probabilities of database tuples and can express arbitrary correlations between tuples, such as conditional independence and positive and negative correlations. This formalism subsumes tuple-independent probabilistic databases, such as NELL tables learned from the web [7], and block-independent disjoint databases, such as Google Squared tables that aggregate Google Search results [18] and probabilistic tables obtained from information extraction models [26]. It can also capture more complex correlations that arise when conditioning probabilistic databases using constraints [35].

The query language considered in this article is that of relational algebra. Following the standard deterministic case where queries map between relational databases, in our setting queries map between pc-tables: The query results are pc-tables, whose tuples are computed as in the deterministic case and their associated formulas are constructed from the input formulas using conjunction, disjunction, and negation, and reflect the derivation of the result from the input [28].

When contrasted to the deterministic case, the key novel challenge in the probabilistic setting is the computation of probabilities of query results or, equivalently, the probability computation for the propositional formulas that annotate these results. This latter problem is known to be #P-hard already for positive bipartite formulas in disjunctive normal form [45]. In database terms, the query evaluation problem has #P-hard data complexity already for a limited form of conjunctive queries without self-joins (i.e., conjunctive queries without repeating relation symbols) on tuple-independent probabilistic databases, where the annotations of the input tuples are independent random variables [11].

Approximate query evaluation is preferred over exact query evaluation in cases where fast approximate answers are favoured over delayed exact answers. This article describes the approximate probability computation algorithm used

Electronic supplementary material The online version of this article (doi:10.1007/s00778-013-0310-5) contains supplementary material, which is available to authorized users.

This research was funded by European Research Consortium grant agreement FOX number FP7-ICT-233599.

R. Fink · D. Olteanu (✉)
University of Oxford, Oxford, UK
e-mail: dan.olteanu@cs.ox.ac.uk

R. Fink
e-mail: robert.fink@cs.ox.ac.uk

J. Huang
Yale University, New Haven, CT, USA
e-mail: jiewen.huang@yale.edu

Published online: 02 July 2013

by the SPROUT query engine [18,27] and thereby unifies aspects of this algorithm that have previously been reported separately [19,20,41]. The algorithm works for *arbitrary propositional formulas* and thus for any relational algebra query and pc-table, it is *anytime*, and *admits absolute and relative error guarantees*. Given a formula Φ whose exact probability is p, it computes a probability $\hat{p}$ such that

(i) $p - \epsilon \leq \hat{p} \leq p + \epsilon$ for a given absolute error ϵ, or

(ii) $p(1 - \epsilon) \leq \hat{p} \leq p(1 + \epsilon)$ for a given relative error ϵ.

More precisely, the algorithm can compute a contiguous interval of approximations $\hat{p}$ within the given error bounds. It starts with trivial lower and upper bounds [0, 1] and incrementally tightens the interval until the approximation is reached. It is thus an anytime algorithm, since it returns an approximation even if it is stopped before it ends and finds increasingly better bounds the longer it runs.

The algorithm is an interplay of two techniques: (1) efficient computation of lower and upper bounds on the probability of a formula and (2) decomposition of a formula into an equivalent disjunction or conjunction of independent or mutually exclusive simpler formulas. Given bounds on the probability of the simpler formulas, we can then efficiently compute bounds on the probability of the original formula.

The algorithm works as follows: Given a formula, we first compute probability bounds via the first technique. If the bounds are tight enough to meet the error guarantee, then we stop; otherwise, we decompose the formula into simpler formulas using the second technique, compute probability bounds for the simpler formulas using the first technique and use them to derive bounds for the original formula. This decomposition step is repeated until the error guarantee is met. By exhaustively decomposing the formula down to literals as approached by knowledge compilation techniques [13], we can compute its exact probability.

The first technique can compute probability bounds without error guarantees in polynomial time. The second technique can compute bounds with error guarantees, and although each decomposition step takes polynomial time, it may require exponentially many steps to reach the desired approximation. The quality of bounds computed by the first technique is crucial since good bounds can drastically decrease the number of decomposition steps performed by the second technique. Our approach is to derive such bounds via model-based approximation as explained next.

Given a formula Φ, let $\mathscr{M}(\Phi)$ denote the set of models (or satisfying assignments) of Φ. A pair of formulas Φ_L and Φ_U from a language $\mathscr{L}'$ of propositional formulas represents a model-based approximation of Φ if $\mathscr{M}(\Phi_L) \subseteq \mathscr{M}(\Phi) \subseteq \mathscr{M}(\Phi_U)$, that is, the set of models of Φ includes the set of models of Φ_L and is included in the set of models of Φ_U. We call Φ_L and Φ_U lower and upper bounds for Φ, respectively. The lower bound Φ_L is optimal if there is no other formula Φ'_L in $\mathscr{L}'$ such that $\mathscr{M}(\Phi_L) \subset \mathscr{M}(\Phi'_L) \subseteq \mathscr{M}(\Phi)$; the case of optimal upper bounds is symmetric. Our interest in model bounds stems from the observation that model bounds imply probability bounds: $P(\Phi_L) \leq P(\Phi) \leq P(\Phi_U)$. The bounds Φ_L and Φ_U are chosen such that they are optimal in a more restrictive language $\mathscr{L}'$ of formulas for which probability computation can be done in polynomial time. In this article, we consider the language of so-called read-once formulas, called 1OF for short, and its subset of positive formulas in disjunctive normal form, called iDNF for short. These languages are defined in Sect. 2.

There are two main observations behind the design of our algorithm. Firstly, sufficient approximations can often be obtained within a few decomposition steps and there is thus no need to exhaustively decompose the input formula down to literals. This motivates the incremental nature of the algorithm as well as the use of effective termination checks. According to our experiments, a relatively small number of well-chosen decomposition steps computable within a few seconds usually suffice to guarantee good precision. The formulas obtained after these decomposition steps may still be large, yet they account for a very small fraction of the overall probability mass. The second observation concerns tractable query evaluation in probabilistic databases. We can effectively derive orders of decomposition steps under which any approximation, and in particular, the exact probability can be obtained in polynomially many steps when decomposing formulas annotating results of tractable conjunctive query without self-joins. Most notably, this is achieved without a priori knowledge of the query or the input database.

In addition to the anytime algorithm, this article contributes a syntactic characterization of optimal iDNF lower and upper bounds for positive formulas in disjunctive normal form. We give algorithms that enumerate such bounds with polynomial delay and thus allow to inspect several lower and upper bounds and choose among them a pair whose probabilities are closest to the exact probability of the input formula. For formulas with clauses of arity at most k, such as those annotating results of positive relational queries on tuple-independent databases [53], we give a polynomial-time algorithm that can find an optimal iDNF lower bound whose probability is within a factor k from the largest probability of optimal iDNF lower bounds; finding the latter is NP-hard. For nested positive formulas, we show how to efficiently compute 1OF lower and upper bounds.

This article is organized as follows. Section 2 recalls background on propositional formulas and probabilistic databases. Sections 3 and 4 discuss model-based approximations and incremental decomposition of formulas. We report on experiments in Sect. 5, on related work in Sect. 6, and we conclude in Sect. 7. Proofs of formal statements and details on the experiments can be found in "Appendix".

2 Preliminaries

2.1 Propositional formulas

Syntax. Let $\mathbf{X}$ be a finite set of Boolean variables. A literal is a variable or its negation. A clause is a conjunction of literals. A formula can be constructed using variables and constants $\top$ (true) and $\bot$ (false) using the logical connectives $\vee$ ("or"), $\wedge$ ("and"), and $\neg$ ("not"). We use $\mathrm{vars}(\Phi)$ to denote the set of variables of the formula Φ. Two formulas Φ and Ψ are independent if their variable sets are disjoint, i.e., $\mathrm{vars}(\Phi) \cap \mathrm{vars}(\Psi) = \emptyset$; otherwise, they are dependent.

We assume that formulas are in negation normal form (NNF), where negation can only appear at literals; any formula can be brought in NNF in linear time. A formula (in NNF) is positive if all of its literals occur positively; unate formulas, i.e., formulas where each variable occurs either only positively or only negatively, can trivially be converted into equivalent positive formulas.

A formula is in disjunctive normal form (DNF) if it is a disjunction of clauses; it is in one-occurrence form (1OF) or read-once, if each of its variables occurs once; it is in independent DNF (iDNF) if it is in 1OF and DNF.

We alternatively see clauses as sets of literals, DNF formulas as sets of clauses, and arbitrary (NNF) formulas as their parse trees: Inner nodes are logical connectives, and leaves are variables. Given two DNF formulas (clauses) Φ and Ψ, $\Phi \subseteq \Psi$ means that Φ consists of a subset of the clauses (literals, respectively) of Ψ. The constants $\top$ and $\bot$ correspond to an empty set of literals and clauses, respectively. By using the set perspective, we implicitly assume that a clause (a literal) only appears once in a DNF formula (clause, respectively); indeed, for all our purposes, we can ignore duplicate clauses and literals; if duplicates are created during processing, then they are trivially removed.

Example 1 The formula $x_1y_1 \vee x_1y_2$ is in DNF, but not in iDNF, since its clauses share variable x_1. Its equivalent formula $x_1(y_1 \vee y_2)$ is in 1OF. Figures 6 and 7 show the parse trees of two formulas. If the marked leaves are removed, we obtain formulas in 1OF.

A DNF formula Φ has *arity* (*max-arity*) k if every clause in Φ has exactly (at most, respectively) k variables.

Semantics. Given the set $\mathbf{X}$ of variables, we denote by $\mathscr{I}$ the set of possible assignments of all variables from $\mathbf{X}$ to constants true and false. For a formula Φ, its set of assignments is denoted by $\mathscr{I}(\Phi) = \{I : \mathrm{vars}(\Phi) \rightarrow \{\mathrm{true}, \mathrm{false}\}\}$. A *model* of Φ is a satisfying assignment, i.e., an assignment I that maps Φ to true, also denoted by $I \models \Phi$. The set of models of Φ is denoted by $\mathscr{M}(\Phi)$.

Given two formulas Φ and Ψ over the same variables, Φ is *semantically* contained in Ψ, denoted by $\Phi \models \Psi$, if $\mathscr{M}(\Phi) \subseteq \mathscr{M}(\Psi)$. We also say that Φ *implies* Ψ. Equivalence is two-way containment and denoted by $\equiv$. Given two clauses Φ and ψ, it holds that $\varphi \models \psi$ if and only if $\varphi \supseteq \psi$. For DNF formulas, we use the following key result (formulated in terms of tableaux containment, Theorem 3 in [49]):

Lemma 1 ([49]) *Let Φ, Ψ be positive DNF formulas. Then,*

$$\Phi \models \Psi \quad \textit{iff} \quad \forall \varphi \in \Phi\ \exists \psi \in \Psi\ :\ \varphi \models \psi \tag{1}$$

$$\textit{iff} \quad \forall \varphi \in \Phi\ \exists \psi \in \Psi\ :\ \varphi \supseteq \psi. \tag{2}$$

A positive DNF formula Φ is *reducible* if there exist clauses $\varphi, \psi \in \Phi$ that satisfy $\varphi \models \psi$, in which case φ is redundant and can be removed; otherwise, it is *irreducible*.

Corollary 1 (Lemma 1) *Two irreducible positive DNF formulas are equivalent if and only if they have the same clauses.*

Computing the irreducible equivalent of a positive DNF formula can be done in polynomial time.

Probabilistic interpretation. Let $\mathbf{X}$ be a set of independent Boolean random variables. For each variable $x \in \mathbf{X}$, let P_x be the probability of x being true; we assume $P_x > 0$ without loss of generality. The probability mass function $\Pr(I) = \left[\prod_{x\in\mathbf{X}}^{I(x)=\mathrm{true}} P_x\right] \cdot \left[\prod_{x\in\mathbf{X}}^{I(x)=\mathrm{false}} (1 - P_x)\right]$ for each assignment $I \in \mathscr{I}$ and the probability measure $\Pr(E) = \sum_{I\in E} \Pr(I)$ for all $E \subseteq \mathscr{I}$ define the probability space $(\mathscr{I}, 2^{\mathscr{I}}, \Pr)$ that we call the *probability space induced by* $\mathbf{X}$.

A formula Φ can be interpreted as a random variable $\Phi : \mathscr{I} \rightarrow \{\mathrm{true}, \mathrm{false}\}$ over $(\mathscr{I}, 2^{\mathscr{I}}, \Pr)$ by letting $\Phi : I \mapsto I(\Phi)$ and with probability distribution defined as

$$\Pr(\Phi = \mathrm{true}) = \sum_{I\in\mathscr{I}, I(\Phi)=\mathrm{true}} \Pr(I). \tag{3}$$

We write P_Φ or $P(\Phi)$ for $\Pr(\Phi = \mathrm{true})$ and $P_{\neg\Phi}$ or $P(\neg\Phi)$ for $\Pr(\Phi = \mathrm{false}) = 1 - \Pr(\Phi = \mathrm{true})$.

The task of probability computation is #P-hard already for bipartite positive DNF formulas [45]. In contrast, formulas in 1OF and iDNF admit probability computation in linear time. Recall that a 1OF formula is a variable or a conjunction or disjunction of independent 1OF formulas. Computing the probability of a 1OF formula Φ is a trivial task:

- If $\Phi = \Phi_1 \wedge \cdots \wedge \Phi_n$, then $\Pr(\Phi) = \Pi_{i=1}^{n} \Pr(\Phi_i)$.
- If $\Phi = \Phi_1 \vee \cdots \vee \Phi_n$, then $\Pr(\Phi) = 1 - \Pi_{i=1}^{n}(1 - \Pr(\Phi_i))$.
- If $\Phi = x$ for a variable x, then $\Pr(\Phi) = P_x$.

2.2 Probabilistic databases

Syntax and semantics. Probabilistic c-multitables, or pc-tables for short, are relational databases where each tuple is annotated with a propositional formula over a set $\mathbf{X}$ of independent Boolean random variables [28,53]. Under possible worlds semantics, a pc-table $\mathscr{D}$ represents a finite set of possible worlds: Each valuation I of the variables in $\mathbf{X}$ defines a possible world representing a relational database consisting of exactly those tuples in $\mathscr{D}$ whose annotations are satisfied by I. The probability of each world is the product of probabilities of the variable assignments in I. This representation formalism is complete in the sense that it can represent arbitrary probability distributions over any finite set of possible worlds. It is a strong representation system since query results can be represented as pc-tables.

Query evaluation. Given a query Q and a pc-table $\mathscr{D}$, the query evaluation problem is to compute the distinct tuples in the results of Q in the worlds of $\mathscr{D}$ together with their probabilities. The probability $P(t \in Q(\mathscr{D}))$ of a tuple t is the probability that t is in the result of Q in a world randomly drawn from $\mathscr{D}$. We adopt the *intensional approach* to query evaluation [53]: For each answer tuple t, first construct a propositional formula Φ that annotates t such that $P(t \in Q(\mathscr{D})) = P(\Phi)$, then compute $P(\Phi)$.

The tuples together with their annotation in the result of a query Q can be computed directly from the input pc-table $\mathscr{D}$, thus circumventing the need to evaluate the query Q in each possible world of $\mathscr{D}$. This is achieved by rewriting Q into a query Q' such that standard relational evaluation of Q' on $\mathscr{D}$ yields a pc-table $\mathscr{D}'$ representing the results of Q in the worlds of $\mathscr{D}$. Figure 1 specifies such a rewriting function $\llbracket\cdot\rrbracket$ for any relational algebra query. It assumes that the formulas annotating the tuples in the input pc-table are stored in a distinguished column called Φ; for a relation R, we consider that this column is not selected by the selector `R.*`. The rewriting is expressed here in SQL and—besides a straightforward encoding of the relational operators in SQL—it constructs formulas annotating result tuples based on the formulas of input tuples as follows. In case of identity, selection, and renaming operators, the input annotations are just copied in the result. For projection, the formula of each distinct result tuple is constructed as the disjunction of all input tuples with the same restriction to the attributes in the projection list. For the join operator, the formula of a result tuple is the conjunction of the formulas of the contributing input tuples; to avoid cluttering, we slightly abuse notation in stating the attributes of the select clause: `R.*, S.*` means here the set-union of the attributes in R and S. The union operator is treated similarly to the projection operator, where the distinct result tuples are annotated by the disjunctions of equal input tuples. A tuple t in the result of $Q_1 - Q_2$ has annotation Φ_1 if t is in Q_1 with annotation Φ_1 and t is not in Q_2 and has annotation $\Phi_1 \wedge \neg\Phi_2$ if t is in Q_1 with annotation Φ_1 and t is in Q_2 with annotation Φ_2. These two cases are implemented in $\llbracket\cdot\rrbracket$ by a left outer join.

Given the annotated query result, the remaining query evaluation task is the computation of its probabilities, i.e., the probability $P(t \in Q(\mathscr{D})) = P(\Phi)$ for each result tuple t and its annotation Φ. In general, this problem is #P-hard, though there are known polynomial-time cases [53]. We refer to queries with #P-hard and PTIME data complexity as *intractable* and *tractable* queries, respectively.

Example 2 Figure 2 shows pc-tables, where each tuple is annotated by a distinct Boolean random variable stored in column Φ, and the formulas Φ_1 and Φ_2 annotating the results of two Boolean queries Q_1 and Q_2, respectively. The answers

$$
\begin{aligned}
\llbracket R\rrbracket &= \text{R}\\
\llbracket \delta_{A\to B}(Q)\rrbracket &= \text{select R.*, R.A as B, R.}\Phi\text{ from } (\llbracket Q\rrbracket)\text{ R}\\
\llbracket \sigma_{\varphi}(Q)\rrbracket &= \text{select R.* from } (\llbracket Q\rrbracket)\text{ R where } \varphi\\
\llbracket \pi_{A}(Q)\rrbracket &= \text{select R.A, } \textstyle\bigvee(\text{R.}\Phi) \text{ as } \Phi \text{ from } (\llbracket Q\rrbracket)\text{ R}\\
&\quad\ \text{group by R.A}\\
\llbracket Q_1 \bowtie_{\varphi} Q_2\rrbracket &= \text{select R.*, S.*, R.}\Phi\wedge\text{S.}\Phi\text{ as }\Phi\\
&\quad\ \text{from } (\llbracket Q_1\rrbracket)\text{ R}, (\llbracket Q_2\rrbracket)\text{ S where } \varphi\\
\llbracket Q_1 \cup Q_2\rrbracket &= \text{select R.*, } \textstyle\bigvee(\text{R.}\Phi) \text{ as } \Phi \text{ from}\\
&\quad\ \big(\text{select * from } (\llbracket Q_1\rrbracket) \text{ union all}\\
&\quad\quad \text{select * from } (\llbracket Q_2\rrbracket)\big) \text{ R group by R.*}\\
\llbracket Q_1 - Q_2\rrbracket &= \text{select R.*, R.}\Phi \wedge\neg\text{ S.}\Phi\text{ as }\Phi\\
&\quad\ \text{from } (\llbracket Q_1\rrbracket)\text{ R left out join } (\llbracket Q_2\rrbracket)\text{ S on R.* = S.*}
\end{aligned}
$$

Fig. 1 Query rewriting procedure for queries over pc-tables

R

A	Φ
1	x_1
2	x_2
3	x_3

S

A	B	Φ
1	1	y_1
1	2	y_2
2	1	y_3
2	2	y_4
3	3	y_5
3	4	y_6

T

B	Φ
1	z_1
2	z_2
3	z_3
4	z_4

$$Q_1 = \pi_{\emptyset}\big(R \bowtie_A S \bowtie_B T\big)$$
$$Q_2 = \pi_{\emptyset}\big(R \bowtie_{R.A\leq S.A} S \bowtie_B T\big)$$

$$
\begin{aligned}
\Phi_1 &= x_1y_1z_1 \vee x_1y_2z_2 \vee x_2y_3z_1 \vee x_2y_4z_2 \vee x_3y_5z_3 \vee x_3y_6z_4\\
\Phi_2 &= x_1y_1z_1 \vee x_1y_2z_2 \vee x_1y_3z_1 \vee x_1y_4z_2 \vee x_1y_5z_3 \vee x_1y_6z_4 \vee\\
&\quad\ x_2y_3z_1 \vee x_2y_4z_2 \vee x_2y_5z_3 \vee x_2y_6z_4 \vee x_3y_5z_3 \vee x_3y_6z_4
\end{aligned}
$$

Fig. 2 A pc-table with relations R, S, T annotated with variables from disjoint sets $V_1 = \{x_1, \ldots, x_3\}$, $V_2 = \{y_1, \ldots, y_6\}$, $V_3 = \{z_1, \ldots, z_4\}$, respectively. Formulas Φ_1 and Φ_2 are the annotations of the results of queries Q_1 and Q_2 evaluated on this database

to these queries are the sets consisting of the empty tuple whose probability is $P(\Phi_1)$ and $P(\Phi_2)$, respectively.

3 Model-based approximations

The problem investigated in this section is as follows: Given a formula Φ in a propositional language $\mathscr{L}$, find formulas in a (more restrictive) language $\mathscr{L}'$ that approximate Φ. We restrict our investigation to positive (or without loss of generality, unate) and irreducible formulas; the reason for this restriction is efficiency, since for arbitrary formulas, it is NP-hard to decide whether true ($\top$) or false ($\bot$) are bounds. This defeats our goal of efficient model-based approximation.

Before looking into approximating positive DNF formulas within the language of iDNF (Sects. 3.1 and 3.2) and of positive nested formulas within the language of 1OF (Sect. 3.3), we define the notion of optimal bounds.

Definition 1 (*Optimal lower and upper bounds*) Let $\mathscr{L}'$ and $\mathscr{L}$ be languages of propositional formulas such that $\mathscr{L}' \subseteq \mathscr{L}$. Given a formula $\Phi \in \mathscr{L}$, the formulas $\Phi_L, \Phi_U \in \mathscr{L}'$ are *lower and upper bounds for Φ with respect to $\mathscr{L}'$* if $\mathscr{M}(\Phi_L) \subseteq \mathscr{M}(\Phi)$ and $\mathscr{M}(\Phi) \subseteq \mathscr{M}(\Phi_U)$, respectively.

If in addition there are no formulas $\Phi'_L, \Phi'_U \in \mathscr{L}'$ such that $\mathscr{M}(\Phi_L) \subset \mathscr{M}(\Phi'_L) \subseteq \mathscr{M}(\Phi)$ and $\mathscr{M}(\Phi) \subseteq \mathscr{M}(\Phi'_U) \subset \mathscr{M}(\Phi_U)$, then Φ_L is a *greatest lower bound (GLB)* and Φ_U is a *least upper bound (LUB) for Φ with respect to $\mathscr{L}'$.*

The notions GLB and LUB provide an intuitive semantic characterization of *optimal* bounds. We consider languages $\mathscr{L}'$ such that (i) we can efficiently find optimal bounds in $\mathscr{L}'$, and (ii) $\mathscr{L}'$ allows for efficient probability computation. The 1OF and iDNF languages satisfy both desiderata.

Our interest in model-based approximation stems from the observation that, for formulas over independent random variables, model-based bounds imply probability bounds:

Proposition 1 *For any formulas Φ and Ψ with $\Phi \models \Psi$ and thus $\mathscr{M}(\Phi) \subseteq \mathscr{M}(\Psi)$, it holds that $P(\Phi) \leq P(\Psi)$.*

Before we discuss iDNF and 1OF approximations, we give a simple syntactic characterization of (not necessarily optimal) bounds for positive DNF formulas:

Proposition 2 *Let Φ and Ψ be positive DNF formulas. The following statements are equivalent:*

1. *$\Phi \models \Psi$;*
2. *Ψ can be obtained from Φ by adding clauses and removing literals from Φ's clauses;*
3. *Φ can be obtained from Ψ by removing clauses and adding literals to Ψ's clauses.*

According to Proposition 2, *all* bounds for a given positive DNF formula Φ can be defined by simple *syntactic* manipulations of Φ: Lower bounds by removing clauses or adding literals to existing clauses, and upper bounds by adding clauses or removing literals from existing clauses. Removing all clauses results in the lower bound $\bot$. Removing all literals from a clause results in the upper bound $\top$. Indeed, by removing clauses from a formula, we reduce its set of models. By adding variables to clauses, we further constrain their satisfiability and hence reduce the set of models. In both cases, one obtains formulas that are lower bounds. The inverse manipulations lead to upper bounds. More importantly, Proposition 2 states that all bounds can be gained in this way. The following sections show that it is not necessary to add literals to clauses in order to find *optimal* iDNF lower bounds, while removing variables may be required in order to get to optimal iDNF upper bounds.

Example 3 Neither formula Φ_1 nor Φ_2 from Fig. 2 are in iDNF. The following iDNF formulas are lower and upper bounds for Φ_1 (several others are possible):

$$\Phi_L = x_1y_1z_1 \vee x_2y_4z_2 \vee x_3y_5z_3$$
$$\Phi_U = z_1 \vee x_1y_2 \vee x_2y_4z_2 \vee x_3.$$

Indeed, Φ_L is a lower bound since it is a subset of Φ. Each clause in Φ implies at least one clause of Φ_U; hence, Φ_U is an upper bound. A closer inspection reveals that each clause in Φ_U can be obtained by dropping literals from clauses in Φ. We will show in the remainder of this section that the bounds Φ_L and Φ_U are indeed *optimal* for iDNF.

The following sections characterize syntactically the optimal iDNF bounds for positive formulas and give algorithms to find such bounds.

3.1 iDNF greatest lower bounds

Our main tool used to find optimal iDNF lower bounds is the syntactic notion of maximal lower bounds.

Definition 2 (*Maximal lower bound for iDNF*) Let Φ be a positive DNF formula. An iDNF formula Φ_L is a *maximal lower bound (MLB)* for Φ if it satisfies the conditions:

1. (*Lower bound*) Φ_L is a subset of Φ: $\Phi_L \subseteq \Phi$;
2. (*Maximality*) Φ_L cannot be further extended: There is no clause $\varphi \in \Phi$ with $\mathrm{vars}(\varphi) \cap \mathrm{vars}(\Phi_L) = \emptyset$.

The first condition ensures that Φ_L is indeed an iDNF lower bound. The second condition enforces that it is a maximal lower bound. An MLB for a formula Φ is thus a maximal set of pairwise independent clauses of Φ.

Example 4 The MLBs for formula Φ_1 from Fig. 2 are:

$$1 : x_1y_1z_1 \vee x_2y_4z_2 \vee x_3y_5z_3 \quad 2 : x_1y_1z_1 \vee x_2y_4z_2 \vee x_3y_6z_4$$
$$3 : x_1y_2z_2 \vee x_2y_3z_1 \vee x_3y_5z_3 \quad 4 : x_1y_2z_2 \vee x_2y_3z_1 \vee x_3y_6z_4$$

With respect to the iDNF language, maximal lower bounds correspond precisely to greatest lower bounds. This is particularly important, since MLB is a syntactic notion defined in terms of sets of clauses, and GLB is a semantic notion defined in terms of sets of models.

Theorem 1 *Let Φ be a positive DNF formula. An iDNF formula Φ_L is a maximal lower bound for Φ if and only if Φ_L is a greatest lower bound for Φ.*

The implication GLB ⇒ MLB holds since if any of the two MLB conditions fail, then we cannot obtain a GLB. For the other direction, it can be shown that no strict upper bound Φ'_L for Φ_L is an iDNF lower bound for Φ. The latter case has two subcases: There is a clause $\varphi \in \Phi$ that is in Φ'_L and either is or is not in Φ_L. In the first subcase, φ necessarily shares variables with one clause in $\varphi_L \in \Phi_L$, and, according to Lemma 1, it must be contained in φ_L. But then, φ must be φ_L; otherwise, Φ_L can be further extended and hence is no MLB. The second subcase follows similarly.

The syntactic definition of MLBs suggests an algorithm for enumerating all GLBs by recursively constructing all maximal subsets of pairwise independent clauses of Φ. This algorithm needs time exponential in the number of clauses. The following proposition shows that this is necessarily so:

Proposition 3 *The positive DNF formula*

$$\Phi = (x_1y_1 \vee x_1y_2) \vee \cdots \vee (x_ny_{2n-1} \vee x_ny_{2n})$$

has $2n$ clauses and 2^n iDNF GLBs. Any disjunction of either x_iy_{2i-1} or x_iy_{2i} for all $1 \leq i \leq n$ is an iDNF GLB of Φ.

Since there can be exponentially many GLBs that are by definition incomparable with respect to their model sets, it may be desirable to find "the best" GLB for a formula according to different criteria. Possible rankings of GLBs are by the number of clauses or, useful in our context, by their probabilities, provided they are defined over random variables. Additionally, it can be useful to have an enumeration of all GLBs with polynomial delay [31], which means that the time before finding the first GLB, as well as the time between every two consecutive GLBs, is polynomial only in the input size, and not in the number of GLBs. We obtain results for ranking and enumerating GLBs by exploiting the following correspondence between GLBs and independent sets in the clause-dependency graphs of formulas.

Definition 3 (*Clause-dependency graph*) Let Φ be a DNF formula. The *clause-dependency graph* of Φ is a graph $G = (\Phi, E)$, where nodes are clauses of Φ, and two nodes φ and ψ are connected if $\text{vars}(\varphi) \cap \text{vars}(\psi) \neq \emptyset$.

Each node $\varphi \in \Phi$ in G has a weight defined by

$$w(\varphi) = -\ln\bigl(1 - P(\varphi)\bigr) = -\ln\left(1 - \prod_{x \in \varphi} P_x\right). \tag{4}$$

The weight $W(\Psi)$ of a node set Ψ is $W(\Psi) = \sum_{\psi \in \Psi} w(\psi)$.

Lemma 2 *Let Φ be a positive DNF formula. A formula $\Phi_L \subseteq \Phi$ is an iDNF greatest lower bound for Φ if and only if the clauses of Φ_L represent a maximal independent set in the clause-dependency graph of Φ.*

As the following proposition shows, the correspondence of Lemma 2 can be extended to the case of greatest lower bounds with the largest probability. Such bounds correspond to maximal independent sets with the largest weight.

Proposition 4 *Let Φ be a positive DNF formula and $G = (\Phi, E)$ be its clause-dependency graph. A formula Φ_L is an iDNF greatest lower bound of Φ with the largest probability among all iDNF greatest lower bounds of Φ if and only if Φ_L is a maximum-weight independent set in G.*

By the correspondence between iDNF greatest lower bounds and maximal independent sets, known results for the latter, such as enumeration of maximal independent sets with polynomial delay, immediately carry over to the former.

Corollary 2 (Lemma 2, [55,31]) *The set of all iDNF greatest lower bounds for a positive DNF formula with n clauses can be enumerated with $O(n^3)$ delay.*

Since finding a maximum independent set is NP-hard, we cannot efficiently enumerate the maximal independent sets, or equivalently the iDNF GLBs, in decreasing order of their sizes, and, more importantly here, of their probabilities.

Corollary 3 (Lemma 2, [21]) *Enumerating iDNF greatest lower bounds for a given positive DNF formula Φ in decreasing order of their probabilities is NP-hard.*

Although we cannot efficiently obtain an iDNF GLB with largest probability, a constant-factor approximation can be obtained for positive DNF formulas with max-arity k. For such a formula Φ, Algorithm 1 constructs an iDNF greatest lower bound Φ_L by iterating over Φ's clauses in decreasing order of their probabilities and greedily selecting a maximal set of independent clauses: Φ_L thus contains the first clause φ_1 in the order, then the next clause independent of φ_1, and so on. The probability of this lower bound is at most a factor k smaller than the largest probability among the probabilities of all iDNF greatest lower bounds for Φ:

Algorithm 1: Finding an iDNF greatest lower bound with probability at most factor k smaller than the largest probability of iDNF greatest lower bounds.

```
GreedyIDnfGlb(Positive DNF $\Phi$ with max-arity $k$)
    ▷ outputs an iDNF GLB of $\Phi$
    Sort clauses of $\Phi$ in descending order $\varphi_1, \ldots, \varphi_n$ of their
    probabilities $P(\varphi_1) \geq P(\varphi_2) \geq \cdots \geq P(\varphi_n)$.
    $\Phi_L \leftarrow \emptyset$
    for $i = 1 \ldots n$ do
        if $vars(\varphi_i) \cap vars(\Phi_L) = \emptyset$ then
            $\Phi_L \leftarrow \Phi_L \cup \{\varphi_i\}$
output $\Phi_L$
```

Proposition 5 *Given a positive DNF formula Φ with max-arity k and n clauses, Algorithm 1 constructs an iDNF greatest lower bound Φ_L for Φ in time $O(n^2)$ such that $P(\Phi'_L) \leq k \cdot P(\Phi_L)$ for every iDNF greatest lower bound Φ'_L for Φ.*

3.2 iDNF least upper bounds

As for iDNF greatest lower bounds, we next give a syntactic characterization of iDNF least upper bounds for positive DNF formulas. We first introduce a few necessary notions.

Definition 4 (*Witness and critical witness*) Let Φ and Ψ be formulas. A clause $\varphi \in \Phi$ is a *witness* for a clause $\psi \in \Psi$ if $\varphi \models \psi$. We also say that ψ *has the witness* φ. If in addition, there is no clause $\psi' \in \Psi$ with $\psi \neq \psi'$ and $\varphi \models \psi'$, then φ *is a critical witness* for ψ, and ψ *has the critical witness* φ.

Example 5 Consider formulas Φ_1 in Fig. 2 and $\Psi = x_1y_1 \vee x_2 \vee z_2 \vee x_1y_6$. Clause $x_1y_1 \in \Psi$ has the critical witness $x_1y_1z_1 \in \Phi_1$, clause $x_2y_4z_2$ is a non-critical witness for x_2 and $z_2 \in \Psi$, and clause $x_1y_6 \in \Psi$ has no witness in Φ_1.

We are now ready to present our main syntactic tool for finding iDNF optimal upper bounds.

Definition 5 (*Minimal upper bound for iDNF*) Let Φ be a positive DNF formula. An iDNF formula Φ_U is a *minimal upper bound (MUB)* for Φ if it satisfies the conditions:

1. (*Upper bound*) Every clause $\varphi \in \Phi$ is a witness for at least one clause in Φ_U;
2. (*Maximality*) There is no clause $\varphi_u \in \Phi_U$ that can be extended by a variable from vars(Φ) while preserving condition (1) and keeping Φ_U in iDNF;
3. (*Criticality*) Every clause $\varphi_u \in \Phi_U$ has at least one critical witness in Φ.

Following Lemma 1, the first condition ensures that $\Phi \models \Phi_U$. The second condition ensures that we cannot obtain an iDNF refinement of Φ_U which is still an upper bound for Φ by making clauses in Φ_U more specific (thus narrowing the set of models of Φ_U). The third condition states that all clauses in Φ_U are necessary, and dropping any of them would lead to clauses in Φ violating Lemma 1.

This syntactic characterization of upper bounds precisely matches the semantic notion of least upper bounds:

Theorem 2 *Let Φ be a positive DNF formula. An iDNF formula Φ_U is a minimal upper bound for Φ if and only if Φ_U is a least upper bound for Φ.*

The implication LUB ⇒ MUB holds since, if any of the three MUB conditions fail, we cannot obtain an LUB. For the other direction, any difference between an MUB Φ_U for Φ and a potentially better upper bound Φ'_U requires a clause $\varphi'_u \in \Phi'_U$ strictly containing (syntactically) a clause $\varphi \in \Phi_U$. A case analysis on a variable $x \in \varphi'_u$ and $x \notin \varphi_u$ shows that Φ_U does not satisfy the syntactic characterization of an MUB: Either the criticality condition fails or a clause in Φ_U may be expanded by x to obtain a smaller bound.

Example 6 The iDNF formula $\Phi^1_{1,u} = x_1y_1 \vee y_2 \vee x_2y_3z_1 \vee y_4z_2 \vee x_3y_5z_3 \vee y_6z_4$ is an MUB for Φ_1 from Fig. 2. Every clause in $\Phi^1_{1,u}$ has exactly one witness in Φ_1 which is also critical. No clause can be extended further, since vars$(\Phi^1_{1,u}) =$ vars(Φ_1). The iDNF formula $\Phi^2_{1,u} = x_1 \vee x_2 \vee x_3y_5z_3 \vee y_6z_4$ is also an MUB; each of the two clauses x_1 and x_2 has two critical witnesses. It can be checked that $\Phi^2_{1,u}$ cannot be extended by any variable from vars(Φ_1).

As in the case of lower bounds, the number of least upper bounds for a given formula can be exponential in its size:

Proposition 6 *The positive DNF formula*

$$\Phi = (x_1y_1 \vee x_1y_2) \vee \cdots \vee (x_ny_{2n-1} \vee x_ny_{2n})$$

has $2n$ clauses and 3^n iDNF LUBs. Any disjunction of either x_i, or $x_iy_{2i-1} \vee y_{2i}$, or $y_{2i-1} \vee x_iy_{2i}$ for all $1 \leq i \leq n$ is an iDNF LUB of Φ.

Algorithm 2 enumerates iDNF least upper bounds with polynomial delay using a simple strategy that relies on the compositional nature of minimal upper bounds: An MUB of an irreducible positive DNF formula $\Phi = \psi \vee \Phi'$, where ψ is a clause and Φ' is a formula, is the disjunction of the MUB of ψ, which is ψ itself, and of the MUB of Φ', where we remove from Φ' the variables in ψ and subsumed clauses. Multiple MUBs can then be obtained by picking different clauses ψ that share a variable. This condition is necessary to guarantee distinct MUBs; indeed, these clauses are incomparable and each of them will be in an MUB. Note how empty clauses are handled: If the irreducible input formula is $\{\emptyset\}$, i.e., true, then PolyMub returns its only MUB $\{\emptyset\}$. In each recursive call, Φ cannot contain the empty clause, because removing from the irreducible formula Φ_R the variables from one of its clauses ψ cannot yield an empty clause.

Algorithm 2: Enumerating iDNF least upper bounds with polynomial delay. Initial call: PolyMub(Φ, $\emptyset$).

```
PolyMub(Positive DNF $\Phi$, iDNF $\Phi_U$)
    ▷ outputs iDNF MUBs of $\Phi$ with polynomial delay
    if $\Phi = \{\emptyset\}$ then
        ▷ $\Phi$ is true and has the trivial MUB $\{\emptyset\}$
        output $\Phi$
    else if $\Phi = \emptyset$ then
        output $\Phi_U$
    else
        ▷ Remove redundant clauses from $\Phi$
        ▷ Duplicates are automatically discarded since $\Phi_R$ is a set
        $\Phi_R \leftarrow \{\varphi \mid \varphi \in \Phi \wedge \nexists \psi \in \Phi : \psi \subset \varphi\}$

        $x \leftarrow$ Pick variable from $\Phi_R$
        foreach $\psi \in \Phi_R$ with $x \in \psi$ do
            ▷ Add clause $\psi$ to $\Phi_U$ and recursively find MUBs of the
              formula obtained from $\Phi_R$ by removing $\psi$ and variables
              in vars($\psi$) from all other clauses
            PolyMub$\big(\{\varphi \setminus \mathrm{vars}(\psi) \mid \varphi \in \Phi_R \wedge \varphi \neq \psi\}, \Phi_U \cup \{\psi\}\big)$
```

Theorem 3 *Algorithm 2 enumerates iDNF least upper bounds of a positive DNF formula with n clauses with $O(n^3)$ delay.*

Algorithm 2 is correct, i.e., every returned formula is an MUB and thus an LUB, since properties 1–3 in Definition 5 inductively carry over from the recursively generated MUB to the formula obtained by adding clause ψ. The returned MUBs are unique and thus incomparable, since they differ in the clause that contains variable x picked before the for-loop. The delay between consecutive MUBs is cubic in the number n of clauses in the input formula Φ, because the recursion depth of PolyMub is bounded by n, and each recursion step can be performed in time quadractic in n.

Algorithm 2 is not complete: For instance, it does not find MUBs $a \vee b$ and $x \vee y$ for the formula $ax \vee bx \vee ay \vee by$, because it cannot factorize it into $(a \vee b) \wedge (x \vee y)$. A complete enumeration algorithm for MUBs is given in the electronic supplementary material, albeit without polynomial delay guarantees.

3.3 1OF lower and upper bounds

We now turn to approximation of nested formulas. Such formulas naturally annotate results of queries with negation in probabilistic databases, as per their construction using the translation in Fig. 1. While their size is polynomial in the size of the input database, un-nesting them in DNF can lead to formulas of size exponential in the size of the input database. We therefore search for approximations that are nested as well and that can be computed without unfolding the input in DNF. We recall our restriction to positive (or unate) formulas for efficiency reasons. This restriction corresponds here to queries where all occurrences of each relation are either positive or negative.

The iDNF bounds are, by definition, in DNF and thus do not meet our requirements here. However, the 1OF language naturally allows for nested formulas. Consider for instance the unate formula $\Phi = (\bigvee_i x_i) \wedge \bigwedge_j (\neg y_j)$. Then, any two clauses in the DNF of Φ would share variables, which means that we can only use one of these clauses as iDNF lower bound. The formula Φ is however in 1OF.

As in the iDNF case, Proposition 2 gives us a useful tool to find 1OF bounds. A practical caveat is that it talks about clauses in the DNF representation of a formula, whereas we deal here with nested formulas. We therefore consider an efficient heuristic for computing (a) lower bounds by dropping clauses, in particular by setting one of their literals to false, and (b) upper bounds by dropping literals from clauses, in particular by setting these literals to true. The following proposition formalizes this intuition.

Proposition 7 *Let Φ be a positive formula, x be a variable in Φ, and Φ_L and Φ_U be formulas obtained from Φ by replacing one occurrence of x by $\bot$ and, respectively, $\top$. Then, Φ_L and Φ_U are lower and, respectively, upper bounds of Φ.*

We can then obtain lower and upper *1OF bounds* of a positive (or equivalently, unate) formula by setting all but one occurrence of each distinct variable to false and, respectively, to true. Depending on which occurrences are chosen, we may obtain different 1OF bounds.

Example 7 The formula $\Phi = (x_1 y_1 \vee \neg x_2 y_2)[x_1(y_1 \vee y_3) \vee x_3(y_4 \vee y_5)]$ is not in 1OF since x_1 and y_1 occur twice. Setting the second occurrences of both these variables to false results in the 1OF lower bound $(x_1 y_1 \vee \neg x_2 y_2) x_3 (y_4 \vee y_5)$. For an 1OF upper bound, we set the same occurrences of the two variables to true and obtain $x_1 y_1 \vee \neg x_2 y_2$.

Different bounds can be obtained by setting different occurrences of the two variables to false and true, respectively. For example, setting the first occurrences of x_1 and y_1 results in the lower bound $(\neg x_2 y_2)[x_1(y_1 \vee y_3) \vee x_3(y_4 \vee y_5)]$ and the upper bound $x_1(y_1 \vee y_3) \vee x_3(y_4 \vee y_5)$.

4 Probability computation via decomposition

This section introduces our probability computation algorithm for propositional formulas. We first give in Sect. 4.1 an exact version and then lift it to a memory-efficient anytime approximation algorithm in Sect. 4.2. Considerations for efficient implementation are given in Sect. 4.2.4.

4.1 Exact probability computation

The key idea behind our algorithm is that the probability of formulas $\Phi \wedge \Psi$ and $\Phi \vee \Psi$ can be efficiently computed from the probabilities $P(\Phi)$ and $P(\Psi)$ of the subformulas Φ and Ψ, if these subformulas are independent or inconsistent:

- if Φ and Ψ are independent, then

$$P(\Phi \wedge \Psi) = P(\Phi) \cdot P(\Psi)$$
$$P(\Phi \vee \Psi) = 1 - (1 - P(\Phi)) \cdot (1 - P(\Psi))$$

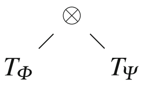
- if Φ and Ψ are inconsistent (i.e., there is no valuation of the random variables for which both are true: the disjunction is *exclusive*), then

$$P(\Phi \wedge \Psi) = 0$$
$$P(\Phi \vee \Psi) = P(\Phi) + P(\Psi).$$

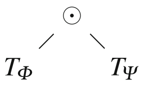
The above equations can be derived from Eq. (3). We use explicit notation to denote independence and mutual exclusiveness: $\otimes$ for independent-or, $\odot$ for independent-and and $\oplus$ for exclusive-or. A *decomposition tree* is an alternative representation of formulas that makes independent and exclusive relationships between subformulas explicit.

Definition 6 (*Decomposition Tree*) A decomposition tree (d-tree) is recursively defined as follows:

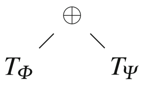
- Every propositional formula Φ is a d-tree T with a single node representing Φ.
- If T_Φ and T_Ψ are d-trees representing independent formulas Φ and Ψ, then



and



are d-trees representing $\Phi \vee \Psi$ and $\Phi \wedge \Psi$, respectively.
- If T_Φ and T_Ψ are d-trees representing mutually exclusive formulas Φ and Ψ, then



is a d-tree representing the formula $\Phi \vee \Psi$.

A d-tree in which all leaves are literals is *complete*.

A d-tree can also be seen as the parse tree of a formula composed of propositional formulas (with $\vee$, $\wedge$, $\neg$) and independent and mutual exclusive operators $\otimes$, $\odot$, $\oplus$.

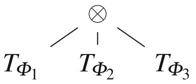
Example 8 Consider the formula $(x \vee y) \wedge ((z \wedge u) \vee (\neg z \wedge v))$. It is easy to verify that this formula satisfies the independence and mutual exclusiveness properties expressed by the equivalent partial d-tree $(x \otimes y) \odot (zu \oplus \neg zv)$ and the complete d-tree $(x \otimes y) \odot \big((z \odot u) \oplus (\neg z \odot v)\big)$.

Definition 7 (*Probability of D-tree*) The probability of a d-tree T is recursively defined on its structure:

- If T is a leaf node for formula Φ, then $P(T) = P_\Phi$
- If $T = T_\Phi \otimes T_\Psi$, then $P(T) = 1-(1-P(T_\Phi))(1-P(T_\Psi))$
- If $T = T_\Phi \odot T_\Psi$, then $P(T) = P(T_\Phi) \cdot P(T_\Psi)$
- If $T = T_\Phi \oplus T_\Psi$, then $P(T) = P(T_\Phi) + P(T_\Psi)$

Example 9 The probability of formula $(x \vee y) \wedge ((z \wedge u) \vee (\neg z \wedge v))$ and its d-tree $(x \otimes y) \odot (z \odot u \oplus \neg z \odot v)$ is $\big(1 - (1 - P_x) \cdot (1 - P_y)\big) \cdot \big(P_z \cdot P_u + P_{\neg z} \cdot P_v\big)$.

Since the operations $\otimes$, $\odot$ are associative, the independent decompositions readily extend to decompositions into more than two independent formulas; for example, $\Phi_1 \vee \Phi_2 \vee \Phi_3$ (with Φ_1, Φ_2, Φ_3 independent) can be decomposed into



It follows from the definitions of $\oplus$, $\otimes$, and $\odot$ that:

Proposition 8 *Given the probabilities of all the leaves of a d-tree, its probability can be computed in time linear in its size (assuming unit-cost arithmetic operations).*

Any formula can be compiled into a d-tree by finding independent subformulas and decomposing with respect to $\otimes$ and $\odot$, and otherwise by applying Shannon expansion to yield a mutually exclusive $\oplus$-decomposition. Algorithm 3 sketches this generic compilation approach, where the compilation is exhaustive, i.e., the leaves of the d-tree represent literals. This is refined in Sect. 4.2, where we describe how a partial compilation of d-trees gives rise to approximate probability computation. The notations $\Phi|_{x \leftarrow \top}$ and $\Phi|_{x \leftarrow \bot}$ denote formulas obtained by replacing every occurrence of variable x in Φ by $\top$ and, respectively, $\bot$.

Algorithm 3: Compilation of formulas into d-trees.

```
COMPILE (Formula $\Phi$)
    RemoveRedundantClauses ($\Phi$)
    if $\Phi$ is a literal or $\top$ or $\bot$ then
        return $\Phi$
    if $\exists$ independent $\Phi_1, \Phi_2$ s.t. $\Phi_1 \vee \Phi_2 = \Phi$ then
        return COMPILE($\Phi_1$) $\otimes$ COMPILE($\Phi_2$)
    if $\exists$ independent $\Phi_1, \Phi_2$ s.t. $\Phi_1 \wedge \Phi_2 = \Phi$ then
        return COMPILE($\Phi_1$) $\odot$ COMPILE($\Phi_2$)
    $\triangleright$ Apply Shannon expansion:
    Choose variable $x \in \mathbf{X}$ occurring in $\Phi$
    return $\big(x \odot$ COMPILE($\Phi|_{x \leftarrow \top}$)$\big) \oplus \big(\neg x \odot$ COMPILE($\Phi|_{x \leftarrow \bot}$)$\big)$
```

Example 10 Figure 3 shows a formula and its complete d-tree obtained by executing Algorithm 3 to completion.

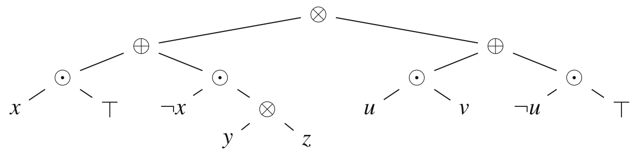


Fig. 3 D-tree for $\Phi = x \vee \neg xy \vee \neg xz \vee uv \vee \neg u$

The correctness of Algorithm 3 follows directly from the recursive definition of the represented formula and the probability of a d-tree (Definitions 6 and 7):

Proposition 9 *Let Φ be a formula and T be the d-tree returned by* Compile(Φ)*. Then, the formula represented by T is equivalent to Φ and $P_\Phi = P(T)$.*

Any formula can be decomposed into a complete d-tree, since Shannon expansion (a.k.a. variable elimination in the Davis–Putnam algorithm for SAT solving [14]) can always be applied and leads to formulas with less variables. We can compute the probability of the input formula by inlining the equations from Definition 7 in Algorithm 3.

The order of the variable choices in Shannon expansion influences the size of the d-tree. In general, the compilation of a formula creates a d-tree of exponential size, and it is important to find compilation strategies that lead to d-trees of small sizes [13,35]. For tractable queries, it is possible to efficiently identify optimal variable orders [39–41]. For non-tractable queries, we choose a variable that occurs most frequently in the formula.

We next analyze the complexity of Algorithm 3. All three decompositions steps can be done efficiently. Shannon expansion requires linear time for each subformula. The independent-or and independent-and partitioning finds connected components in the dependency graph of the input formula Φ. Given $\Phi = \bigwedge_i \Phi_i$ or $\Phi = \bigvee_i \Phi_i$, the dependency graph of Φ has one node for each subformula Φ_i, and there is an edge between two nodes Φ_i and Φ_j if $\text{vars}(\Phi_i) \cap \text{vars}(\Phi_j) \neq \emptyset$. A more effective independent-and partitioning can be achieved if Φ is in DNF, since we can apply existing algebraic factorisation of DNF formulas [6]. For a relational encoding of DNF formulas of arity k and n clauses, as used in query evaluation on probabilistic databases [3], the $\otimes$-decomposition is unique and requires time $O(k \cdot n \cdot \log n)$ [42].

We close this section with some observations regarding tractable classes of conjunctive queries without self-joins and their connection to linear-size d-trees [38–41].

Firstly, the annotations of the results of any tractable conjunctive query without self-joins on a tuple-independent database are positive DNF formulas equivalent to formulas in 1OF and can be compiled in polynomial time into complete d-trees of size linear in the number of variables and whose inner nodes are $\otimes$ and $\odot$ only [38].

Secondly, the annotations of the results of any query in a previously defined class of tractable conjunctive queries with inequalities on tuple-independent databases are positive DNF formulas that can be compiled in polynomial time into complete d-trees of size linear in the size of the formulas and whose inner nodes are $\oplus$ only [39].

4.2 Approximate probability computation

Let us now turn to approximate probability computation. We start by discussing the correspondence between d-trees and probability bounds for represented formulas, then define our approximation algorithm (Sect. 4.2.2) and refine it to a memory-efficient implementation (Sect. 4.2.3).

4.2.1 Lower and upper probability bounds for D-trees

Algorithm 3 and Definition 7 give us a recipe for exact probability computation for any formula, provided we exhaustively decompose it into a d-tree where the leaves are literals with known exact probabilities. By using partial instead of complete decompositions, and thus by allowing arbitrary formulas at leaves, the same approach yields lower and upper probability bounds for partial d-trees.

We next assume that for each leaf node Φ, we know probabilities P^L_Φ and P^U_Φ such that $P_\Phi \in [P^L_\Phi, P^U_\Phi]$.

Definition 8 (*Probability Bounds of D-trees*) The lower and upper probability bounds $P^L(T)$ and $P^U(T)$ of a d-tree T are recursively defined by its structure:

- If T is a leaf node for formula Φ, then
$$P^L(T) = P^L_\Phi, \qquad P^U(T) = P^U_\Phi.$$
- If $T = T_\Phi \otimes T_\Psi$, then
$$P^L(T) = 1 - (1 - P^L(T_\Phi))(1 - P^L(T_\Psi))$$
$$P^U(T) = 1 - (1 - P^U(T_\Phi))(1 - P^U(T_\Psi))$$
- If $T = T_\Phi \odot T_\Psi$, then
$$P^L(T) = P^L(T_\Phi) \cdot P^L(T_\Psi)$$
$$P^U(T) = P^U(T_\Phi) \cdot P^U(T_\Psi)$$
- If $T = T_\Phi \oplus T_\Psi$, then
$$P^L(T) = P^L(T_\Phi) + P^L(T_\Psi)$$
$$P^U(T) = P^U(T_\Phi) + P^U(T_\Psi)$$

Intuitively, the definition implies that the lower (upper) probability bound of a d-tree is obtained by propagating up

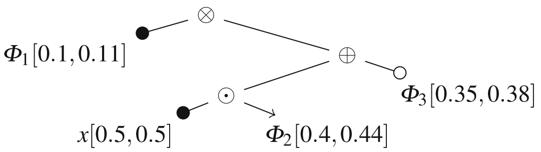


Fig. 4 Partially decomposed d-tree. Leaves: Φ_1 and x are closed (●), Φ_2 is current (→), Φ_3 is open (○)

the lower (upper) probability bounds of the leaf formulas according to Definition 7. Since the operators in this definition are all monotonically increasing as a function of the probability of their child nodes, we obtain

Proposition 10 *For any d-tree T, $P(T) \in [P^L(T),\ P^U(T)]$.*

Example 11 Consider the partial d-tree $T = \Phi_1 \otimes ((x \odot \Phi_2) \oplus \Phi_3)$ of Fig. 4, where the leaves are annotated with their lower and upper bounds. Then, the lower and upper bounds $[P^L(T), P^U(T)]$ of the d-tree can be computed as follows:

$$P^U(T) = 1-(1-0.11)\cdot(1-(0.5\cdot 0.44+0.38)) = 0.644$$
$$P^L(T) = 1-(1-0.1)\cdot(1-(0.5\cdot 0.4+0.35)) = 0.595$$

An immediate question is how good the bounds of a partial d-tree are and whether one can make guarantees about the approximation quality. We consider two types of such guarantees, given a fixed error threshold $\epsilon \in [0, 1]$:

Definition 9 A value $\hat{p}$ is an *absolute* (or *additive*) ϵ-approximation of a probability p if $p - \epsilon \leq \hat{p} \leq p + \epsilon$.

A value $\hat{p}$ is a *relative* (or *multiplicative*) ϵ-approximation of a probability p if $(1-\epsilon)\cdot p \leq \hat{p} \leq (1+\epsilon)\cdot p$.

We will abbreviate $P^L(T)$ and $P^U(T)$ with L and U if the context is clear. Let us now investigate under which conditions the bounds $[L, U]$ of a given d-tree for a formula Φ are sufficiently tight to specify an ϵ-approximation of P_Φ. The connection between bounds of d-trees and ϵ-approximations of formulas is given by the following proposition.

Proposition 11 *Let ϵ be a fixed error threshold, and Φ a formula with probability bounds $[L, U]$, i.e., $P_\Phi \in [L, U]$.*

- *If $U - \epsilon \leq L + \epsilon$, then any value in $[U - \epsilon, L + \epsilon]$ is an absolute ϵ-approximation of P_Φ.*
- *If $(1-\epsilon)\cdot U \leq (1+\epsilon)\cdot L$, then any value in $[(1-\epsilon)\cdot U, (1+\epsilon)\cdot L]$ is a relative ϵ-approximation of P_Φ.*

A d-tree for a formula Φ is an ϵ-approximation of Φ if its bounds satisfy one of the above sufficient conditions. Both conditions can be checked in linear time in the size of the d-tree, because its lower and upper bounds can be computed in one bottom-up pass, cf. Proposition 8.

Algorithm 4: Approximate probability computation with relative or absolute error guarantee for a partial d-tree T.

```
APPROX(D-TREE $T$, ERROR GUARANTEE $\epsilon$, ERROR TYPE TYPE)
    $L \leftarrow P^L(T)$
    $U \leftarrow P^U(T)$
    if TYPE = absolute and $U - L \leq 2\epsilon$ then
        return $[U - \epsilon, L + \epsilon]$
    if TYPE = relative and $(1-\epsilon)\cdot U - (1+\epsilon)\cdot L \leq 0$ then
        return $[(1-\epsilon)\cdot U, (1+\epsilon)\cdot L]$

    ▷ Approximation not reached; choose and refine a leaf in $T$
    Choose a leaf $t$ of $T$ representing a formula $\Phi$
    RemoveRedundantClauses ($\Phi$)
    if $\Phi$ is a literal or $\top$ or $\bot$ then
        ▷ Do nothing
    else if $\exists$ independent $\Phi_1, \Phi_2$ s.t. $\Phi_1 \vee \Phi_2 = \Phi$ then
        Replace $t$ in $T$ by $\Phi_1 \otimes \Phi_2$
    else if $\exists$ independent $\Phi_1, \Phi_2$ s.t. $\Phi_1 \wedge \Phi_2 = \Phi$ then
        Replace $t$ in $T$ by $\Phi_1 \odot \Phi_2$
    else
        Choose variable $x \in \mathbf{X}$ occurring in $\Phi$
        Replace $t$ in $T$ by $(x \odot \Phi|_{x\leftarrow\top}) \oplus (\neg x \odot \Phi|_{x\leftarrow\bot})$
    return APPROX($T$, $\epsilon$, TYPE)
```

4.2.2 An anytime approximation algorithm

We next present two algorithms for computing ϵ-approximations of formula probabilities via d-trees. The algorithm detailed in this section resembles Best-first Search in that it incrementally builds a d-tree from a formula by decomposing one of its leaves until the desired approximation is reached. Since the d-tree can be exponential in the number of variables in worst-case, the next section presents a refined version that mimics depth-first search and only keeps in memory one root-to-leaf path in the d-tree at any one time.

Algorithm 4 specifies our incremental refinement procedure for approximate probability computation. Starting with the d-tree $T = \Phi$ for a formula Φ, it checks whether T is already an ϵ-approximation for Φ. For this, it computes lower and upper bounds for Φ and its probability. If the desired approximation is reached, it returns the probability interval according to Proposition 11. Otherwise, it picks a leaf of the d-tree, e.g., a leaf with widest probability interval among all leaves, it performs one decomposition step, and it recurses.

The critical ingredient to this algorithm is the availability of good bounds for formulas at leaves. Given a positive formula Φ, we employ the techniques for model-based formula approximation (cf. Sect. 3) to approximate Φ and hence P_Φ by (i) optimal iDNF bounds if Φ is a DNF formula, and (ii) 1OF bounds if Φ is a nested formula.

In case a leaf represents a non-unate formula, we do not have access to efficiently computable model-based bounds, since deciding their existence is already a hard problem:

Proposition 12 *Let Φ be a formula. Deciding whether there exist non-trivial iDNF formulas $L \neq \bot$ and $U \neq \top$ such that $\mathscr{M}(L) \subseteq \mathscr{M}(\Phi) \subseteq \mathscr{M}(U)$ is NP-hard.*

This follows from hardness of satisfiability and tautology for propositional formulas. We thus cannot even decide efficiently whether the probability of a non-unate formula is 0 or 1. By default, we approach this obstacle by first repeatedly applying Shannon expansion until all leaves represent unate formulas. Once the leaves represent unate formulas, we can compute model-based bounds efficiently as described in Sect. 3.3.

In case of non-unate DNF formulas, such as those annotating results of positive queries on block-independent disjoint tables, we consider a different approach. We compute iDNF lower bounds using Algorithm 1. Such bounds are however not optimal in the iDNF language; for instance, the non-unate DNF formula $xy \vee x\neg y$ is equivalent to x, which represents the iDNF optimal lower and upper bounds, yet Algorithm 1 would output one of the clauses as iDNF lower bound. As upper bound, we compute a probability bound U instead of a model-based bound, as the sum of probabilities of the lower bound returned by Algorithm 1 and of all other clauses in the input DNF formula. If U is larger than 1, we set it to the trivial bound 1.

We close this section with two remarks. Firstly, in order to compute the new bounds $[P^L(T), P^U(T)]$ after a decomposition step, it is not necessary to recompute the probability bounds of the entire d-tree. If we memorize the local probability bounds at each node of T, then it suffices to propagate the new bounds of the decomposed leaf up the d-tree.

Secondly, it is not immediately clear that the overall probability bound of a d-tree improves between two subsequent refinement steps. Consider a leaf node with memorized probability bounds $[L, U]$. The decomposition of this leaf node may result in new probability bounds $[L', U']$ that are no improvement over $[L, U]$, i.e., $[L', U'] \not\subseteq [L, U]$. However, since both bounds are correct, we may memorize and propagate their intersection $[L, U] \cap [L', U'] \subseteq [L, U]$ and thereby obtain an *anytime* algorithm by guaranteeing that the bounds never get worse between subsequent refinement steps.

4.2.3 A memory-efficient variant

Algorithm 4 may construct an exponentially large d-tree before reaching the desired approximation. This memory usage is not feasible for large input. We next present a variant that only keeps in memory one root-to-leaf path of the d-tree at any one time. For exact computation, we accumulate the probability of the d-tree while exploring it depth-first. The explored branches can be safely discarded since their probabilities were already accumulated. The same idea can be used to devise an approximation approach as follows. We also explore the complete d-tree depth-first, but stop as soon as the desired approximation is reached, cf. Proposition 11. For this, we need to maintain a pair of lower and upper probability bounds for the not-yet-explored children of each node along the current root-to-leaf path. While this approximation approach performs better than the exact algorithm since it may avoid the complete exploration of the d-tree, it has one important practical shortcoming: It still needs to explore deep fragments of the d-tree, yet the deeper a branch goes, the more work it requires and the less probability mass it contributes to the probability of the d-tree.

Similarly to Algorithm 4, we would like to traverse a shallow upper part of the d-tree. In contrast to Algorithm 4, we would like to traverse the tree depth-first and only keep one path in memory at any one time. This requires a criterion to stop the depth-first exploration when the probability mass represented by the sub-tree of a node is insignificant and can be discarded under the premise that subsequent exploration of the remaining tree can still yield the desired approximation. We next detail this approach; it represents the approximation algorithm implemented in the SPROUT engine.

In the sequel, we call leaves that may be further refined *open* and leaves that are not further refined *closed*. We need to address the challenge of deriving an ϵ-approximation condition in the presence of closed leaves. Based on this, we can incrementally compile the input formula into a d-tree in depth-first traversal and decide *locally*, whether the current leaf under exploration can be closed or must be refined further. When a leaf gets closed, its bounds are accumulated and it can be removed from the d-tree.

For reasons explained later, we restrict our discussion to d-trees in which at most one child of each $\odot$ node can be closed without computing its exact probability. This constraint does not restrict our encoding of Shannon expansion using $\odot$ nodes, since one of their two children is always a literal for which the exact probability is known.

To understand the worst-case scenario when we want to close a leaf t in a d-tree T, we need to compute the largest bounds interval of T for any possible exact probability that each open leaf may take. If these worst-case bounds fail to satisfy the condition for an ϵ-approximation, then we might not reach the approximation by only refining the open leaves and must thus not close t. Let us formalize this intuition.

Definition 10 The *bounds space* of a d-tree T is the set of possible bounds $[L, U]$ of T obtained by choosing for each open leaf any point interval between the bounds of that leaf and for each closed leaf Ψ its probability interval $[P^L_\Psi, P^U_\Psi]$.

Let us denote by $L(T)$ the element of the bounds space obtained by choosing for each open leaf Φ the point interval $[P^L_\Phi, P^L_\Phi]$, where P^L_Φ is a lower bound for Φ.

Lemma 3 *For a d-tree T, $L(T)$ is the pair of bounds $[L, U]$ that maximizes $U - L$ for absolute approximation and*

$(1-\epsilon) \cdot U - (1+\epsilon) \cdot L$ for relative approximation, over the entire bounds space of T.

Lemma 3 gives us a strategy to decide whether closing leaves in a d-tree still allows for an ϵ-approximation. This only holds for closing $\otimes$ and $\oplus$ nodes; for $\odot$ nodes, $L(T)$ does not necessarily maximize the bounds in general. This explains our above restriction for $\odot$ nodes.

Since we can compute $L(T)$ in just one scan of T, finding the maximal values of $U - L$ and $(1-\epsilon) \cdot U - (1+\epsilon) \cdot L$ can be done very efficiently. Our main result concerning the closing of leaves follows from Lemma 3 and the fact that refinement eventually leads to completion of T.

Theorem 4 *Let T be a d-tree for a formula Φ, and ϵ be a fixed error. If the bounds $L(T)$ satisfy the sufficient condition for an ϵ-approximation in Proposition 11, then there is a refinement of T that is an ϵ-approximation of P_Φ.*

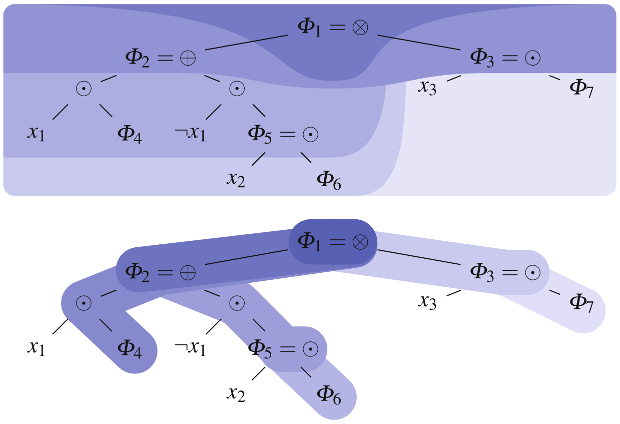
Example 12 Consider the d-tree T of Fig. 4 and an absolute error $\epsilon = 0.012$. We are at Φ_2 and would like to know (1) whether we can stop with an absolute ϵ-approximation, and in the negative case, (2) whether we can close Φ_2.

(1) We compute the lower and upper bounds of the d-tree as if all the leaves are closed. We plug in the lower bounds of the leaves and obtain $L = 0.1 \otimes ((0.5 \odot 0.4) \otimes 0.35) = 0.595$. Similarly for the upper bound: $U = 0.11 \otimes ((0.5 \odot 0.44) \otimes 0.38) = 0.644$. The condition $U - L = 0.049 \leq 2 \cdot 0.012 = 0.024$ is not satisfied. Hence, we cannot stop now.

(2) We compute $L(T) = [L, U']$, where L is as before and $U' = 0.11 \otimes ((0.5 \odot 0.44) \otimes 0.35) = 0.6173$. We then have that $U' - L = 0.0223 \leq 0.024$. We may thus close this leaf.

Our incremental algorithm follows the depth-first compilation scheme of Algorithm 3 where instead of constructing the d-tree, node probabilities are computed and propagated recursively. Before constructing a node, we perform two checks: (1) The sufficient condition of Proposition 11 that checks whether ϵ-approximation is already reached, and if this fails, then (2) the condition of Theorem 4 that decides whether the current node to be constructed can be safely closed. This algorithm closes d-tree branches eagerly in order to keep the d-tree height small, while still guaranteeing that the required ϵ-approximation can be reached.

Example 13 We would like to find relative and absolute 0.1-approximations for Φ_1 from Fig. 2 using our incremental decomposition and iDNF model-based bounds that are factor-k lower bounds from Proposition 5 and upper bounds according to Algorithm 2. For this, we incrementally construct the d-tree in Fig. 5 as discussed below.



$$\Phi_1 = x_1y_1z_1 \vee x_1y_2z_2 \vee x_2y_3z_1 \vee x_2y_4z_2 \vee x_3y_5z_3 \vee x_3y_6z_4 = \Phi_2 \otimes \Phi_3$$
$$L_1 = x_1y_1z_1 \vee x_2y_4z_2 \vee x_3y_5z_3$$
$$U_1 = x_1y_1z_1 \vee y_2z_2 \vee x_2y_3 \vee y_4 \vee y_5z_3 \vee x_3y_6z_4$$
$$\Phi_2 = x_1y_1z_1 \vee x_1y_2z_2 \vee x_2y_3z_1 \vee x_2y_4z_2 = (x_1 \odot \Phi_4) \oplus (\neg x_1 \odot \Phi_5)$$
$$L_2 = x_1y_1z_1 \vee x_2y_4z_2 \quad \text{and} \quad U_2 = x_1y_1z_1 \vee y_2z_2 \vee x_2y_3 \vee y_4$$
$$\Phi_3 = x_3y_5z_3 \vee x_3y_6z_4 = x_3 \odot \Phi_7, \text{ where } \Phi_7 = y_5z_3 \vee y_6z_4$$
$$L_3 = x_3y_5z_3 \quad \text{and} \quad U_3 = y_5z_3 \vee x_3y_6z_4$$
$$\Phi_4 = y_1z_1 \vee y_2z_2 \vee x_2y_3z_1 \vee x_2y_4z_2$$
$$L_4 = y_1z_1 \vee x_2y_4z_2 \quad \text{and} \quad U_4 = y_1z_1 \vee y_2z_2 \vee x_2y_3 \vee y_4$$
$$\Phi_5 = x_2y_3z_1 \vee x_2y_4z_2 = x_2 \odot \Phi_6, \text{ where } \Phi_6 = y_3z_1 \vee y_4z_2$$
$$L_5 = x_2y_4z_2 \quad \text{and} \quad U_5 = x_2y_3z_1 \vee y_4z_2$$

Fig. 5 D-tree for formula Φ_1 from Fig. 2 used in Example 13. Some inner nodes carry an additional label (e.g. $\Phi_2 = \oplus$) in order to better illustrate the decomposition steps taken. In the *top* d-tree, colored areas represent d-trees T_1 (*darkest*) through T_5 (*lightest*) discussed in Example 13. In the *bottom* d-tree, colored areas represent root-to-leaf paths as discovered by the algorithm's depth-first exploration: *darker* paths are discovered earlier, *lighter* paths are discovered later. At each point in time, the algorithm needs to memorize only one of these paths

For numerical calculations, we use the following probabilities: $P_{x_1} = 0.2$, $P_{x_2} = 0.4$, $P_{x_3} = 0.4$, $P_{y_1} = 0.4$, $P_{y_2} = 0.1$, $P_{y_3} = 0.0$, $P_{y_4} = 0.7$, $P_{y_5} = 0.5$, $P_{y_6} = 0.3$, $P_{z_1} = 0.4$, $P_{z_2} = 0.9$, $P_{z_3} = 0.5$, $P_{z_4} = 0.3$.

1st Iteration. We start with the d-tree $T_1 = \Phi_1$ and compute model-based bounds L_1 and U_1 that yield probability bounds $[0.348,\ 0.809]$. This is not a 0.1-approximation, so we next decompose Φ_1 using independent-or: $\Phi_1 = \Phi_2 \otimes \Phi_3$.

2nd Iteration. We obtain bounds $[0.348,\ 0.809]$ for the probability of the d-tree $T_2 = \Phi_2 \otimes \Phi_3$, where we used the model-based bounds in Fig. 5 for Φ_2 and Φ_3 with probability bounds $[0.276,\ 0.736]$ and $[0.1,\ 0.277]$. T_2's bounds are as in the 1st Iteration, since independent-or partitioning does not alter iDNF bounds, and are thus not tight enough.

We next check whether we can close Φ_2 and compute the pessimistic bound $L(T_2)$ using the above bounds for Φ_2 and the point interval defined by a lower bound for the open node Φ_3. The pessimistic lower bound $L_2 \otimes L_3$ has probability 0.348, and the upper bound $U_2 \otimes L_3$ has probability 0.809.

These bounds fail the stopping condition for both relative and absolute errors; hence, we cannot close Φ_2. We further decompose Φ_2 using Shannon expansion by variable x_1.

3rd Iteration. The bounds now become [0.348, 0.753] for d-tree $T_3 = ((x_1 \odot \Phi_4) \oplus (\neg x_1 \odot \Phi_5)) \otimes \Phi_3$, yet no 0.1-approximation is reached. We continue with Φ_4 and first check whether we can close it. The pessimistic bounds are [0.348, 0.420] and suffice for both relative and absolute approximation; we thus close Φ_4. We continue depth-first with Φ_5; this node cannot be closed and we decompose it into $x_1 \odot \Phi_6$.

4th Iteration. Since Φ_6 is in 1OF, we can directly compute its probability 0.63. The resulting d-tree T_4 has bounds [0.348, 0.534] that constitute an absolute 0.1-approximation interval $[0.534 - 0.1,\ 0.348 + 0.1] = [0.434,\ 0.448]$.

Since we have not yet reached a relative approximation, we decompose the next open leaf in depth-first traversal without known exact probability, i.e., Φ_3, into $x_3 \odot \Phi_7$.

5th Iteration. We detect that Φ_7 is in 1OF and compute its probability 0.318. The resulting d-tree T_5 has bounds [0.368, 0.438] and is thus a relative 0.1-approximation interval $[(1 - 0.1) \cdot 0.438,\ (1 + 0.1) \cdot 0.368] = [0.3942,\ 0.4048]$.

4.2.4 Considerations for an efficient implementation

We next discuss several aspects regarding the efficient implementation of our algorithm. The following desiderata have been considered in the design of the algorithm: (1) The input formula can be very large and copying parts of it after each decomposition step is not feasible. (2) For each decomposition step, we should only touch necessary nodes and as few times as possible. (3) We should use memory-conscious data structures to support our decomposition steps.

Formulas are decomposed into subformulas such that the latter represent syntactic restrictions of the former. This motivates the following compact representation of formulas in our implementation: Whenever a formula Ψ is constructed by decomposing a formula Φ, instead of explicitly storing Ψ, we represent it by a *mask* applied to Φ, where we specify which clauses and literals from Φ make up the new formula Ψ. A mask together with the original formula uniquely identify any subformula obtained by decomposition.

Algorithm 5 depicts the procedure Mask. When called with truth value b on a node n in the parse tree of a formula Φ in NNF, it computes the formula obtained from Φ by setting the variable x in node n to b and propagating $\top$ and $\bot$ in the parse tree of the formula by means of the equivalences $\bot \wedge \Psi \equiv \bot, \top \wedge \Psi \equiv \Psi$, $\bot \vee \Psi \equiv \Psi$, and $\top \vee \Psi \equiv \top$.

Example 14 Consider Shannon expansion for variable x_1 in formula Φ from Fig. 6a. Instead of constructing data structures for the formulas $\Phi|_{x_1 \leftarrow \top}$ and $\Phi|_{x_1 \leftarrow \bot}$, we can mask the parts of the formula affected by setting x_1 to $\top$ and $\bot$. The result of this masking operation is depicted in Fig. 6b, c; when the blue parts are ignored, those parse trees represent the formulas $\Phi_{x_1 \leftarrow \top}$ and $\Phi_{x_1 \leftarrow \bot}$.

Algorithm 5: The masking procedure. We set a leaf node n to truth value b and then further simplify the formula.

```
MASK(NODE n, BOOL b, ROOT NODE r)
    switch n do
        case leaf node
            if n represents positive literal x then
                MASK(n.PARENT(), b, r)
            else
                ▷ n represents negative literal ¬x
                MASK(n.PARENT(), ¬b, r)
        case inner node
            if (b=true and n=∨) or (b=false and n=∧) then
            if n.ALLLEAVESMASKED() and n ≠ r then
                MASK(n.PARENT(), b, r)
```

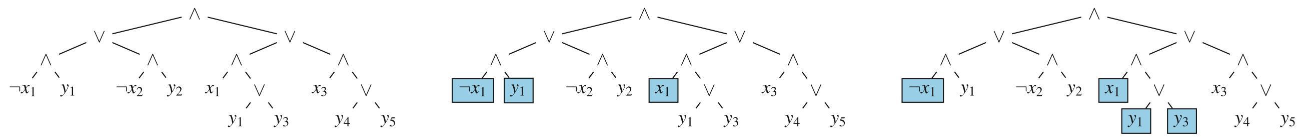
In order to compute $\Phi|_{x_1 \leftarrow \top}$, we need to mask every occurrence of x_1 in Φ with *true*, cf. Fig. 6a. We hence mask the leaf $\neg x_1$; since it is unsatisfiable under $x_1 \leftarrow \top$ and since its parent p is $\wedge$, we mask all other leaves under p in order to propagate the identity $\bot \wedge \Psi \equiv \bot$. We move up to the parent of p. This is a $\vee$-node with unmasked leaves and we stop. We then mask the second leaf of x_1. We stop since its parent is a $\wedge$-node that has un-masked leaves. The masked tree is shown in Fig. 6b and represents $\Phi|_{x_1 \leftarrow \top}$.

To compute $\Phi_{x_1 \leftarrow \bot}$, we mask all occurrences of x_1 to false: the leftmost leaf $\neg x_1$, and then the second leaf of x_1 together with all the other leaves of its parent.

Besides its use to encode subformulas obtained by decomposition, this masking procedure can also be used to compute 1OF bounds for formulas as defined in Sect. 3.3.

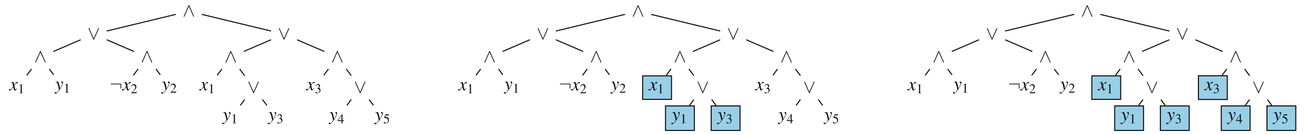
Example 15 1OF lower and upper bounds for unate formulas can be obtained by setting all but one occurrence of each variable to false and true, respectively. This can be implemented using masking: Both x_1 and y_1 occur twice in formula Φ of Fig. 7a; masking the second occurrences of x_1 and y_1 to false (true), yields a lower (upper) bound, cf. Fig. 7b, and c, respectively.

Our approach to masking leaves has proven very effective in capturing the first two above-mentioned efficiency desiderata. To speed it up, and in response to the third desideratum, we record at each inner node the start and end index of the leaf nodes under that node in the list of all leaf nodes. The masking status of leaves is kept in a bitset, with one bit per leaf. Checking whether all leaves under a given node are masked then requires to check whether a range of bits in a bitset is set. Masking is compositional: Subsequent masking can be added to an existing one by a bitwise-or operation.



(a) $\Phi = (\neg x_1 y_1 \vee \neg x_2 y_2)[x_1(y_1 \vee y_3) \vee x_3(y_4 \vee y_5)]$ **(b)** $\Phi|_{x_1 \leftarrow \top} = \neg x_2 y_2[y_1 \vee y_3 \vee x_3(y_4 \vee y_5)]$ **(c)** $\Phi|_{x_1 \leftarrow \perp} = (y_1 \vee \neg x_2 y_2) x_3 (y_4 \vee y_5)$

Fig. 6 Masking for Shannon expansion: Parse tree for Φ, and after masking all occurrences of x_1 to true and false, respectively



(a) $\Phi = (x_1 y_1 \vee \neg x_2 y_2)[x_1(y_1 \vee y_3) \vee x_3(y_4 \vee y_5)]$ **(b)** Lower bound $(x_1 y_1 \vee \neg x_2 y_2) x_3 (y_4 \vee y_5)$ **(c)** Upper bound $x_1 y_1 \vee \neg x_2 y_2$

Fig. 7 Masking for bounds: Parse tree for Φ, and for bounds of Φ obtained by masking the second x_1 and y_1 to false and true, respectively

Efficient independent decompositions are implemented using a linear algorithm for finding connected components in a graph. We keep a bitset to encode which variables occur under each child of an and/or-operation. Checking independence can then be formulated as bitset intersection. If two children are not independent, then the explanation is given by means of variables with bits set in this intersection. Also, 1OF formulas are not decomposed further; instead, we can directly evaluate their probability in one pass.

5 Experimental evaluation

In this section, we report on experiments with our approximate and exact techniques and existing techniques used by probabilistic database management systems. We found that our approximation technique is consistently more efficient than sampling-based approximation and offers a reliable, significant performance advantage over exact methods, even for small relative error thresholds. It can also keep pace with state-of-the-art techniques for tractable queries.

Algorithms. We compare the following algorithms:

MC (Monte Carlo) computes an FPTRAS approximation of result probabilities. MystiQ [46] uses an MC variant based on the Karp–Luby unbiased estimator for DNF counting [33]. MayBMS [27] uses the probabilistic adaptation of a version of the Karp–Luby estimator described by Vazirani [57] which computes fractional estimates that have smaller variance than the zero-one estimates of the Karp–Luby estimator. It combines this estimator with the Dagum–Karp–Luby–Ross optimal algorithm for Monte Carlo estimation [9], which determines the number of invocations of the Karp–Luby estimator needed to achieve the required bound by running the estimator a small number of times to estimate its mean and variance. We use the MayBMS implementation and set the probabilistic guarantee to 0.0001.

TRACT is the algorithm implemented in SPROUT for exact probability computation of tractable queries [39,40].

d-tree and d-tree$^\epsilon$ are the exact and approximate probability computation techniques as presented in this article.

Experimental Setup. The experiments were performed on an AMD 64 x2 Dual Core Processor 4600+ (2.4 GHz), 2 GB running Linux 3.2.0-32, gcc 4.6.3, JDK1.6. Our algorithms are implemented in C (for positive queries) and Java/C (for queries with negation). They are integrated in the SPROUT query engine, which is an extension of PostgreSQL8.3.3.

Queries are executed in the psql shell and their results materialized to disk. Each point in the reported plots represents 25 runs. We used five different seeds to generate random input probabilities; for each seed, we run the query five times and report average wall-clock execution times after dropping the largest and smallest times. For TPC-H queries, we report the average times and standard deviation over the five seeds. For the other queries, we only report the average wall-clock execution times.

Experiment Design. We provide insight into the performance of our techniques across a variety of queries.

Queries. We consider four classes of queries: tractable TPC-H conjunctive queries, intractable (#P-hard) TPC-H conjunctive queries, intractable conjunctive queries that express patterns in graphs, and TPC-H queries with negation and self-joins. Queries and statistics for formulas annotating their results are given in the electronic supplementary material.

Probabilistic databases. We generated tuple-independent TPC-H databases up to scale factor 1 using TPC-H 2.7.0 and annotate tuples with distinct Boolean random variables whose probabilities are drawn at random. We also used block-independent-disjoint tables representing graphs as edge relations with two records for each edge encoding the probability

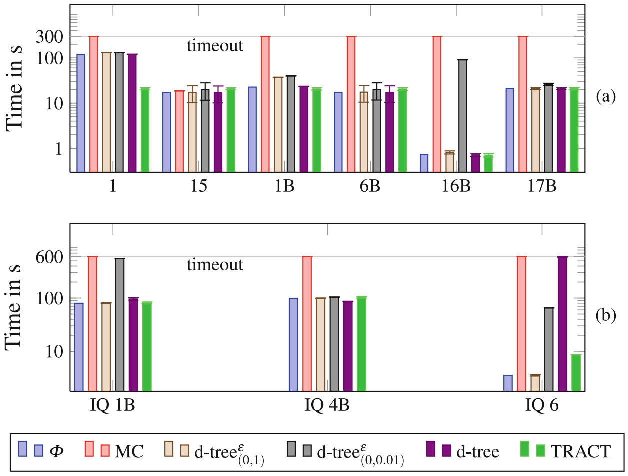


Fig. 8 Experiments with tractable TPC-H queries on tuple-independent TPC-H data with scale factor 1. The graphs compare Monte Carlo (MC) and d-tree-based (d-tree$^{\epsilon}$) relative ϵ-approximations with exact d-tree-based (d-tree) and TRACT performance. For the experiments d-tree$^{\epsilon}_{(0,0.01)}$ and d-tree$^{\epsilon}_{(0,1)}$, the probability of each tuple being present in the database is drawn from the intervals (0, 0.01) and (0, 1), respectively. The time taken to compute the result tuples and their annotations is depicted in column Φ. The x-axis specifies the queries used. In queries in (**a**), aggregations and inequality joins are dropped; queries in (**b**) use inequality joins. All approximations are relative with error $\epsilon = 0.01$

for being in and missing from the graph. The queries under consideration also generate complex and highly correlated formulas that annotate the query results.

Easy-hard-easy pattern. In earlier work [35], we observed an easy-hard-easy pattern similar to those in combinatorial algorithms for propositional satisfiability and constraint satisfaction: When the ratio of the number of variables to formula size is large, the formula readily decomposes into independent parts. Conversely, if there are only few variables in a large formula, then a few Shannon expansion steps completely decompose the formula. In both cases, satisfiability, model counting, and probability are efficiently computable. However, in-between there is a critical region of variable-to-formula-size ratios where probability computation is hard. There is hence a pitfall in increasing the instance sizes in experiments: If we do not proportionally add interesting variability (and increase the probability space), then the instances get easier rather than harder. On the other hand, an easy-hard-easy pattern is also good news, because it shows that hard instances are only restricted to a narrow section of the space of possible input instances.

Absolute versus relative approximation. For probabilities close to 1, our algorithm behaves similarly for both absolute and relative approximation. To study relative approximation, we thus have to construct instances with small result probabilities. As pointed out in the previous paragraph, this is not entirely trivial. However, understanding the properties of relative approximation for our algorithm is important, since relative approximation is a staple of MC.

Experiments with tractable TPC-H conjunctive queries. We consider tractable conjunctive queries without self-joins (1, 15, 1B, 6B, 16B, 17B) and with inequality joins (IQ1B, IQ4B, IQ6) from the literature [40,39]; queries marked with "B" are Boolean, i.e., they return the probability that the query is true. These are modified versions of standard TPC-H queries without negation and aggregation but with special aggregates for probability computation. The queries 1, 1B, 6B are selections on the large lineitem table, all other queries are joins of two large tables (e.g., lineitem with supplier, orders, or part). Each query IQ1B, IQ4B, and IQ6 joins two relations using an inequality join and an equality join.

Figure 8 shows the running times of these queries on tuple-independent probabilistic TPC-H databases of scale factor 1. As expected, TRACT performs best in most cases, since it is specifically designed to tackle tractable queries. MC times out in all cases but query 15 (where the result annotation consists of three clauses), as it does not exploit the structure of the annotations for tractable queries for efficient evaluation; it needs a number of runs proportional to the number of clauses, even if these clauses are pairwise independent and allow for trivial probability computation.

The plots show that our exact and approximation algorithms are able to detect the tractable structure in the input without knowledge of the query. They are very competitive as they perform about the same or even better than TRACT in some cases. When they need visibly much more time than TRACT, it is due to the overhead of searches for independence partitioning of the formulas at the leaves and, in case of approximation, also of lower and upper bounds computation. In case of query 1 (and its Boolean version, 1B), the formula is a disjunction of independent literals; by static analysis of the query, TRACT expects this and can efficiently compute the overall probability, whereas our algorithms first need to discover the independence of the literals. d-tree$^{\epsilon}$ can be better than TRACT, e.g., for queries 15, 6B, and IQ6, if it reaches an approximation in less than linearly many decomposition steps, as required by TRACT.

A further observation is that most of the execution time is dedicated to constructing the result tuples and their annotations (leftmost bar, Φ). TRACT even outperforms Φ in some cases since it can effectively use PostgreSQL's pipelining mechanism to access annotations as they are produced and need not wait for the entire result to materialize.

The standard deviation is small for the exact and approximation algorithms when run over data with five different seeds for input probabilities. This suggests that similar input probability distributions lead to similar performance of our algorithms. When the input probabilities are randomly drawn from the much smaller interval (0,0.01), d-tree$^{\epsilon}$ needs to work harder and dig deeper in the decomposition tree to gain probability mass and it hence makes a small progress with each decomposition step, e.g., for queries 16B, IQ1B, and IQ6. In particular, for input probabilities drawn from (0,1), the probability of query 16B is very close to 1 and d-tree$^{\epsilon}$ converges quickly; for input probabilities drawn from (0,0.01), the query probability is approximately 0.47 and d-tree$^{\epsilon}$ takes longer to converge.

Experiments with intractable TPC-H conjunctive queries. We consider four Boolean intractable conjunctive queries: Query 20B is a join on supplier, nation, partsupplier, and part; query 21B is a join on supplier, lineitem, orders, and nation; query 2B is a join on part, supplier, partsupplier, nation, and region; and query 9B is a join on part, supplier, lineitem, partsupplier, orders, and nation.

Figure 9 depicts the time needed by d-tree$^{\epsilon}$ and MC to compute the approximate probabilities of the four queries. Our algorithm d-tree$^{\epsilon}$ outperforms MC for such queries on tuple-independent TPC-H data of varying scale factors. These queries have many joins, which leads to low marginal probabilities of clauses, while formulas annotating the results have up to 400 clauses and 1,000 variables (query 20B), up to 75,000 clauses and 150,000 variables (query 21B), up to 640 clauses and 1,600 variables (query 2B), and up to 350,000 clauses and 729,000 variables (query 9B).

Statistics collected from runtime traces show that in general, as the size of formulas increases, so does the number of decomposition steps. However, two scenarios may change this trend. Firstly, for bound computation, with more input clauses, both the lower and upper bounds increase while maximal values of upper bounds are 1. If upper bounds reach 1 and lower bounds still increase, this can lead to quick convergence. This is for instance the case for query 9B for scale factor 1. Secondly, the formulas of some TPC-H queries (that have equality selections with constants) have the property that very few variables from one input table occur in most of the clauses. For instance, for queries 20B and 21B, there is only one variable coming from table nation. After eliminating this variable, the residual formula has many independent clauses and the approximation approach captures this and tightens the lower and upper bounds very quickly. Hence, the number of decomposition steps constructed remains small and is not affected by the formula size.

As shown in the bottom-right plot in Fig. 9, most of the query execution time for d-tree$^{\epsilon}$ is spent in the first phase of constructing the result tuples and their annotations; except for query 21B, d-tree$^{\epsilon}$ needs less than 10 % of the overall execution time.

The top-right plot in Fig. 9 shows that for query 2B, the bounds computed at each decomposition step quickly achieve a relative error of 0.01. The plateau witnessed for scale 0.1 actually represents very small improvements made to both lower and upper bounds over 60 steps. For scales 0.01, 0.5, and 1, it converges in maximum five steps.

Experiments with intractable positive graph queries. We next study queries on databases that encode social networks. We consider two classes of datasets modeled as block-independent disjoint tables. The first set consists of synthetic random graphs where all edges have the same probability P_e. An undirected random graph with n nodes is a probabilistic database in which the possible worlds are the subgraphs (obtained by removing zero or more edges) of the n-clique. In the case of $P_e = 1/2$, the probability distribution over this set of possible worlds is uniform and each world has probability $(1/2)^{n \cdot (n-1)}$. The second class of graph datasets are well-known social networks taken from the literature: One is Zachary's classic "Karate club" [60] with 34 nodes, and the other represents friendship among a group of dolphins.

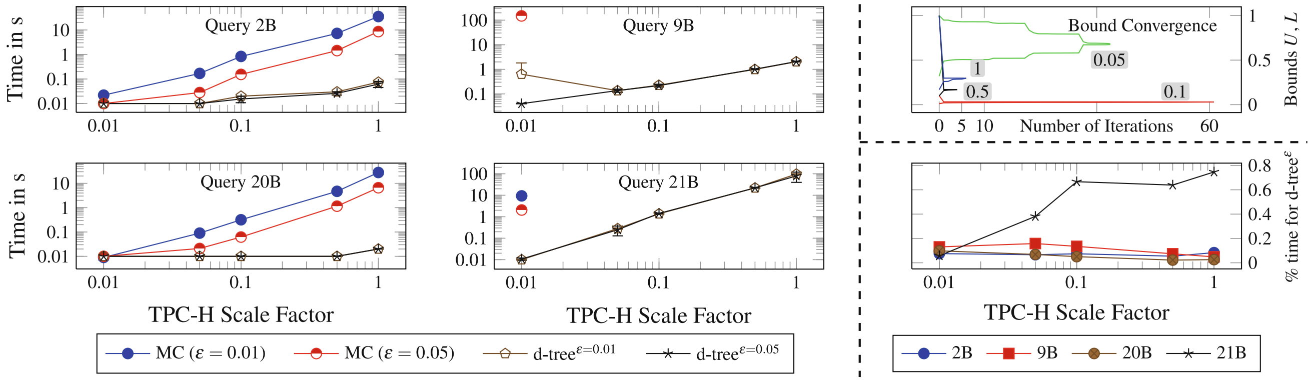


Fig. 9 Experiments with intractable TPC-H queries 2B, 9B, 20B, 21B on tuple-independent tables. The left and middle graphs compare Monte Carlo-based (MC) and d-tree-based (d-tree$^{\epsilon}$) relative ϵ-approximation performance for TPC-H scale factors 0.01 through 1; the timeout is set to 300 s. The bottom-right figure shows the percentage of the total query answering time taken by approximate probability computation with d-tree$^{\epsilon=0.01}$ for the four queries. The top-right figure shows the convergence of lower and upper bounds (y-axis) after each decomposition step (x-axis) for d-tree$^{\epsilon=0.01}$, query 2B and scales 0.05, 0.1, 0.5, 1; scale 0.01 behaves similarly to 0.5

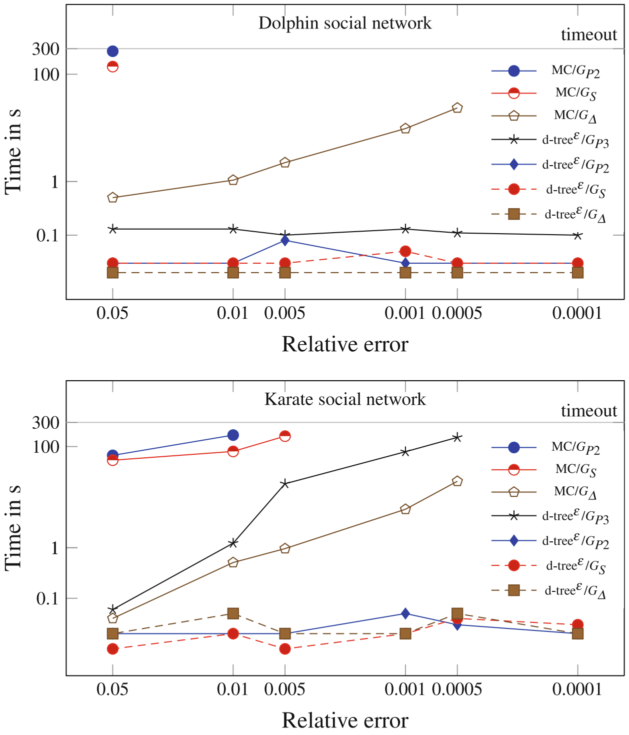


Fig. 10 Experiments with graph queries on social networks

The social networks generalize random graphs in that some edges are missing with certainty, and the remaining edges have varying probability of being present in the graph. The idea here is that friendship between nodes is established by observation and there may be a varying degree of confidence in that a pair of nodes are friends (for dolphins), or varying degrees of friendship (for karatekas).

The four queries in our experiments detect the following patterns in graphs: triangles (G_Δ), paths of length 2 (G_{P2}), paths of length 3 (G_{P3}), and pairs of nodes that have at most two degrees of separation (G_S).

Our experimental results for queries on social networks and random graphs are reported in Figs. 10 and 11. In case of random graphs, for large edge probabilities (above 0.5), d-tree$^\epsilon$ converges quickly, since each clause has a non-negligible marginal probability. When we consider smaller edge probabilities (below 0.1), d-tree needs more time to converge, especially for queries involving more joins (such as the path queries). We witness an easy-hard-easy pattern for edge probabilities of 0.3 in case of G_Δ and G_{P2}.

It is worth pointing out that while the random graphs and social networks used here (on the order of 50 nodes) may not seem very large, they are actually substantial; a 40-nodes random graph has up to 780 edges. The triangle query uses a three-way self-join and generates a formula of 780 variables and 9,880 clauses; the path query G_{P2} uses a three-way self-join and generates 60,000 clauses; the path query G_{P3} uses a six-way self-join.

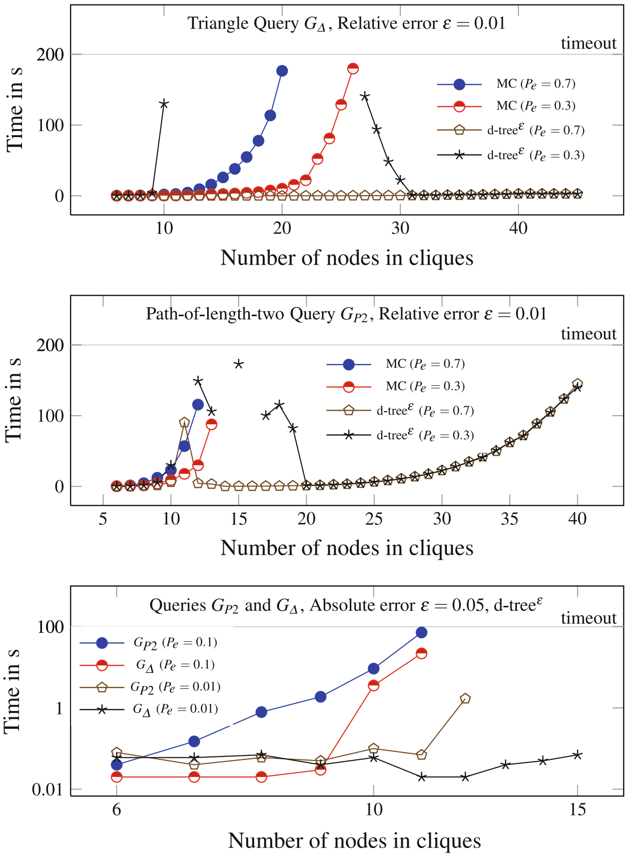


Fig. 11 Experiments with graph queries on random graphs. P_e is the probability for each edge to be in the graph

Experiments with TPC-H queries with negation. Let us now turn to queries with negation. We investigate four such queries on tuple-independent TPC-H databases: N1 lists all parts that were only ever sold at quantities of at least five per order; N2 lists all parts that were only ever sold at quantities of at least five per order and whose availability is smaller than 8,500 at all suppliers; N3 lists all suppliers that offer all parts with certain brand and size; and N4 lists all redundant suppliers, i.e., suppliers for parts that are also available from another supplier.

The number of tuples in the query answers is at least one order of magnitude larger than in the deterministic case. This is because tuples that do not qualify in the answer to a difference operation in the deterministic case may qualify with a nonzero probability in the probabilistic case. We note that this is specific to queries with difference and does not happen for positive queries.

Another aspect concerns the size of annotations in query results. For all queries but N3, the average size stabilizes when the scale factor increases. This is inherent to TPC-H data: The number of input tuples that participate in the generation of one answer tuple remains roughly constant in case of our queries, regardless of the database size. However, the number of result tuples increases. The size of N3's answer is expected to increase with the scale factor as it depends on the number of suppliers.

Regarding running time, the computation of the answer tuples and their annotations is rather close to the deterministic case for our queries, since in both cases, the queries have to "touch" all the relevant input tuples and most of the intermediate tuples. The overhead due to the materialization of more tuples in the answer and to the computation and materialization of the annotations is below 100 % for N1, N2, and N4, but becomes one order of magnitude for N3.

Figure 12 compares the d-tree and d-tree$^\epsilon$ algorithms for queries N1, N2, N3, N4 on the TPC-H datasets. Queries N1 and N2 are tractable [20], and both the exact and approximate d-tree algorithms perform within 10 seconds for scale factor 1. Although the answer sizes increase linearly with the scale factor, the average annotation size stays nearly constant (between 30 and 40 bytes); thus, the performance of probability computation is of the same order of magnitude for all scales. Query N3 exhibits a considerable performance gap between exact and approximation computation: d-tree$^\epsilon$ performs better than d-tree, since it can find good 1OF bounds and stops.

Averaging over runs with different randomly assigned probabilities to input variables yields runtime standard deviation of less than 10 % in case of query N3. However, we found this variation to be of the same order of magnitude as the runtime fluctuation over different runs with the same probability assignments to the input variables.

We verified the effectiveness of our heuristic for computing 1OF lower and upper bounds. In all scenarios with hard queries (N3, N4), all computed bounds were non-trivial, except for non-unate formulas where we first had to apply Shannon expansion on the variables occurring with both polarities. Non-trivial means here that the lower bound is not 0, unless the upper bound is also zero, and similarly, the upper bound is not 1, unless the lower bound is also 1.

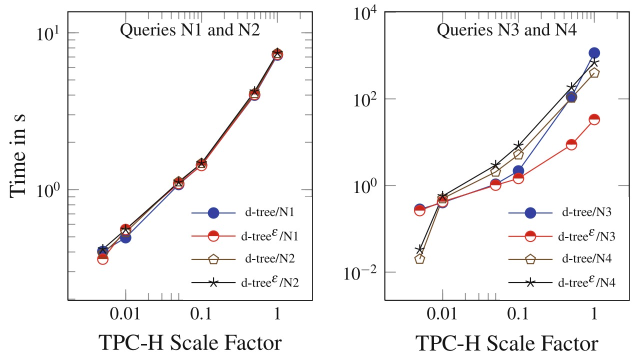


Fig. 12 Performance of probability computation in the TPC-H scenario for queries with negation. All experiments use a relative error $\epsilon = 10^{-4}$

The lower and upper bounds are also tight. We recorded all bounds over all the experiments and computed the mean 10^{-4} and standard deviation 10^{-2} of their gaps.

6 Related work

We next survey related work on model-based approximations and approximate computation in #SAT and databases.

Model-based Approximations. The closest in spirit to this work on model-based bounds is by Selman on approximating CNF theories by model-based lower and upper bound conjunctions of Horn clauses [50]. The languages iDNF and 1OF used in this article are incomparable to conjunctions of Horn clauses and are natural in the context of probability computation due to their efficiency and connection to tractability of query evaluation in probabilistic databases.

The 1OF language has a long history and many names, such as read-once functions [23], fanout-free functions, or non-repeating trees [44], and also many applications including logic synthesis and circuit design [44] and probabilistic databases [38]. Problems such as satisfiability, model counting, and probability computation are hard for general propositional formulas but tractable for 1OF formulas. The equivalent 1OF of any positive DNF formula, if it exists, can be found in polynomial time [23] and is unique up to commutativity of the binary connectives [44]. The 1OF language is particularly relevant in the context of probabilistic databases, since the formulas annotating the results of tractable conjunctive queries without repeating relation symbols on tuple-independent probabilistic databases admit 1OF equivalents [38,51] and their probability can be computed using relational query plans [11,40].

Beyond 1OF, tractability of probability computation is lost quickly. It is NP-hard to decide if a positive formula admits a read-twice equivalent formula, i.e., a formula where every variable occurs at most twice [16]; this is still open for positive DNF formulas. For read-4 formulas, model counting, and hence probability computation, is #P-hard [56].

A different direction is to consider complete languages to express the bounds, i.e., languages that can represent any propositional formula and that still allow for efficient probability computation. A prime example of such languages is the family of Binary Decision Diagrams, including OBDDs, FBDDs, and d-DNNFs [13]. Such languages allow for several equivalent representations of a formula but with an exponential gap between their sizes. Finding a minimal representation in any of these languages is NP-hard [37]. D-trees generalize ws-trees [35] with independent-and decompositions, which are crucial for the treatment of tractable queries because they capture 1OF formulas [38]. We have also generalized the formalism to partial decompositions, which are the foundation of the approximation techniques in Sect. 4.2. The and/xor trees of [36] are modeled on the ws-trees but are a weaker representation system in that they have tuples, rather than clauses, at their leaves.

An alternative approach to computing upper probability bounds for positive DNF formulas has been proposed in the context of propagation scores of conjunctive queries [22]. These bounds are not model-based, and in particular not optimal. Although the formulas annotating query results can be used to compute upper bounds on their probabilities, their interpretation is non-standard: Each occurrence of a variable v is considered a fresh variable with the same probability as v. Arbitrary positive DNF formulas are thus seen as iDNF formulas. One subtlety regarding the applicability of this method to formulas annotating results of queries with self-joins concerns clauses with multiple occurrences of the same variable. Consider the formula $\Phi = (x \vee y) \wedge x$, where x occurs twice. This approach would thus consider instead the formula $\Phi' = (x_1 \vee y) \wedge x_2$, where x_1 and x_2 are distinct but have the probability of x. Since $\Phi = x$, it follows that Φ' is *not* an upper bound of Φ, but in fact a lower bound. Whereas in a DNF representation we could trivially remove such redundancies, this is not always possible for nested formulas without an exponential blowup in the formula size. A possible approach that is easy to integrate in our algorithm is as follows: We detect clauses with multiple occurrences of the same variable x by checking whether any two leaves of x have an and-operation as a common ancestor. If this is the case, we replace one of the occurrences by *true* and mask.

Ré and Suciu discuss model-based lower bounds and Taylor/Fourier approximation of positive DNF formulas [47]. Their focus is on compressing a DNF formula by eliminating subformulas that only account for up to a given threshold of its probability mass.

Application to provenance databases. Besides probabilistic databases, our model-based approximations can also benefit provenance management for annotated databases. Similarly to the probabilistic case, formulas annotating query results encode symbolically all possible explanations for the existence of a tuple in the query answer in terms of combinations of the input tuples and can be interpreted as the provenance of query results. Provenance bounds can then represent coarser, more compact, and efficiently computable explanations of the query result. Lower bounds are *correct but possibly incomplete* explanations, while upper bounds are *complete but potentially incorrect* explanations in the following sense: Every explanation for a lower bound is necessarily an explanation for the result; however, there may exist explanations for the result that are not explanations for the lower bound. The situation for upper bounds is symmetric.

Application to relational databases. Existing work on approximate query answering in relational databases considers the use of synopses (histograms, join synopses, and wavelets) to speedup the evaluation of aggregate queries with the goal of quickly reporting the leading digits of the answers. A survey of these synopsis-based techniques has been recently compiled [8]. Their focus is not on deriving optimal bounds within a given language.

Given a query Q in a query language $\mathscr{L}$, our approximation problem in the relational setting is to compute two queries Q_L and Q_U in a query language $\mathscr{L}'$ such that $Q_L(D) \subseteq Q(D) \subseteq Q_U(D)$ for any relational database D and the computational complexity for $\mathscr{L}'$ is lower than for $\mathscr{L}$. Within this framework, recent work characterizes lower bounds for conjunctive queries where the bounds themselves are expressed as conjunctive queries [4,19].

#SAT. There is a wealth of literature on approximate and exact techniques for model counting used in #SAT solvers [24]. Some of these techniques consider formulas with various restrictions (such as bounded treewidth) or focus on lower-bounding in extremely large combinatorial problems, with bounds off the true count by many orders of magnitude, e.g., [59]. Extensions of the Davis–Putnam procedure (which is based on Shannon expansion) have been used for counting the solutions to formulas [5]. The decomposable Negation Normal Form [13] (and variations thereof) is a propositional theory with efficient model counting, which uses Shannon expansion and independence partitioning. Our previous work [35] uses similar ideas to design an exact probability computation algorithm, although without polynomial-time guarantees for tractable queries. These approaches do not consider model-based approximation of propositional formulas. Our decomposition approach shares ideas with these techniques, yet two of its main aspects remain novel: (1) the combination of incremental compilation and of model-based bounds for fast approximate computation with error guarantees, and (2) polynomial-time evaluation for classes of tractable queries on probabilistic databases.

A different line of research is on randomised approximation algorithms. It was first shown by Karp, Luby, and Madras [33] that there is a fully polynomial-time randomised approximation scheme (FPTRAS) for DNF counting based on Monte Carlo simulation. This algorithm can be modified to compute the probability of a DNF over independent discrete random variables [12,25,34,46]. These techniques yield an efficiently computable unbiased estimator that in expectation returns the probability p of a DNF of n clauses such that computing the average of a polynomial number of such Monte Carlo steps (which are calls to the Karp–Luby unbiased estimator) is an (ϵ, δ)-approximation for the probability (i.e., a relative approximation): If the average $\hat{p}$ is taken over at least $\lceil 3 \cdot n \cdot \log(2/\delta)/\epsilon^2 \rceil$ Monte Carlo steps, then $\Pr\left[|p - \hat{p}| \geq \epsilon \cdot p\right] \leq \delta$.

In contrast to our algorithm, the Monte Carlo algorithm runs in polynomial time, yet it has the following limitations: (1) It can only guarantee probabilistically the computed approximation; (2) running one more Monte Carlo step does not necessarily lead to a refinement of probability bounds, and hence the approximation is not truly incremental [53]; (3) it only works for DNF formulas; (4) it sees the input as a black box and does not exploit their structure for faster evaluation [35]. Limitations (1) and (2) have been shown to be a fundamental constraint for its applicability to ranking in probabilistic databases [43]. In order to decide the relative rank of two answer tuples, it suffices to approximate their probability intervals until they are disjoint; this requires the approximation to be both deterministic and incremental. Limitation (3) prohibits the use of the Monte Carlo algorithm for queries with negation. Limitation (4) means that it does not understand tractable queries and cannot distinguish between structurally simple (such as when the ratio of clauses to variables is very small or very large) and complex formulas.

The work by Karp, Luby, and Madras has started a line of research to derandomise these approximation techniques, eventually leading to a polynomial time deterministic $(\epsilon, 0)$-approximation algorithm [54] (for k-DNF, i.e., the size of clauses is bounded, which is not an unrealistic assumption for probabilistic databases, where k is bounded by the number of joins for DNFs constructed by positive relational algebra). However, the constant in this algorithm is astronomical: This constant is above 2^{50} for 3-DNF.

Probabilistic databases. Query evaluation is a core task in probabilistic databases, and restricted instances of this problem have been much investigated in recent years [53]. Common restrictions consider conjunctive queries without self-joins and tuple-independent databases, e.g. [12,40]. Beyond these restrictions, there is work on tractability of positive queries [10], on queries with aggregates [17,48] and with negation [20,58]. The approximate probability computation algorithm described in this article is used by the SPROUT query engine [18] to complement its optimized exact algorithm for tractable conjunctive queries without self-joins [40]. It considers relational algebra queries and pc-tables that go beyond these restrictions. This article unifies work reported previously on decomposition-based approximation for positive queries [41] and for queries with negation [20] and on model-based approximation [19]. The approximation algorithms used by the probabilistic database management systems MystiQ [46], MayBMS [27], and MCDB [29] are variations of the Monte Carlo algorithm presented above. Although used for probability computation and not for modeling purposes, the decomposition trees resemble the probabilistic XML model with independence and mutex inner nodes and data values at leaves [1].

More recent work extends the probability computation framework presented in this article. ProApproX evaluates queries on probabilistic XML [52], where the formula annotating the query answer is compiled into a restricted d-tree without using Shannon expansion. Using cost-based heuristics, different approximation algorithms, including the Monte Carlo algorithm discussed above, are run at the leaves; the bounds at the leaves are then used to derive overall probability bounds as discussed in Sect. 4. Top-k queries can be evaluated by incrementally approximating the probabilities of the results using our anytime algorithm until the lower bounds of k tuples are greater than the upper bounds of the remaining tuples [43]. Our model-based bounds were also used for first-order formulas, as recently proposed for top-k query evaluation in the presence of non-materialized views [15]. The d-tree formalism can support sensitivity analysis and explanation for queries [32] by efficiently computing the result probability in the face of changes to the input probabilities. An extension of our framework has been recently proposed to compile expressions over semirings and semimodules such as those defining the exact probabilities of queries with aggregates in probabilistic databases [2,17].

7 Conclusion and future work

This article presents an approximation algorithm for computing probabilities of propositional formulas over discrete random variables. This is used by the SPROUT query engine to compute confidences in results to relational queries on probabilistic databases. Approximation is key to scalable query evaluation in probabilistic databases since already simple queries have #P-hard data complexity.

The proposed approximation algorithm has two key ingredients: an incremental decomposition of formulas into independent or mutually exclusive subformulas, and an approach to compute model-based bounds on these subformulas and to use them to derive lower and upper bounds on the probability of the input formula. Together, the two components give rise to an anytime, memory-efficient approximation algorithm with relative or absolute error guarantees.

Extensions of the decomposition tree formalism and the model-based bounds put forward in this article have been used in the context of probabilistic databases for the exact evaluation of top-k queries [15,43] and of queries with aggregates [17], for approximate query evaluation over probabilistic XML [52], for explaining query tractability via linear-size d-trees [30,38,39], for conditioning probabilistic databases [35] and for explanation and sensitivity analysis [32].

The investigation of approximate probability computation for queries with aggregates via d-trees for semiring and semimodule expressions [2] is compelling since the performance of exact computation for such queries is prohibitively low. Another promising extension of this work is to consider model bounds with respect to more expressive formula languages such as decision diagrams.

Acknowledgments We would like to thank the anonymous reviewers and Peter Haas for their insightful comments that helped improve this article. We also thank Christoph Koch and Swaroop Rath for their collaboration on earlier work on which this article is partially based. Jiewen Huang's work was done while at Oxford.

Appendix A: Proofs

This section contains proofs of formal statements in previous sections. Due to space constraints, further proofs can be found in the electronic supplementary material.

A.1 Proof of Theorem 1

Let us first show the direction MLB ⇒ GLB. Let Φ be an irreducible positive DNF formula and Φ_L an MLB for Φ. Assume Φ_L is no GLB for Φ. Then, there exists an iDNF formula Φ'_L such that $\mathscr{M}(\Phi_L) \subset \mathscr{M}(\Phi'_L) \subseteq \mathscr{M}(\Phi)$ and the following properties hold:

- (i) $\Phi_L \subseteq \Phi$ (every clause in Φ_L is also a clause in Φ)
- (ii) Every clause φ in Φ and not in Φ_L contains a conflicting variable, i.e., a variable $x \in \varphi$ such that there exists a clause $\bar{\varphi}_L \in \Phi_L$ with $x \in \varphi_L$ (Φ_L is MLB for Φ)
- (iii) $\forall \varphi'_L \in \Phi'_L : \exists \varphi \in \Phi : \varphi'_L \supseteq \varphi$ ($\Phi'_L \models \Phi$ and Lemma 1)
- (iv) $\forall \varphi_L \in \Phi_L : \exists \varphi'_L \in \Phi'_L : \varphi_L \supseteq \varphi'_L$ ($\Phi_L \models \Phi'_L$ and Lemma 1)
- (v) No two clauses $\varphi, \bar{\varphi} \in \Phi$ satisfy $\varphi \models \bar{\varphi}$ (Φ is irreducible)

Properties (i) and (ii) follow from Φ_L being an MLB for Φ, (iii)–(iv) from $\mathscr{M}(\Phi_L) \subset \mathscr{M}(\Phi'_L) \subseteq \mathscr{M}(\Phi)$. We prove that Φ_L is equivalent to Φ'_L by case differentiation:

Case 1 $\top \in \Phi$. Then, since $\top$ has no conflicting variables with any other clause, it follows that $\top \in \Phi_L$ and thus we have $\mathscr{M}(\Phi_L) = \mathscr{M}(\Phi'_L) = \mathscr{M}(\Phi)$ which is a contradiction to the assumption that Φ_L is no GLB for Φ.

Case 2 $\top \notin \Phi$. Let $\varphi'_L \in \Phi'_L$ be any clause in Φ'_L and $\varphi \in \Phi$ a clause such that $\varphi'_L \supseteq \varphi$ according to (iii). With respect to property (i), we have the following two cases:
Case 2(a) $\varphi \in \Phi_L$. Let $\bar{\varphi}'_L \in \Phi'_L$ be a clause with $\varphi \supseteq \bar{\varphi}'_L$ according to (iv). Together with $\varphi'_L \subseteq \varphi$ from above, we have $\bar{\varphi}'_L \supseteq \varphi \supseteq \varphi'_L$, and since none of $\varphi, \varphi'_L, \bar{\varphi}'_L$ can be the empty clause (see case 1), φ'_L and $\bar{\varphi}'_L$ must share at least one variable. Since $\Phi'_L \in$ iDNF, it follows that φ'_L and $\bar{\varphi}'_L$ are the same clause. Thus, $\varphi \supseteq \bar{\varphi}'_L = \varphi'_L \supseteq \varphi$, i.e., $\varphi = \varphi'_L$. It follows that $\Phi_L = \Phi'_L$ which is a contradiction to the assumption $\mathscr{M}(\Phi_L) \subset \mathscr{M}(\Phi'_L)$.

Case 2(b) $\varphi \notin \Phi_L$. According to (ii), there is a variable $x \in \varphi$ such that there exists a clause $\varphi_L \in \Phi_L$ with $x \in \varphi_L$; furthermore, $\varphi_L \in \Phi$ due to (i). From $\varphi'_L \supseteq \varphi$, it follows $x \in \varphi'_L$. From (iv), it follows that there exist a clause $\bar{\varphi}'_L \in \Phi'_L$ such that $\varphi_L \supseteq \bar{\varphi}'_L$. We distinguish two cases:

Case 2(b) i $x \in \bar{\varphi}'_L$. It follows $\bar{\varphi}'_L = \varphi'_L$, because $\Phi'_L \in$ iDNF. From $\varphi_L \supseteq \bar{\varphi}'_L = \varphi'_L \supseteq \varphi$, it follows that $\varphi_L \supseteq \varphi$. Since $\varphi_L, \varphi \in \Phi$, this is a contradiction to the assumption that Φ is irreducible, property (v).

Case 2(b) ii $x \notin \bar{\varphi}'_L$. Then, according to (iii), there is a $\bar{\varphi} \in \Phi$ with $\bar{\varphi}'_L \supseteq \bar{\varphi}$ and thus $x \notin \Phi$ which in turn implies $\bar{\varphi} \neq \varphi$ because $x \in \varphi$. Transitivity of $\supseteq$ implies $\varphi_L \supseteq \bar{\varphi}$ which is a contradiction to the assumption that Φ is irreducible, property (v).

Secondly, we prove the direction GLB ⇒ MLB. Assume that a GLB Φ_L for Φ is no MLB for Φ. Then, at least one of the two MLB properties in Definition 2 is unsatisfied. We show that in either case, Φ_L is no GLB for Φ.

Case 1 Assume $\Phi_L \not\subseteq \Phi$. Then, Φ_L contains a clause φ_L that is not in Φ.

Case 1(a) If there is a clause $\varphi \in \Phi$ such that $\varphi_L \supseteq \varphi$, then $\varphi_L \supset \varphi$. Let x be a variable that occurs in φ_L but not in φ. Then, the iDNF formula obtained from Φ_L by removing the variable x from φ has strictly more models than Φ_L, and thus, Φ_L is no GLB for Φ.

Case 1(b) If there is no such clause, then $\Phi_L \not\models \Phi$ following Lemma 1 and thus Φ_L is no lower bound and in particular no greatest lower bound for Φ.

Case 2 If there is a clause $\varphi \in \Phi$ such that $\Phi_L \cup \{\varphi\} \in$ iDNF, then $\mathscr{M}(\Phi_L) \subset \mathscr{M}(\Phi_L \cup \{\varphi\})$ because none of the variables in φ occur in Φ_L and thus Φ_L is no GLB for Φ.

A.2 Proof of Theorem 2

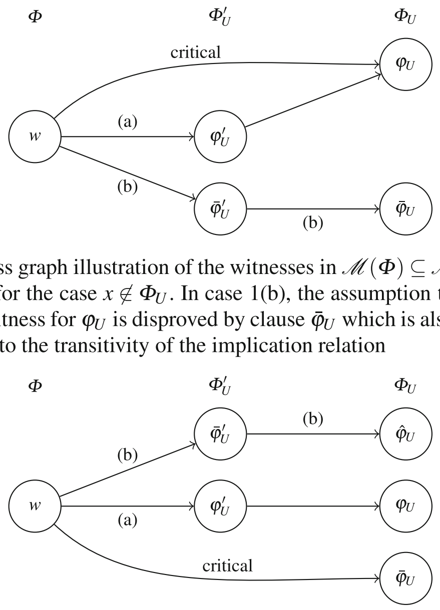
We use a graph representation of the witness relationships between clauses, cf. Fig. 13. The clauses are the nodes of the graph, and there is a directed edge between two clauses φ and ψ whenever $\varphi \models \psi$, i.e., φ is a witness of ψ.

MUB ⇒ LUB. Assume Φ_U is an MUB for Φ but not an LUB for Φ. Then, there exists a better iDNF upper bound Φ'_U for Φ that satisfies $\mathscr{M}(\Phi) \subseteq \mathscr{M}(\Phi'_U) \subset \mathscr{M}(\Phi_U)$. Using Lemma 1, the second inclusion unfolds to:

$$
\begin{aligned}
&\mathrm{Z}\ \mathscr{M}(\Phi'_U) \subset \mathscr{M}(\Phi_U)\\
&\quad \text{iff } \mathscr{M}(\Phi'_U) \subseteq \mathscr{M}(\Phi_U) \text{and not } \mathscr{M}(\Phi_U) \subseteq \mathscr{M}(\Phi'_U)\\
&\quad \text{iff } \forall \varphi'_U \in \Phi'_U : \exists \varphi_U \in \Phi_U : \varphi'_U \supseteq \varphi_U\\
&\quad \text{and } \exists \varphi_U \in \Phi_U : \forall \varphi'_U \in \Phi'_U : \neg(\varphi_U \supseteq \varphi'_U).
\end{aligned}
$$

By collecting the assumptions and unfolding the definitions:

$$
\begin{aligned}
&\text{(i)}\ \ \forall \varphi \in \Phi : \exists \varphi'_U \in \Phi'_U : \varphi \supseteq \varphi'_U && (\Phi \models \Phi'_U)\\
&\text{(ii)}\ \ \forall \varphi \in \Phi : \exists \varphi_U \in \Phi_U : \varphi \supseteq \varphi_U && (\Phi \models \Phi_U)\\
&\text{(iii)}\ \ \forall \varphi'_U \in \Phi'_U : \exists \varphi_U \in \Phi_U : \varphi'_U \supseteq \varphi_U && (\Phi'_U \models \Phi_U)
\end{aligned}
$$



(a) Witness graph illustration of the witnesses in $\mathscr{M}(\Phi) \subseteq \mathscr{M}(\Phi'_U) \subset \mathscr{M}(\Phi_U)$ for the case $x \notin \Phi_U$. In case 1(b), the assumption that w is a critical witness for φ_U is disproved by clause $\bar{\varphi}_U$ which is also implied by w due to the transitivity of the implication relation

(b) Witness graph illustration of the witnesses in $\mathscr{M}(\Phi) \subseteq \mathscr{M}(\Phi'_U) \subset \mathscr{M}(\Phi_U)$ for the case $x \in \Phi_U$. In cases 2(a) and 2(b), the assumption that w is a critical witness for $\bar{\varphi}_U$ is disproved by clause φ_U or $\hat{\varphi}_U$, respectively, which are also implied by w due to the transitivity of the implication relation

Fig. 13 Witness graphs used in the proof of Theorem 2

(iv) $\exists \varphi_U \in \Phi_U : \forall \varphi'_U \in \Phi'_U : \neg(\varphi_U \supseteq \varphi'_U) \quad (\Phi_U \not\models \Phi'_U)$
(v) There is no clause $\varphi_U \in \Phi_U$ that can be extended by a variable from vars(Φ) and the resulting formula is still in iDNF and implied by Φ
(vi) Every $\varphi \in \Phi$ is a witness for at least one clause $\varphi_U \in \Phi_U$, i.e., $\Phi \models \Phi_U$ according to Lemma 1
(vii) Every clause $\varphi_U \in \Phi_U$ has a critical witness in Φ.

Sentences (i)–(iv) are due to the assumption that Φ_U is upper bound but no LUB for Φ and sentences (v)–(vii) are the syntactical characterization of the MUB property of Φ_U.

Let $\varphi_U \in \Phi_U$ be as in (iv). Let $\varphi'_U \in \Phi'_U$ be a clause in Φ'_U that shares a variable with φ_U. If no such clause exists, then by transitivity of $\supseteq$ and $\Phi_U, \Phi'_U \in$ iDNF, Φ_U cannot have a witness in Φ which is a contradiction to (vii).

Then, since $\Phi_U, \Phi'_U \in$ iDNF, φ_U can be the only clause in Φ_U that satisfies sentence (iii) and together with $\neg(\varphi_U \supseteq \varphi'_U)$ from (iv) we can conclude $\varphi'_U \supset \varphi_U$. Let x be a variable that occurs in φ'_U, but not in φ_U. We show a contradiction to the above sentences (i)–(vii) by case differentiation:

Case 1 $x \notin \Phi_U$. According to (vii), φ_U has a critical witness $w \in \Phi$. Consider Fig. 13a.

Case 1(a) $w \models \varphi'_U$. Then, φ_U can be extended by x, and the resulting formula is still in iDNF and implied by Φ. This is a contradiction to (v).

Case 1(b) If $w \not\models \varphi'_U$. Then, according to (i), there is a $\bar{\varphi}'_U \in \Phi'_U$ such that $w \models \bar{\varphi}'_U$. Since $\Phi'_U \in$ iDNF, φ'_U and $\bar{\varphi}'_U$ share no variables; due to (iii), there must be a $\bar{\varphi}_U \in \Phi_U$ which is implied by $\bar{\varphi}'_U$ and is different from φ_U due to the iDNF property of Φ_U. From the transitivity of the implication relation, it follows $w \models \bar{\varphi}_U$ which is a contradiction to the assumption that w is a critical witness for φ_U.

Case 2 $x \in \Phi_U$. Let $\bar{\varphi}_U \in \Phi_U$ such that $x \in \bar{\varphi}_U$, and $w \in \Phi$ be a critical witness for $\bar{\varphi}_U$. Consider Fig. 13b.

Case 2(a) $w \models \varphi'_U$. As above, transitivity of $\models$ implies $w \models \varphi_U$ which is a contradiction to the assumption that w is a critical witness for $\bar{\varphi}_U$.

Case 2(b) $w \not\models \varphi'_U$. Then, according to (i), there is a $\bar{\varphi}'_U \in \Phi'_U$ such that $w \models \bar{\varphi}'_U$. Since $x \in \varphi'_U$ and $\Phi'_U \in$ iDNF, $\bar{\varphi}'_U$ does not contain the variable x; due to (iii), there must be a $\hat{\varphi}_U \in \Phi_U$ which is implied by $\bar{\varphi}'_U$ and does thus not contain the variable x and is hence different from $\bar{\varphi}_U$. Transitivity of $\models$ implies $w \models \hat{\varphi}_U$ which is a contradiction to the assumption that w is a critical witness for $\bar{\varphi}_U$.

This completes the proof for the direction MUB ⇒ LUB.

LUB ⇒ MUB. Assume Φ_U is no MUB for Φ. We need to show that whenever any of the three conditions in Definition 5 does not hold, then Φ_U is no LUB for Φ.

Case 1 If there is a clause in Φ which is not a witness of clauses in Φ_U, then $\Phi \not\models \Phi_U$ and thus Φ_U is no upper bound for Φ and in particular no least upper bound.

Case 2 Let x be a variable such that a clause $\varphi_U \in \Phi_U$ can be extended by x and the resulting formula is in iDNF and implied by Φ. It is clear that x does not occur in Φ_U, because otherwise it would not be possible to extend the formula without violating the iDNF property. Then, the assignment with $x \leftarrow$ false and all other variables $true$ is a model of Φ_U but no model of the extended formula Φ_U^{ext}. Thus, $\mathscr{M}(\Phi_U^{ext}) \subset \mathscr{M}(\Phi_U)$ and Φ_U is no LUB for Φ.

Case 3 Let $\varphi_U \in \Phi_U$ be a clause without a critical witness in Φ. Then, removing φ_U from Φ_U creates a formula $\bar{\Phi}_U$ with the following properties:

(i) Since Φ_U is an iDNF formula, removing φ_U creates a formula with strictly fewer models than Φ_U.
(ii) Since all witnesses of φ_U are non-critical, they imply other clauses in Φ_U as well. Hence $\bar{\Phi}_U$ is still implied by Φ.
$\bar{\Phi}_U$ is a better upper bound for Φ and Φ_U is no LUB.

A.3 Proof of Theorem 3

The proof of Theorem 3 comprises three parts: (i) Every formula returned by Algorithm 2 is an MUB, (ii) no two formulas returned by the algorithm are equivalent, and (iii) the delay between returning consecutive MUBs is polynomial in the size of the input formula. Properties (i) and (ii) are shown in Lemmata 4 and 5. Regarding (iii), let us consider the tree corresponding to the execution trace of the algorithm, where each iteration of the for-loop generates a new branch and each recursive call generates a child node; the leaves of the tree represent the MUBs discovered. The algorithm explores this tree in depth-first order. On each branch, the number of recursions and hence the height of the tree is bounded by the number n of clauses of the input formula. The formulas Φ_R, $\{\varphi\backslash\mathrm{vars}(\psi) \mid \varphi \in \Phi_R \wedge \varphi \neq \psi\}$, and $\Phi_U \cup \{\psi\}$ can be constructed in time at most quadratic in n (and linear in the max-arity of Φ). Hence, the time required to traverse the tree from the root to a leaf is bounded by $\mathscr{O}(n^3)$ and constitutes an upper bound for the delay between consecutive MUBs.

Lemma 4 *Each formula returned by Algorithm 2 is an iDNF minimal upper bound for the input formula Φ.*

Proof by induction. Base case: If PolyMub is called with a (possibly empty) clause Φ, then it returns Φ as its only MUB.

For the induction step, let ψ be a clause and Φ a DNF formula such that $\psi \vee \Phi$ is irreducible, and let $\Phi^{r(\psi)}$ be the formula obtained from Φ by removing all occurrences of variables of ψ, i.e. $\Phi^{r(\psi)} = \{\varphi\backslash\mathrm{vars}(\psi) \mid \varphi \in \Phi\}$; we prove the following property: If $\Phi_U^{r(\psi)}$ is an MUB for $\Phi^{r(\psi)}$, then $\varphi \vee \Phi_U^{r(\psi)}$ is an MUB for $\psi \vee \Phi$. This composition property is exactly the induction step showing that PolyMub constructs MUBs.

Let ψ, Φ, $\Phi^{r(\psi)}$ be as above, and let $\Phi_U^{r(\psi)}$ be an MUB for $\Phi^{r(\psi)}$. Then, $\Phi_U^{r(\psi)}$ is an iDNF formula and satisfies the *upper bound, maximality, and criticality* conditions from Definition 5 with respect to $\Phi^{r(\psi)}$. It remains to be shown that $\psi \vee \Phi_U^{r(\psi)}$ is an iDNF formula and satisfies those three conditions with respect to $\psi \vee \Phi$. First, for the iDNF property: Since $\Phi^{r(\psi)}$ does not contain variables from ψ, and since $\Phi_U^{r(\psi)}$ is an MUB for $\Phi^{r(\psi)}$, it follows that ψ and $\Phi_U^{r(\psi)}$ do not share variables, and thus $\psi \vee \Phi_U^{r(\psi)}$ is an iDNF formula.

Upper bound. We need to show that every clause in $\psi \vee \Phi$ implies a clause in $U = \psi \vee \Phi_U^{r(\psi)}$ (cf. Lemma 1). For ψ, this is evident. By transitivity of $\models$, and the fact that $\Phi^{r(\psi)} \models \Phi_U^{r(\psi)}$ by induction hypothesis, it suffices to show $\Phi \models \Phi^{r(\psi)}$. Let φ be a clause in Φ; then $\varphi\backslash\mathrm{vars}(\psi)$ is a clause in $\Phi^{r(\psi)}$ and is implied by φ (cf. Proposition 2).

Maximality. We show that every clause in $U = \psi \vee \Phi_U^{r(\psi)}$ is maximal. By induction hypothesis, $\Phi_U^{r(\psi)}$ is maximal with respect to $\Phi^{r(\psi)}$ and variables $\mathrm{vars}(\Phi^{r(\psi)})$; moreover, $\Phi_U^{r(\psi)}$ is also maximal with respect to Φ, since it cannot be extended by any of the remaining variables $\mathrm{vars}(\psi)$, since ψ is a clause in U and this extension would violate the iDNF property of U. ψ cannot be extended by a variable from $\mathrm{vars}(\Phi)$, as it would violate the upper bound property: Since $\Phi_U^{r(\psi)}$ does not contain any variable from $\mathrm{vars}(\psi)$, ψ only implies clause ψ in U. Any extension to ψ would invalidate this implication.

Criticality. It needs to be shown that every clause in $U = \psi \vee \Phi_U^{r(\psi)}$ has a critical witness in $\psi \vee \Phi$. Since $\psi \vee \Phi$ is irreducible, ψ is the only witness for $\psi \in U$. Now let φ be a clause in $\Phi_U^{r(\psi)}$; φ is not implied by ψ since they do not share any variables. We still need to show that φ has a critical witness in Φ. By induction hypothesis, φ has a critical witness $w \in \Phi^{r(\psi)}$; let $w^{e(\psi)} \in \Phi$ be the clause w extended by variables from $\mathrm{vars}(\psi)$. $w^{e(\psi)}$ is a critical witness for φ, since: (i) $w^{e(\psi)}$ cannot imply a different clause $\varphi' \in \Phi_U^{r(\psi)}$, since φ' does not contain variables from $\mathrm{vars}(\psi)$ and w does not imply φ'; (ii) $w^{e(\psi)}$ cannot imply ψ, since this would require $\psi \subset w^{e(\psi)}$ and hence $\psi \vee \Phi$ would not be irreducible which is a contradiction to our initial assumptions. □

Lemma 5 *Let Φ and Ψ be two formulas returned by Algorithm 2. Then $\Phi \not\equiv \Psi$.*

Proof Φ and Ψ are irreducible since they are iDNF formulas by Lemma 4. By construction, Φ and Ψ contain two distinct clauses φ and ψ that share a variable x. Since Φ and Ψ are iDNF formulas, we thus have $\varphi \in \Phi$, $\varphi \notin \Psi$, $\psi \in \Psi$, and $\psi \notin \Phi$; it then follows from Corollary 1 that $\Phi \not\equiv \Psi$. □

A.4 Proof of Lemma 3

Consider the point interval of each open leaf be $[x, x]$, where x is a distinct variable. The upper and lower bounds of T can be then expressed as functions f_U and f_L, respectively, of such variables. We show that for each such variable x, $\frac{\partial(f_U - f_L)}{\partial x} \leq 0$ and hence $f_U - f_L$ is maximized when x is minimized. That is, when $x = L$, where L is the lower bound of that open leaf.

We denote by f_U^n and f_L^n the lower and upper bound functions in variable x for a node at depth n. These functions are linear: $f_U^n = a_U^n \cdot x + b_U^n$ and $f_L^n = a_L^n \cdot x + b_L^n$.

Base case: Open leaf with variable x, depth n, and $f_U^n = f_L^n = x$. Then, $\frac{\partial(f_U^n - f_L^n)}{\partial x} = 1 - 1 = 0$.

Assume now the property holds at a node c at level $j+1$, and c is an ancestor of the open leaf with x or that open leaf. We show that the property also holds at the parent of c (and depth j).

Case 1 The parent of c is a $\oplus$ node: $\oplus(c_1, \ldots, c_k)$, where c is one of $c_1, \ldots, c_k$. Then,

$$f_U^j = f_U^{j+1} + \alpha_U = a_U^{j+1} \cdot x + b_U^{j+1} + \alpha_U$$
$$f_L^j = f_L^{j+1} + \alpha_L = a_L^{j+1} \cdot x + b_L^{j+1} + \alpha_L$$

where α_U and α_L represent the sum of the upper bounds, and lower bounds, respectively, of all the siblings of c. We then immediately have that $\frac{\partial(f_U^j - f_L^j)}{\partial x} = a_U^{j+1} - a_L^{j+1} \leq 0$.

Case 2 The parent of c is a $\odot$ node: $\odot(c_1, \dots, c_k)$, where c is one of $c_1, \dots, c_k$. Recall that we only consider restricted $\odot$ nodes, where at most one child is not a clause and can have different values for lower and upper bounds. If this child is c, let q be the product of the (exact) probabilities of all other children. Then, $a_U^j = a_U^{j+1} \cdot q$ and $a_L^j = a_L^{j+1} \cdot q$ and thus the inequality $a_U^j - a_L^j \leq 0$ is preserved.
Case 3 The parent of c is a $\otimes$ node: $\otimes(c_1, \dots, c_k)$, where c is one of $c_1, \dots, c_k$. Let

$$\alpha_L = \prod_{i=1, c_i \neq c}^{k} (1 - L(c_i)), \quad \alpha_U = \prod_{i=1, c_i \neq c}^{k} (1 - U(c_i))$$

where $L(c_i)$ and $U(c_i)$ represent the formulas for the lower and upper bounds, respectively, of node c_i. Given that $L(c_i) \leq U(c_i)$ for each node c_i, it holds that $\alpha_L \leq \alpha_U$. Then,

$$\begin{aligned} f_U^j &= 1 - \alpha_U \cdot (1 - f_U^{j+1}) \\ &= \alpha_U \cdot a_U^{j+1} \cdot x + 1 - \alpha_U + \alpha_U \cdot b_U^{j+1} \\ f_L^j &= 1 - \alpha_L \cdot (1 - f_L^{j+1}) \\ &= \alpha_L \cdot a_L^{j+1} \cdot x + 1 - \alpha_L + \alpha_L \cdot b_L^{j+1} \\ \frac{\partial(f_U^j - f_L^j)}{\partial x} &= \alpha_U \cdot a_U^{j+1} - \alpha_L \cdot a_L^{j+1} \leq 0. \end{aligned}$$

The latter inequality holds since $\alpha_U \leq \alpha_L$ (as discussed above) and $a_U^{j+1} \leq a_L^{j+1}$ (by hypothesis).

For relative approximation, we need to find x that maximizes $(1-\epsilon) \cdot U - (1+\epsilon) \cdot L$. This holds by a straightforward extension of the previous proof: The coefficient of x is shown to be greater in L than in U for $U - L$. Since $1 - \epsilon \leq 1 + \epsilon$, this property is preserved.

References

1. Abiteboul, S., Kimelfeld, B., Sagiv, Y., Senellart, P.: On the expressiveness of probabilistic XML models. VLDB J. **18**(5), 1041–1064 (2009)
2. Amsterdamer, Y., Deutch, D., Tannen, V.: Provenance for aggregate queries. In: PODS, pp. 153–164 (2011)
3. Antova, L., Jansen, T., Koch, C., Olteanu, D.: Fast and simple relational processing of uncertain data. In: ICDE, pp. 983–992 (2008)
4. Barcelo, P., Libkin, L., Romero, M.: Efficient approximations of conjunctive queries. In: PODS, pp. 249–260 (2012)
5. Birnbaum, E., Lozinskii, E.: The good old Davis-Putnam procedure helps counting models. J. AI Res. **10**(6), 457–477 (1999)
6. Brayton, R.K.: Factoring logic functions. IBM J. Res. Dev. **31**(2), 187 (1987)
7. Carlson, A., Betteridge, J., Kisiel, B., Settles, B., Hruschka, E.R. Jr., Mitchell, T.M.: Toward an architecture for never-ending language learning. In: AAAI (2010)
8. Cormode, G., Garofalakis, M., Haas, P., Jermaine, C.: Synopses for massive data: samples, histograms, wavelets, sketches. Found. Trends Databases **4**(1–3), 1–294 (2012)
9. Dagum, P., Karp, R.M., Luby, M., Ross, S.M.: An optimal algorithm for Monte Carlo Estimation. SIAM J. Comput. **29**(5), 1484–1496 (2000)
10. Dalvi, N., Schnaitter, K., Suciu, D.: Computing query probability with incidence algebras. In: PODS, pp. 203–214 (2010)
11. Dalvi, N., Suciu, D.: Efficient query evaluation on probabilistic databases. In: VLDB, pp. 864–875 (2004)
12. Dalvi, N., Suciu, D.: Efficient query evaluation on probabilistic databases. VLDB J. **16**(4), 523–544 (2007)
13. Darwiche, A., Marquis, P.: A knowlege compilation map. J. AI Res. **17**, 229–264 (2002)
14. Davis, M., Putnam, H.: A computing procedure for quantification theory. J. ACM **7**(3), 201–215 (1960)
15. Dylla, M., Miliaraki, I., Theobald, M.: Top-k query processing in probabilistic databases with non-materialized views. In: ICDE (2013, to appear)
16. Elbassioni, K., Makino, K., Rauf, I.: On the readability of monotone boolean formulae. In: COCOON, pp. 496–505 (2009)
17. Fink, R., Han, L., Olteanu, D.: Aggregation in probabilistic databases via knowledge compilation. PVLDB **5**(5), 490–501 (2012)
18. Fink, R., Hogue, A., Olteanu, D., Rath, S.: SPROUT^2: a squared query engine for uncertain web data. In: SIGMOD, pp. 1299–1302 (2011)
19. Fink, R., Olteanu, D.: On the optimal approximation of queries using tractable propositional languages. In: ICDT, pp. 174–185 (2011)
20. Fink, R., Olteanu, D., Rath, S.: Providing support for full relational algebra queries in probabilistic databases. In: ICDE, pp. 315–326 (2011)
21. Garey, M.R., Johnson, D.S.: Computers and Intractability; A Guide to the Theory of *NP*-Completeness. W.H. Freeman (1979)
22. Gatterbauer, W., Jha, A.K., Suciu, D.: Dissociation and propagation for efficient query evaluation over probabilistic databases. TR UW-CSE-10-04-01, U. Washington (2010)
23. Golumbic, M., Mintza, A., Rotics, U.: Read-once functions revisited and the readability number of a Boolean function. In: International Colloquium on Graph Theory, pp. 357–361 (2005)
24. Gomes, C.P., Sabharwal, A., Selman, B.: Handbook of satisfiability, Chapter. Model Counting. IOS Press (2009)
25. Grädel, E., Gurevich, Y., Hirsch, C.: The Complexity of query reliability. In: PODS, pp. 227–234 (1998)
26. Gupta, R., Sarawagi, S.: Creating probabilistic databases from information extraction models. In: VLDB 965–976 (2006)
27. Huang, J., Antova, L., Koch, C., Olteanu, D.: MayBMS: a probabilistic database management system. In: SIGMOD, pp. 1071–1074 (2009)
28. Imielinski, T., Lipski, W.: Incomplete information in relational databases. J. ACM **31**(4), 761–791 (1984)
29. Jampani, R., Xu, F., Wu, M., Perez, L.L., Jermaine, C.M., Haas, P.J.: MCDB: a Monte Carlo approach to managing uncertain data. In: SIGMOD, pp. 687–700 (2008)
30. Jha, A.K., Suciu, D.: Knowledge compilation meets database theory: compiling queries to decision diagrams. In: ICDT, pp. 162–173 (2011)
31. Johnson, D., Papadimitriou, C., Yannakakis, M.: On generating all maximal independent sets. Inf. Process. Lett. **27**(3), 119–123 (1988)
32. Kanagal, B., Li, J., Deshpande, A.: Sensitivity analysis and explanations for robust query evaluation in probabilistic databases. In: SIGMOD, pp. 841–852 (2011)

33. Karp, R.M., Luby, M., Madras, N.: Monte-Carlo approximation algorithms for enumeration problems. J. Algorithms **10**(3), 429–448 (1989)
34. Koch, C.: Approximating predicates and expressive queries on probabilistic databases. In: PODS, pp. 99–108 (2008)
35. Koch, C., Olteanu, D.: Conditioning probabilistic databases. PVLDB **1**(1), 313–325 (2008)
36. Li, J., Deshpande, A.: Consensus answers for queries over probabilistic databases. In: PODS, pp. 259–268 (2009)
37. Meinel, C., Theobald, T.: Algorithms and Data Structures in VLSI Design. Springer, Berlin (1998)
38. Olteanu, D., Huang, J.: Using OBDDs for efficient query evaluation on probabilistic databases. In: SUM, pp. 326–340 (2008)
39. Olteanu, D., Huang, J.: Secondary-storage confidence computation for conjunctive queries with inequalities. In: SIGMOD, pp. 389–402 (2009)
40. Olteanu, D., Huang, J., Koch, C.: SPROUT: Lazy vs. Eager query plans for tuple-independent probabilistic databases. In: ICDE, pp. 640–651 (2009)
41. Olteanu, D., Huang, J., Koch, C.: Approximate confidence computation for probabilistic databases. In: ICDE, pp. 145–156 (2010)
42. Olteanu, D., Koch, C., Antova, L.: World-set decompositions: expressiveness and efficient algorithms. Theor. Comput. Sci. **403**(2–3), 265–284 (2008)
43. Olteanu, D., Wen, H.: Ranking query answers in probabilistic databases: complexity and efficient algorithms. In: ICDE, pp. 282–293 (2012)
44. Pe'er, J., Pinter, R.Y.: Minimal decomposition of boolean functions using non-repeating literal trees. In: IFIP Workshop on Logic and Architecture Synthesis (1995)
45. Provan, J.S., Ball, M.O.: The complexity of counting cuts and of computing the probability that a graph is connected. SIAM J. Comput. **12**(4), 777–788 (1983)
46. Ré, C., Dalvi, N., Suciu, D.: Efficient top-k query evaluation on probabilistic data. In: ICDE, pp. 886–895 (2007)
47. Ré, C., Suciu, D.: Approximate lineage for probabilistic databases. PVLDB **1**(1), 797–808 (2008)
48. Ré, C., Suciu, D.: The trichotomy of having queries on a probabilistic database. VLDB J. **18**(5), 1091–1116 (2009)
49. Sagiv, Y., Yannakakis, M.: Equivalences among relational expressions with the union and difference operators. J. ACM **27**(4), 633–655 (1980)
50. Selman, B.: Knowledge compilation and theory approximation. J. ACM **43**(2), 193–224 (1996)
51. Sen, P., Deshpande, A., Getoor, L.: Read-once functions and query evaluation in probabilistic databases. PVLDB **3**(1), 1068–1079 (2010)
52. Souihli, A., Senellart, P.: Optimizing approximations of DNF query lineage in probabilistic XML. In: ICDE (2013, to appear)
53. Suciu, D., Olteanu, D., Ré, C., Koch, C.: Probabilistic databases. Morgan & Claypool Publishers (2011)
54. Trevisan, L.: A note on deterministic approximate counting for k-DNF. In: APPROX-RANDOM, pp. 417–426 (2004)
55. Tsukiyama, S., Ide, M., Ariyoshi, H., Shirakawa, I.: A new algorithm for generating all the maximal independent sets. SIAM J. Comput. **6**(3), 505–517 (1977)
56. Vadhan, S.: The complexity of counting in sparse, regular, and planar graphs. SIAM J. Comput. **32**(2), 398–427 (2001)
57. Vazirani, V.V.: Approximation Algorithms. Springer, Berlin (2001)
58. Wang, T.Y., Ré, C., Suciu, D.: Implementing not exists predicates over a probabilistic database. In: QDB/MUD, pp. 73–86 (2008)
59. Wei, W., Selman, B.: A new approach to model counting. In: SAT, pp. 324–339 (2005)
60. Zachary, W.W.: An information flow model for conflict and fission in small groups. J. Anthropol. Res. **33**, 452–473 (1977)