Approximate Confidence Computation in Probabilistic Databases

Dan Olteanu[1], Jiewen Huang[1], and Christoph Koch[2]

[1]*Oxford University Computing Laboratory, Oxford, OX1 3QD, UK*
[2]*Department of Computer Science, Cornell University, Ithaca, NY 14853, USA*

***Abstract*— This paper introduces a deterministic approximation algorithm with error guarantees for computing the probability of propositional formulas over discrete random variables. The algorithm is based on an incremental compilation of formulas into decision diagrams using three types of decompositions: Shannon expansion, independence partitioning, and product factorization. With each decomposition step, lower and upper bounds on the probability of the partially compiled formula can be quickly computed and checked against the allowed error.**

This algorithm can be effectively used to compute approximate confidence values of answer tuples to positive relational algebra queries on general probabilistic databases (c-tables with discrete probability distributions). We further tune our algorithm so as to capture all known tractable conjunctive queries without self-joins on tuple-independent probabilistic databases: In this case, the algorithm requires time polynomial in the input size even for *exact* computation.

We implemented the algorithm as an extension of the SPROUT query engine. An extensive experimental effort shows that it consistently outperforms state-of-art approximation techniques by several orders of magnitude.

I. Introduction

This paper investigates the following problem: Given a propositional formula Φ in disjunctive normal form (DNF) over independent discrete random variables and an allowed error ϵ, compute a probability value that is within ϵ from the exact probability of Φ. Computing the exact probability of Φ is a generalization of counting the number of its satisfying assignments [12], [7] and is #P-hard for DNFs [25]. Note that #P is an intractable complexity class which contains NP. Our motivation for this study is that probability computation is a core task in probabilistic databases, e.g., [7], [21]. This study may be however of interest to model counting (#SAT) and probabilistic inference in graphical models as well.

Approaches to exact probability computation essentially explore the raw combinatorial search space of the problem and therefore do not scale up to larger problem sizes. Approximate methods, on the other hand, are much faster, though at the price of losing accuracy. They are nevertheless extremely useful in applications where estimates suffice and tiny distinctions are irrelavant. Two fundamental aspects govern such methods [11]: the *estimate quality* and the *correctness confidence* on the reported estimate. It is easy to find a correct lower bound for the true probability p of Φ - just consider the probability p_0 of any of its clauses. However, the quality can be very poor, for p can be orders of magnitude higher than p_0. Also, we may report an estimate much closer to p, but be completely unable to provide any correctness confidence.

In this paper we present a deterministic approximation algorithm for probability computation. The algorithm guarantees both estimate quality and correctness confidence bounds by computing an interval of probabilities that are within the given error from the exact probability of the input DNF. It incrementally performs a sequence of decomposition steps and termination checks until the desired approximation is achieved. In a decomposition step, a DNF Φ is compiled into an equivalent disjunction or conjunction of DNFs $\phi_1, \ldots, \phi_n$ such that (1) they are pairwise independent or mutual exclusive, and (2) lower and upper bounds on the probability of Φ can be easily computed from the bounds on the probabilities of $\phi_1, \ldots, \phi_n$. We consider three types of decompositions: Shannon expansion, independence partitioning, and product factorization. A termination check computes bounds on the probabilities of $\phi_1, \ldots, \phi_n$ and of Φ - we deliberately chose that the bounds computed at this step may fall short of the estimate quality desideratum, yet be quickly computable. If the bounds are however close enough to guarantee the desired approximation, then the algorithm stops. Otherwise, we further decompose and check again for termination.

There are two main observations behind the design of this algorithm. First, sufficient approximations can be obtained within a few decomposition steps and there is thus no need to exhaustively compile the input DNF down to clauses. This motivates the incremental nature of the algorithm as well as the use of efficient termination checks. Being incremental, the algorithm is also useful under a given time budget. According to our experiments with large probabilistic data sets, a small number of well-chosen decomposition steps computable within a few seconds are usually enough to guarantee good precision. The DNFs obtained after decompositions may still be large, yet they account for a very small percentage of the overall probability mass. The second observation is geared at query evaluation in probabilistic databases. We can effectively derive orders of decomposition steps under which any approximation can be obtained in polynomially many steps for decomposing DNFs obtained during the evaluation of any known tractable conjunctive query without self-joins. Most notably, this is achieved *without a-priori knowledge of the query or the input probabilistic database*. These two aspects are, to the best of our knowledge, not considered by any other query evaluation technique nor probability approximation algorithm.

The main contributions of this paper are as follows:

- We introduce a deterministic algorithm with error guarantees for computing probabilities of DNFs, such as those created by the evaluation of positive relational algebra queries on probabilistic databases. In contrast to much of the existing work in probabilistic databases, this algorithm is not only applicable to restricted classes of queries or probabilistic databases, but is generic.
- The algorithm is based on a number of fundamental ideas from combinatorial algorithms, constraint satisfaction, and verification, and turns out to be both simple and extensible. We compile DNFs into a novel type of decision diagram called *d-trees*. Such diagrams decompose DNFs using negative correlations, independence, and factored representations that are easy to compute. Given a d-tree and an approximate (or exact) probability for each of its leaves, we can compute an overall approximate (or exact) probability in just one pass over the d-tree.
- We then show how a given formula can be *incrementally* compiled into fragments of a d-tree *without fully materializing it*. We devise heuristics that allow us to obtain close lower and upper probability bounds within a few compilation steps, thus avoiding exhaustive traversal of a complete d-tree. For a given absolute or relative error bound, we decide locally whether to further compile a subformula under a certain node of the d-tree or move on to a following node. For this, we devise a safety check on which such subformulas can be discarded while still guaranteeing the overall error bound.
- We also show that d-trees in conjunction with our heuristics yield an alternative polynomial-time algorithm for exact confidence computation for cases from the literature for which efficient algorithms for confidence computation are known, namely the hierarchical queries without self-joins [7], with inequalities [20], and certain additional cases in which functional dependencies on the data yield tractability [21]. In fact, these are all the currently known tractable cases in the absence of self-joins. In these cases, our algorithm guarantees a running time linear in query size and quadratic in the size of the input DNF.
- We have implemented the algorithm as a new operator in the SPROUT query engine, which is used by the MayBMS probabilistic database management system.
- We experimentally verify the robustness of our algorithm. We evaluate both tractable and hard queries on various probabilistic databases, such as tuple-independent TPC-H, random graphs, and social networks. In all these experiments, our algorithm consistently outperforms state-of-the-art approximation algorithms by orders of magnitude.
- The experiments also show that our algorithm performs well in practice compared to approaches specialized to tractable queries, which exploit knowledge about the query but are *only* applicable to those tractable queries.

To summarize, this single algorithm is competitive with the most efficient currently known exact and approximation algorithms in their respective domains.

II. State of the art

A very recent survey on approximate and exact techniques for model counting is presented in [11]. 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., [27]. Further, extensions of the Davis-Putnam procedure (which is based on Shannon expansion) have been used for counting the solutions to formulae [4]. The decomposable Negation Normal Form [8] (and variations thereof) is a propositional theory with efficient model counting, which uses Shannon expansion and independence partitioning. Our recent work [17] uses similar ideas to design an exact probability computation algorithm, although without polynomial-time guarantees for tractable queries.

The approach in this paper shares ideas with these techniques, yet two of its main aspects remain novel: (1) the combination of incremental compilation and the use of probability bounds for fast approximate computation with error guarantees, and (2) polynomial-time evaluation for tractable queries on probabilistic databases.

A different line of research is on randomized approximation algorithms. It was first shown in work by Karp, Luby, and Madras [15] that there is a fully polynomial-time randomized 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], [7], [23], [16].

The techniques based on [15] 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 (= 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 $\Pr\left[|p - \hat{p}| \geq \epsilon \cdot p\right] \leq \delta$.

The work by Karp, Luby, and Madras has started a line of research to derandomize these approximation techniques, eventually leading to a polynomial time deterministic $(\epsilon, 0)$-approximation algorithm [24] (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 (above 2^{50} for 3-DNF) and the algorithm is not practical. This is in contrast to observations that the Karp-Luby Monte Carlo algorithm is practical (e.g. [1], [23], and the experiments of the present paper). In fact, it is the state-of-the-art (and only) approximation algorithm used in current probabilistic database management systems such as MystiQ [23] and MayBMS [2].

III. Preliminaries

We denote the domain of a random variable x by Dom_x. *Atomic events* (or *atomic formulae*) are of the form $x = a$ where x is a random variable and $a \in \mathrm{Dom}_x$ is one of its domain values. Random variables with domain $\{\mathrm{true}, \mathrm{false}\}$

are called Boolean and we will write x and $\neg x$ as shortcuts for the atomic events $x =$ true and $x =$ false, respectively.

We define finite probability distributions via a set of *independent* random variables with finite domains. Such a probability distribution is completely specified by a function P that assigns a number $P(x = a) \in (0, 1]$ to each atomic event $x = a$ such that, for each random variable x,

$$\sum_{a \in \mathrm{Dom}_x} P(x = a) = 1.$$

A (positive propositional) *formula* (or *event*) is constructed from atomic events using the binary operations $\vee$ (logical "or") and $\wedge$ (logical "and"). A conjunction of atomic events $(x_1 = a_1) \wedge \cdots \wedge (x_n = a_n)$ is called a clause. A *DNF formula* is a disjunction of clauses.

A *valuation* of the random variables is an assignment of *each* of the random variables to one of its domain values. We can identify *possible worlds* with valuations, or equivalently, with clauses that contain exactly one atomic event for *each* of the random variables. A formula is *consistent* (satisfiable) if there is at least one valuation of the random variables that makes the formula true. For clauses, consistency is easy to check: a clause is consistent iff it does not contain two atomic formulae $x = a$ and $x = b$ where $a \neq b$. We will treat clauses like sets of atomic formulae in that we will always assume the absence of duplicate atoms.

We denote the set of valuations on which a formula ϕ is true by $\omega(\phi)$. The formulae ϕ and ψ are *equivalent* iff $\omega(\phi) = \omega(\psi)$. We call two formulae ϕ and ψ *independent* if there is no random variable that occurs in both ϕ and ψ.

Because of the independence of the random variables, the probability of a consistent clause $(x_1 = a_1) \wedge \cdots \wedge (x_n = a_n)$ is $\prod_{i=1}^{n} P(x_i = a_i)$; if $n = 0$ then it is 1. The probability of a formula ϕ is the sum of the probabilities of all distinct valuations of the random variables (rendered as clauses as discussed above) on which ϕ is true, i.e.,

$$P(\phi) = \sum_{\psi \in \omega(\phi)} P(\psi).$$

The goal of this paper is to develop an efficient algorithm for computing the (possibly approximate) probability of a DNF.

IV. Compiling DNFs into D-Trees

Computing the probability of a formula is #P-hard. In general, there is no efficient way of computing the probability $P(\phi \wedge \psi)$ or $P(\phi \vee \psi)$ from $P(\phi)$ and $P(\psi)$. However, there are important special cases in which this is feasible, in particular,

- if ϕ and ψ are independent, then

$$\begin{aligned} P(\phi \wedge \psi) &= P(\phi) \cdot P(\psi) \\ P(\phi \vee \psi) &= 1 - (1 - P(\phi)) \cdot (1 - P(\psi)) \end{aligned}$$

- if ϕ and ψ are inconsistent with each other (i.e., there is no valuation of the random variables on which both are true: the disjunction is *exclusive*), then

$$P(\phi \vee \psi) = P(\phi) + P(\psi).$$

We will use explicit notation to mark such $\wedge$ and $\vee$-operations: We will use $\otimes$ for independent-or, $\odot$ for independent-and and $\oplus$ for exclusive-or (that is, "or" of inconsistent formulae, as just introduced).

Example 4.1: 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 formula $(x \otimes y) \odot ((z \odot u) \oplus (\neg z \odot v))$. The probability of this formula thus is $\big(1 - (1 - P(x)) \cdot (1 - P(y))\big) \cdot \big(P(z) \cdot P(u) + P(\neg z) \cdot P(v)\big)$. □

For convenience, we also use the Boolean combinators on sets of formulae; i.e., we write $\bigwedge \Phi$ for $\phi_1 \wedge \cdots \wedge \phi_n$ if $\Phi = \{\phi_1, \ldots, \phi_n\}$ and analogously $\bigvee \Phi$, $\bigotimes \Phi$, $\bigoplus \Phi$, and $\bigodot \Phi$. (All the operations are associative, and computing the probabilities of formulae using these set operations is straightforward.)

Definition 4.2: A (partial) *d-tree* (for decomposition tree) is a formula constructed from $\otimes$, $\oplus$, $\odot$ and nonempty DNFs (as "leaves"). A d-tree in which each DNF is a singleton – i.e., contains a single clause – is called a *complete d-tree*. Given a partial d-tree, the d-tree obtained by replacing a leaf DNF by an equivalent partial d-tree is called a *refinement*. □

Thus, in a d-tree (viewed as a parse tree of the d-tree formula), an $\wedge$ or $\vee$ node never occurs above a $\oplus$, $\otimes$, or $\odot$ node.

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

Proposition 4.3: Given the probabilities of all the DNF leaves of a partial d-tree, its probability can be computed in linear time (assuming unit-cost arithmetics). □

Since computing the probability of a leaf requires just a table lookup, the probability of a complete d-tree can be computed in time linear in its size.

Next we present an algorithm for computing a complete (or, if we stop the compilation early, a partial) d-tree from a DNF. For this purpose, we assume a DNF is represented by a set of sets of atomic formulae. In essence, the algorithm repeatedly applies three decomposition methods that correspond to the three types of inner nodes in a d-tree $\oplus$, $\otimes$, and $\odot$:

- Independent-or $\otimes$: Partition Φ into independent DNFs $\Phi_1, \Phi_2 \subset \Phi$ such that Φ is equivalent to $\Phi_1 \vee \Phi_2$.
- Independent-and $\odot$: Partition Φ into independent DNFs $\Phi_1, \Phi_2 \subset \Phi$ such that Φ is equivalent to $\Phi_1 \wedge \Phi_2$.
- Exclusive-or $\oplus$: Choose a variable x in Φ. Replace Φ by

$$\bigoplus_{a \in \mathrm{Dom}_x, \Phi|_{x=a} \neq \emptyset} \big(\{\{x = a\}\} \odot \Phi\,|_{x=a}\big)$$

  where the DNF $\Phi\,|_{x=a}$ is obtained from Φ by removing all clauses $\phi \in \Phi$ for which $\phi \wedge (x = a)$ is inconsistent and (syntactically) removing the atomic formula $x = a$ from the remaining clauses in which it occurs. Obviously, $(x = a) \wedge \Phi$ is equivalent to $(x = a) \wedge \Phi\,|_{x=a}$. This decomposition is called Shannon expansion.

Figure 1 sketches our general compilation approach, which will be refined in the next sections. Here, we consider that the compilation is exhaustive, i.e., the leaves of the d-tree only hold DNFs that are singleton clauses. If approximate

Compile (DNF Φ with $\Phi \neq \emptyset$) returns d-tree

if $(\emptyset \in \Phi)$ **then return** $\{\emptyset\}$

1. remove all subsumed clauses Φ :

foreach $s, t \in \Phi$ **such that** $s \neq t$ **do**

if $(s \subset t)$ **then** $\Phi := \Phi - \{t\}$

2. apply independent-or:

if there are non-empty and pairwise indep. DNFs $\Phi_1, \ldots, \Phi_{|I|}$ **such that** $\Phi = \Phi_1 \cup \ldots \cup \Phi_{|I|}$

then return $\bigotimes_{i \in I} \big(\text{Compile}(\Phi_i)\big)$

3. apply independent-and:

if there are non-empty and pairwise indep. DNFs $\Phi_1, \ldots, \Phi_{|I|}$

such that Φ is equivalent to $\Phi_1 \wedge \ldots \wedge \Phi_{|I|}$

then return $\bigodot_{i \in I} \big(\text{Compile}(\Phi_i)\big)$

4. apply Shannon expansion:

choose a variable x in Φ;

$T := \{\phi \mid \phi \in \Phi, \nexists a \in \text{Dom}_x : (x = a) \subseteq \phi\}$;

$\forall a \in \text{Dom}_x : \Phi\,|_{x=a} := \big\{\{y_1 = b_1, \ldots, y_m = b_m\} \mid \{x = a, y_1 = b_1, \ldots, y_m = b_m\} \in \Phi\big\} \cup T$;

return $\bigoplus_{a \in \text{Dom}_x, \Phi|_{x=a} \neq \emptyset} \big(\{\{x = a\}\} \odot \text{Compile}(\Phi\,|_{x=a})\big)$

Fig. 1. Compiling DNFs into d-trees.

probabilities are sought for, however, the compilation need not be exhaustive and the leaves can hold larger DNFs.

Example 4.4: Figure 2 shows a DNF and the complete d-tree obtained by executing the algorithm of Figure 1 to completion. □

This algorithm is correct:

Proposition 4.5: Any DNF Φ is equivalent to Compile(Φ).

All three decompositions can be done efficiently. Shannon expansion requires linear time for each subformula. The independent-or partitioning is finding connected components in the dependency graph of the input DNF Φ, which consists of a node for each variable of Φ and, for each clause $\bigwedge_{i=1}^{n} x_i = a_i$ of Φ, of the edges (x_i, x_{i+1}) for $1 \leq i < n$. This can be done in time linear in the size of the Φ (using a well-known depth-first algorithm for computing strongly connected components, Tarjan's algorithm). The independent-and partitioning is a special algebraic factorization of DNFs [5]. For relational encodings of DNFs, as obtained by query evaluation on probabilistic databases [2], this factorization is unique and requires time $O(m \cdot n \cdot \log n)$, where n and m are the sizes of the DNF and of the constituent clauses, respectively [22].

The order of the variable choices in Shannon expansion (a.k.a. variable elimination in the Davis-Putnam SAT solving algorithm [9]) greatly influences the size of the d-tree. In general, the compilation of a DNF creates a d-tree of exponential size, and it is important to find compilation strategies that lead to d-trees of small sizes [8], [17]. Section VI-B gives a strategy that applies to DNFs of tractable queries. If that strategy fails, we choose a variable that occurs most frequently in the DNF.

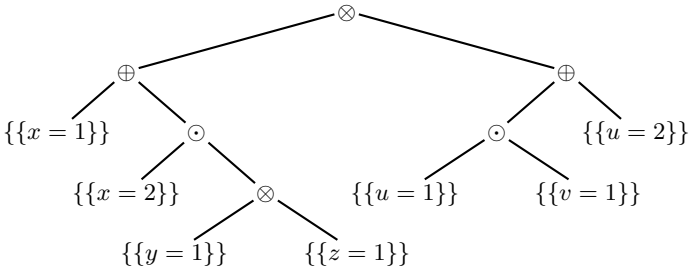


Fig. 2. D-tree of DNF $\Phi = \{\{x = 1\}, \{x = 2, y = 1\}, \{x = 2, z = 1\}, \{u = 1, v = 1\}, \{u = 2\}\}$.

Remark 4.6: D-trees are a generalization of the ws-trees of [17]: We have added independent-and decompositions, which are crucial for application of d-trees to tractable queries. Also, we have generalized the formalism to partial decompositions, which are the foundation of the approximation techniques of Section V. The and/xor trees of [18] are modeled on the ws-trees but are a weaker representation system in that they have tuples, rather than clauses, at their leaves. Complete d-trees with inner nodes $\odot$ and $\otimes$ only capture read-once functions [10] or formulas in one-occurrence forms [19]. □

V. Approximate Probability Computation

As discussed in Section IV, the exact probability of a DNF can be easily computed following the DNF compilation into a complete d-tree. Such an exhaustive compilation is not practical in general. If an approximate probability suffices, then we may only explore a few levels in a d-tree and approximate the probability at its leaves using efficient heuristics.

The key challenge addressed in this section is the design (i) of an efficient and good heuristic for approximating the probability of DNFs at the leaves of d-trees, and (ii) of an efficient algorithm that can compute an approximate probability for a given DNF by incrementally refining its d-tree compiled form.

A. Lower and Upper Probability Bounds for DNFs

We next discuss how to quickly compute lower and upper bounds of the probabilities of DNFs at the leaves of a d-tree without refining them. Figure 3 gives one heuristic that partitions the input DNF Φ into a set of buckets such that the exact probability of each bucket can be computed efficiently.

The lower and upper bounds of the exact probability of Φ are computed as the maximum over the probabilities of the buckets, and the sum of probabilities of the buckets, respectively. Both bounds are correct: Assume that B_i is a bucket with the maximal probability. Since Φ is a set of clauses, $\Phi = B_i \vee \Phi'$. Since each clause in Φ has a non-null probability by definition, $P(B_i) \leq P(B_i \vee \Phi') = P(\Phi)$, and thus $P(B_i)$ is indeed a lower bound for $P(\Phi)$. To see why the

Independent (DNF Φ with $\Phi \neq \emptyset$) returns [Lower, Upper]
  minimally partition Φ **into** $B_1 \vee \ldots \vee B_n$ **such that**
    $\forall 1 \leq i \leq n, \forall d, d' \in B_i : d, d'$ are independent;
  foreach bucket B_i **do**
    $P(B_i) := 0$;
    foreach clause $d \in B_i$ **do**
      $P(B_i) := 1 - (1 - P(B_i)) \cdot (1 - P(d))$;
  return $[\max_{i=1}^{n} P(B_i), \min(1, \sum_{i=1}^{n} P(B_i))]$;

Fig. 3. Computing lower and upper bounds for the probability of DNFs.

sum of probabilities of the buckets is indeed an upper bound, consider the following cases. If the buckets are negatively correlated, then the probability of their disjunction is the sum of their probabilities. In case they are independent or positively correlated, then it follows by definition that the probability of their disjunction is at most the sum of their probabilities.

Proposition 5.1: Let $[L, U] =$ Independent(Φ) for a DNF Φ. It then holds that $L \leq P(\Phi) \leq U$. □

Let us now look closer at how the buckets are created. Each bucket only contains pairwise independent clauses, and each such bucket is maximal, i.e., for a given bucket B there is no clause in Φ and not in B that is pairwise independent with each clause in B. The probability of each bucket can be computed efficiently, as shown in Figure 3. As there may be several possible minimal partitionings of Φ, we empirically noticed that the lower bound computed by this heuristic can be further improved by first sorting Φ descending on the marginal probability of its clauses, and then constructing a bucket that contains the most probable clause and subsequent independent clauses. It turns out that this heuristic behaves very well for all of our experimental scenarios (see Section VII). This heuristic requires time quadratic in the size of the input DNF, the most expensive part being the minimal partitioning.

Example 5.2: Let the DNF $\Phi = c_1 \vee c_2 \vee c_3$, where

$$c_1 = (x \wedge y), c_2 = (x \wedge z), c_3 = v$$

and $P(x) = 0.3$, $P(y) = 0.2$, $P(z) = 0.7$, $P(v) = 0.8$. One minimal partitioning of Φ is $B_1 = c_1 \vee c_3$ and $B_2 = c_2$. Then, $P(B_1) = 1 - (1 - 0.06) \cdot (1 - 0.8) = 0.812, P(B_2) = 0.21$. The bounds are $L(\Phi) = P(B_1) = 0.812$ and $U(\Phi) = \min(1, 0.821 + 0.21) = 1$. Another minimal partitioning can be obtained by first sorting the clauses descending on their marginal probabilities. Then, $B_1 = c_2 \vee c_3$, $B_2 = c_1$, and $P(B_1) = 1 - (1 - 0.21) \cdot (1 - 0.8) = 0.842, P(B_2) = 0.06$. The new bounds are $L(\Phi) = 0.842$ and $U(\Phi) = 0.848$, which approximates better the exact probability of 0.8456. □

Remark 5.3: Finding fast heuristics that better approximate the bounds of DNFs will be future work. A natural extension to our heuristic is to allow positively correlated clauses in the same bucket such that the DNF of each bucket can be factored into one occurrence form, where each variable occurs only once. For instance, Φ of Example 5.2 can be factored as $x \wedge (y \vee z) \vee v$, in which case the whole Φ is allocated to the first bucket. The probability of such factored forms can be computed in linear time [19]. □

B. Lower and Upper Probability Bounds for D-trees

The lower and upper bounds can be propagated from leaves to the root of the d-tree. For this, we make use of the observation that the formulas for probability computation of each decomposition type are monotonically increasing. (A function is monotonically increasing if for all x and y such that $x \leq y$, it holds that $f(x) \leq f(y)$.) If some of the children of an inner node ($\otimes$, $\odot$, or $\oplus$) have smaller (larger) probabilities, then it immediately follows that the probability at that node becomes smaller (larger).

Given bounds at the children, the lower and upper bounds at the parent node are obtained by replacing in the formulas for computing the probability of nodes $\oplus$, $\otimes$, and $\odot$, the exact probability of the children with their lower and upper bounds, respectively. We are now ready to generalize the result of Proposition 5.1 from DNFs to d-trees.

Proposition 5.4: If a d-tree d for a DNF Φ has bounds $[L, U]$, then it holds that $L \leq P(\Phi) \leq U$. □

Example 5.5: Consider the partial d-tree of Figure 4, where the leaves are annotated with lower and upper bounds. Then, the lower and upper bounds $[L, U]$ of the d-tree can be computed as follows (denote $x = 1$ by Φ_4):

$$\begin{aligned} L &= L(\Phi_1) \otimes [(L(\Phi_4) \odot L(\Phi_2)) \oplus L(\Phi_3)] \\ &= 1 - (1 - 0.1) \cdot (1 - (0.5 \cdot 0.4 + 0.35)) = 0.595. \\ U &= U(\Phi_1) \otimes [(U(\Phi_4) \odot U(\Phi_2)) \oplus U(\Phi_3)] \\ &= 1 - (1 - 0.11) \cdot (1 - (0.5 \cdot 0.44 + 0.38)) = 0.644. \end{aligned}$$

□

Remark 5.6: Due to our heuristic to approximate bounds on DNFs, refinement of a d-tree might not always lead to tighter bounds. However, it eventually leads to complete d-trees and hence to termination of the compilation procedure. □

C. Absolute and Relative Approximation Errors

We consider here two types of approximations, given a fixed error factor ϵ ($0 \leq \epsilon < 1$).

Definition 5.7: 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 $(1 - \epsilon) \cdot p \leq \hat{p} \leq (1 + \epsilon) \cdot p$. □

Given a d-tree for a DNF Φ, its bounds $[L, U]$ may contain several ϵ-approximations of $P(\Phi)$, although not every value between these bounds is an ϵ-approximation. The connection between the bounds of a d-tree for Φ and ϵ-approximations of $P(\Phi)$ is given by the following proposition.

Proposition 5.8: Given a DNF Φ, a fixed error ϵ, and a d-tree for Φ with bounds $[L, U]$.

- If $U - \epsilon \leq L + \epsilon$, then any value in $[U - \epsilon, L + \epsilon]$ is an absolute ϵ-approximation of $P(\Phi)$.
- 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 $P(\Phi)$. □

In the sequel, we call a d-tree for a DNF Φ an (absolute or relative) ϵ-approximation of Φ if its bounds satisfy the above sufficient condition. Written differently, the condition becomes

$$U - L \leq 2 \cdot \epsilon \text{ in the absolute case, and}$$
$$(1-\epsilon) \cdot U - (1+\epsilon) \cdot L \leq 0 \text{ in the relative case.}$$

This condition can be checked in linear time in the size of the d-tree: We need one pass over the d-tree to compute its lower and upper bounds, and then check the above condition, which only involves the bounds and the fixed error.

Example 5.9: Recall the DNF Φ from Example 5.2. Its exact probability is $p = 0.8456$. With bounds [0.842,0.848], we obtain precisely one absolute 0.003-approximation $\hat{p} = 0.845$, because $0.848 - 0.003 = 0.842 + 0.003$.

With the same bounds, any value in [0.844,0.846] is an absolute 0.004-approximation, because 0.848 - 0.004 = 0.844 and 0.842 + 0.004 = 0.846. □

D. An Incremental and Memory-Efficient Algorithm

Proposition 5.8 can be effectively used for approximate probability computation as follows. While compiling a DNF into a d-tree, we can ask before the construction of each node of the d-tree whether the sufficient condition on the approximation is reached. If this is the case, then we can stop the compilation and output the interval of ϵ-approximations. If this is not the case, then we continue with the compilation and choose the leaf with the largest bounds interval and further refine it. This already gives us an incremental algorithm for computing ϵ-approximations.

The algorithm sketched above needs to keep every node it creates in main memory. This is infeasible for large inputs. We therefore consider next the practical question of whether the sufficient condition for ϵ-approximation can still be fulfilled after subsequent refinement even if some leaves are not refined anymore. In the sequel, we call such leaves *closed*; an *open* leaf may be further refined to completion.

The challenge we need to address is to derive an ϵ-approximation condition in the presence of closed leaves. Based on this, we can incrementally compile the input DNF into a d-tree in depth-first left-to-right traversal, and decide *locally* whether the current leaf under exploration can be closed or must be refined further. When a leaf is closed, its bounds are used to update a pair of aggregated bounds of all the leaves already closed, and the leaf is released. This gives us a very efficient algorithm that need only keep in memory the current root-to-leaf path under construction and some local information at each node along this path.

In the sequel, we consider d-trees, where at most one child of each $\odot$ node may be closed without being complete. This does not restrict our encoding of variable elimination as given in Figure 1, since the $\odot$ nodes needed there are binary and one of their children is always a clause, i.e., it is complete, and for which the exact probability is known.

To understand the worst-case scenario in case we want to close a leaf in a d-tree d, we need to compute the largest bounds interval of d for any possible probability each open leaf may take. If these bounds fail to satisfy the condition for an ϵ-approximation, then we may not reach such an approximation by refinements that complete the open leaves. In this case, we must not close that leaf. We elaborate on this next.

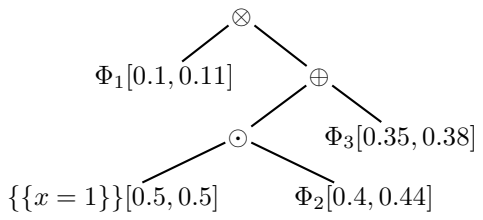


Fig. 4. D-tree. Leaves: Φ_1 is closed, Φ_2 is current, Φ_3 is open.

Definition 5.10: The *bound space* of a d-tree d is the set of possible bounds $[L, U]$ of d obtained by choosing for each open leaf any point interval between the bounds of that leaf.□

Let us denote by $L(d)$ the element of the bound space obtained by choosing for each open leaf the point interval $[L_i, L_i]$, where L_i is a lower bound for that leaf.

Lemma 5.11: For a d-tree d, $L(d)$ is the pair of bounds $[L, U]$ that maximizes each of $U - L$ and $(1-\epsilon)\cdot U - (1+\epsilon)\cdot L$ over the entire bound space of d. □

Proof: Consider the point interval of each open leaf be $[x_i, x_i]$, where x_i is a distinct variable. The upper and lower bounds of d can be then expressed as functions f_U and f_L, respectively, of such variables. We show that for each such variable x, $\frac{\delta(f_U - f_L)}{\delta x} \leq 0$ 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.

Base case: We are at the open leaf with variable x. Let us denote by n the level of this leaf. We have $f_U^n = a_U^n \cdot x + b_U^n$ and $f_L^n = a_L^n \cdot x + b_L^n$, where $a_U^n = a_L^n = 1$ and $b_U^n = b_L^n = 0$. It then holds that

$$\frac{\delta(f_U^n - f_L^n)}{\delta x} = a_U^n - a_L^n \leq 0.$$

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

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

$$\frac{\delta(f_U^j - f_L^j)}{\delta x} = a_U^{j+1} - a_L^{j+1} \leq 0.$$

Case 2: The parent of c is a $\odot$ node: $\odot(c_1, \ldots, c_k)$, where c is one of $c_1, \ldots, c_k$. 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: $\oplus(c_1, \ldots, c_k)$, where c is one of $c_1, \ldots, c_k$. 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,

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

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. ■

Lemma 5.11 gives us the necessary strategy to decide whether closing leaves in a d-tree still allows to obtain an ϵ-approximation. Finding the maximal values of $U - L$ and $(1-\epsilon) \cdot U - (1+\epsilon) \cdot L$ can be done very efficiently by computing $L(d)$ in just one scan of d. Our main result concerning the closing of leaves follows then from Lemma 5.11 and the fact that refinement eventually leads to completion of d.

Theorem 5.12: Given a d-tree d for a DNF Φ, and a fixed error ϵ. If the bounds $L(d)$ satisfy the sufficient condition for an ϵ-approximation in Proposition 5.8, then there is a refinement of d that is an ϵ-approximation of Φ. □

Example 5.13: Consider the d-tree d of Figure 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 $L = 0.1 \otimes ((0.5 \odot 0.4) \oplus 0.35) = 0.595$. Similarly for the upper bound: $U = 0.11 \otimes ((0.5 \odot 0.44) \oplus 0.38) = 0.644$. The condition $U - L = 0.049 \leq 2 \cdot 0.012 = 0.024$ is not satisfied. Hence, we cannot stop now.

(2) We compute $L(d)$ as before: $L(d) = [L, U']$, where $U' = 0.11 \otimes ((0.5 \odot 0.44) \oplus 0.35) = 0.6173$. We then have that $U' - L = 0.0223 \leq 0.024$. We may thus close this point. □

Our incremental algorithm is the compilation scheme of Figure 1, where the variable choices are according to the variable elimination order in Section IV and of Lemma 6.8 discussed in the next section. The nodes in the d-tree are constructed in depth-first manner. Before constructing a node, we perform two checks: (1) the sufficient condition of Proposition 5.8, which tells us whether we already reached an ϵ-approximation and we can safely stop, and (2) the condition of Theorem 5.12 on whether the current node to be constructed can be safely closed, in case the condition at step (1) is not satisfied. In step (2), we compute the bounds of the DNF at the leaf using the Independent heuristic of Figure 3.

E	U	V	P	ϕ
	5	7	.9	e_1
	5	11	.8	e_2
	6	7	.1	e_3
	6	11	.9	e_4
	6	17	.5	e_5
	7	17	.2	e_6

(a)

E'	U	V	$\in$	P	ϕ
	5	7	1	.9	e_1
	5	7	0	.1	$\neg e_1$
	⋮	⋮	⋮	⋮	
	7	17	1	.2	e_6
	7	17	0	.8	$\neg e_6$

(b)

R	ϕ
	$e_3 \wedge e_5 \wedge e_6$

(c)

R	V	ϕ
	6	$e_5 \wedge e_6 \wedge \neg e_3$
	11	$(e_1 \wedge e_2) \vee (e_3 \wedge e_4)$
	17	$e_3 \wedge e_5 \wedge \neg e_6$

(d)

Fig. 5. Tuple-independent (a) and block-independent-disjoint representation (b) of a social network, and results (c,d) of the queries in Section VI-A.

VI. Tractability Results

We next discuss the connection between tractability of query evaluation on probabilistic databases and polynomial-time probability computation with d-trees. For this, we recall how DNFs are obtained by query evaluation using a social network example. We refer to the literature [3], [7], [2] for techniques for evaluating queries on probabilistic databases and casting tuple confidence computation as a problem of computing the probability of a DNF: these techniques are well-established, and we lack the space to cover them in detail.

A. Examples of Query Evaluation on Probabilistic Databases

Consider a representation of a social network as an undirected graph in which nodes represent individuals and edges represent friendship. Assume that the edges are associated with a degree of belief in their presence (e.g., from mail server logs). No correlations between the probabilities of edges are known, so the edge probabilities are assumed independent.

Figure 5 (a) gives a so-called *tuple-independent* table that encodes the edge relation of a social network. The Boolean random variables $e_1, \ldots, e_6$ represent the six edges – that is, the i-th edge is present in those worlds in which e_i is true. This table represents 2^6 possible worlds, each holding a relation of schema $E(U, V)$. For instance, the world with edges e_1, e_2, and e_3, but not the others, has probability $.9 * .8 * .1 * (1 - .9) * (1 - .5) * (1 - .2)$.

The following query computes the probability that there is a triangle (a 3-clique of friends) in this graph (such small patterns are also called *motifs*):

```
select conf() as triangle_prob
from   E n1, E n2, E n3
where  n1.v = n2.u and n2.v = n3.v and
n1.u = n3.u and n1.u < n2.u and n2.u < n3.v;
```

The relational algebra part of this query computes the table of Figure 5 (c). That is, there is a triangle in those worlds that

contain the third, fifth, and sixth edge.

Figure 5 (b) gives an alternative equivalent representation E' of the edge relation E. This is a *block-independent-disjoint* table [7]. The difference to E is that the alternatives – each edge is either present or not – are both represented. Alternatives in a group are mutually exclusive and different groups are independent from each other.

We can now ask queries involving the absence of an edge from a world, such as the query for nodes within two, but not one, degrees of separation from node 7. The query shall be skipped here, although it is not hard to write in positive relational algebra assuming a relation of those edges missing with certainty from the graph is available. The result is the table of Figure 5 (d).

In both examples, we need to compute the probability of the query answers, or equivalently the probability of the DNFs ϕ in Figures 5 (c) and (d).

B. From Tractable Queries to Linear-Size Complete D-Trees

DNFs associated with answers to known tractable queries on tuple-independent probabilistic databases can be compiled efficiently into d-trees. The classes of tractable queries considered here are (1) the hierarchical queries without self-joins [7], (2) queries that are "hard" in general, but become tractable on restricted databases [21], and (3) queries with inequalities [20].

The DNFs associated with answers to any tractable conjunctive query without self-joins are factorizable into one occurrence form (1OF), where each variable occurs exactly once [19]. Such queries are called hierarchical, and can be easily defined using Datalog notation, where joins are expressed by occurrences of query variables in several subgoals.

Definition 6.1 ([7]): A conjunctive query is hierarchical if for any two non-head query variables, either their sets of subgoals are disjoint or one set is contained in the other. □

Example 6.2: The following queries are hierarchical:

$$q_1() :\!- R_1(A,B), R_2(A,C)$$
$$q_2(D) :\!- R_1(A,B,C), R_2(A,B), R_3(A,D)$$

Formulas in 1OF can be arbitrarily nested using $\wedge$ and $\vee$, e.g., $((x_1 \vee x_2) \wedge (\neg y_1 \vee y_2)) \vee (x_3 \wedge \neg y_3)$. □

Complete d-trees can represent 1OFs by turning $\vee$ into $\otimes$ and $\wedge$ into $\odot$. Following our compilation scheme of Figure 1,

Proposition 6.3: Any DNF formula factorizable in 1OF can be compiled in polynomial time into a complete d-tree with one leaf per distinct variable and inner nodes $\otimes$ and $\odot$. □

The query $q :\!- R(X), S(X,Y), T(Y)$ is the prototypical #P-hard query [7]. It is non-hierarchical since the sets of subgoals of X and Y overlap, yet one does not include the other. The DNFs for hard queries are factorizable in 1OF for restricted tuple-independent databases. Due to lack of space, we only state here a tractable case that exploits regularities in the structure of table S.

Theorem 6.4: The DNFs associated with the hard query pattern $R(X), S(X,Y), T(Y)$ are factorizable in 1OF if each connected component of the bipartite graph of S is

- functional, and S can be probabilistic or deterministic, or
- complete, and S is deterministic. □

The bipartite graph G of table S can be obtained as follows: The distinct X-values and Y-values in S form the two disjoint sets of nodes in G, and each tuple (x,y) in S induces an edge between the nodes of x and of y in G. A bipartite subgraph of G over node sets X' and Y' is functional, if either there are no two X'-nodes connected to the same Y'-node, or no two Y'-nodes connected to the same X'-node.

Theorem 6.4 and Proposition 6.3 generalize an early tractability result obtained for hard patterns where functional dependencies hold on the *entire* table S [7], [21].

Very recent work defines tractable queries with inequalities ($<$) [20]. We only consider here the core tractable language of so-called IQ queries defined in that work, all extensions presented in there carry over here as well.

Definition 6.5 ([20]): Let the disjoint sets of query variables $\overline{x_1}, \ldots, \overline{x_n}$. A conjunction of inequalities over these sets has the *max-one* property if at most one query variable from each set occurs in inequalities with variables of other sets. □

Definition 6.6 ([20]): An *IQ query* has the form

$$Q(\overline{x_0}) :\!- R_1(\overline{x_1}), \ldots, R_n(\overline{x_n}), \Phi$$

where $R_1, \ldots, R_n$ are distinct tuple-independent tables, the sets of query variables $\overline{x_1} - \overline{x_0}, \ldots, \overline{x_n} - \overline{x_0}$ are pairwise disjoint, and Φ has the max-one property over these sets. □

Example 6.7: The following are IQ queries

$$q_1() :\!- R(E,F), T(D), T'(G,H), E < D < H$$
$$q_2() :\!- R'(E,F), T(D), S(B,C), E < D, E < C$$
$$q_3() :\!- R(A), T(D)$$
$$q_4() :\!- R(A), T(D), R'(E,F), T'(G,H), A < E, D < E, D < G$$

□

We compile DNFs of IQ queries using the variable elimination order given next in Lemma 6.8. The following new result captures the core observation of our previous work [20]. By *co-factor* of a variable v in a DNF Φ, we denote the DNF Φ' such that $\Phi = v \wedge \Phi' \vee \Psi$, and the DNF Ψ does not contain v.

Lemma 6.8: Let Φ be the DNF of an IQ query over relations $R_1, \ldots, R_k$. Then, there is a variable v from R_i ($1 \leq i \leq k$) that occurs in clauses of Φ with all variables of all R_j, $j \neq i$. Also, the co-factor of v subsumes $\Phi|_v$. □

The variable v can be found as follows. We first compute the number of variables in Φ from each relation $R_1, \ldots, R_k$. We next do the same counting process, but now restricted to those clauses that contain a given variable x. If we obtain the same counts as in the unrestricted case for all but the relation of x, then x is the chosen variable. Otherwise, we check for a different variable until we exhaust the set of variables in Φ. Counting the number of variables of a relation can be done by scanning the clauses of Φ and marking for each variable all but the first of its occurrences; the number of the unmarked variables is the desired count. This requires time bounded by the number of variables times the size of Φ.

The subsumption property is what makes IQ queries tractable. We exemplify with query $q() :\!- R(X), S(Y), X < Y$ on a database with random Boolean variables $x_1, \ldots, x_n$ in R and $y_1, \ldots, y_m$ in S ($n \geq m$). Assume wlog that the indices

of variables correspond to the sorting order of relations R and S. According to Lemma 6.8, x_1 is chosen first and

$$\Phi|_{x_1} = \bigvee_j (y_j) \vee \bigvee_{1<i\leq n} (x_i \wedge \Phi|_{x_i}) = \bigvee_j (y_j)$$
$$\Phi|_{\neg x_1} = \bigvee_{1<i\leq n} (x_i \wedge \Phi|_{x_i}).$$

The co-factor of x_1 is $\bigvee_j (y_j)$, a disjunction of all the variables in S that annotate Y-values that are greater than the X-value annotated by x_1. Following the semantics of the inequality join, any other variable x_i can only be paired with a disjunction of a (non-necessarily strict) subset of variables in S, hence $\bigvee_j (y_j) \vee \bigvee_{1<i\leq n} (x_i \wedge \Phi|_{x_i}) = \bigvee_j (y_j)$. Both DNFs $\Phi|_{x_1}$ and $\Phi|_{\neg x_1}$ can be decomposed using the same heuristic: Any variable y_j is chosen in $\Phi|_{x_1}$, and variable x_2 (or some y_j) is chosen in $\Phi|_{\neg x_1}$. Also, the sum of their sizes is less than the size of Φ. This observation leads to the following result.

Theorem 6.9: The DNF Φ of any IQ query can be compiled in polynomial time into a complete d-tree with $\oplus$-nodes only, such that for each literal in Φ there is at most one $\oplus$-node.□

VII. Experiments

In this section, we report on experiments with our new approximate probability computation algorithm.

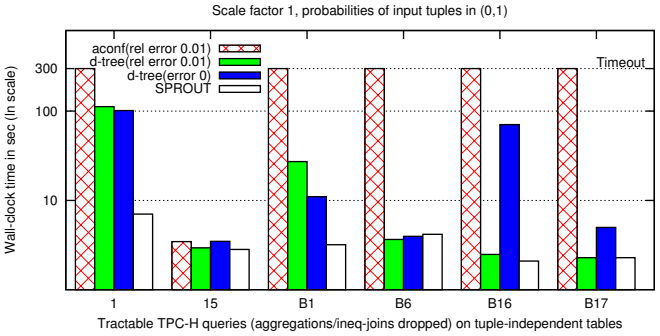
1) Algorithms: We experimentally compare our approach (called d-tree in the sequel) with the following algorithms:

aconf: The algorithm aconf() computes an (ϵ, δ)-approximation of tuple confidence and takes ϵ and δ as arguments. It is a combination of the Karp-Luby unbiased estimator for DNF counting [15] in a modified version adapted for confidence computation in probabilistic databases (cf. e.g. [16]) and the Dagum-Karp-Luby-Ross optimal algorithm for Monte Carlo estimation [6]. The latter is based on sequential analysis and 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 actually use the probabilistic variant of a version of the Karp-Luby estimator described in the book [26] which computes fractional estimates that have smaller variance than the zero-one estimates of the classical Karp-Luby estimator.

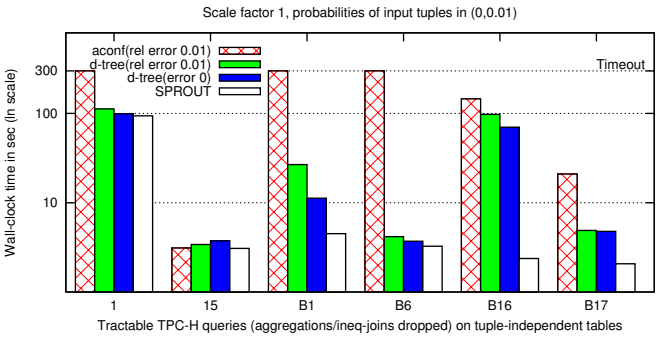
SPROUT: This efficient secondary-storage algorithm is the state of the art exact confidence computation technique for currently all known classes of conjunctive queries with inequalities and without self-joins on tuple-independent probabilistic databases [21], [20].

2) Experimental Setup: The experiments were performed on an AMD Athlon Dual Core Processor 5200B 64bit/3.9GB/Linux2.6.25/gcc 4.3.0.

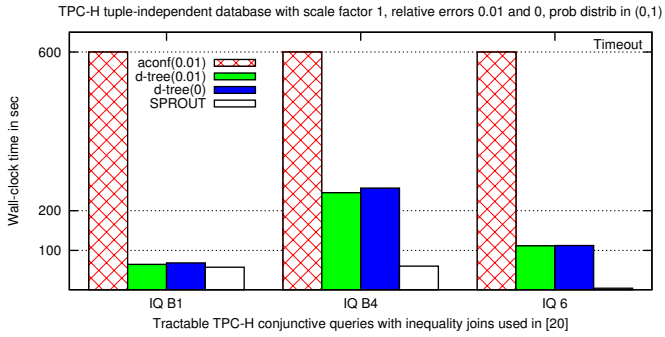
Our technique was implemented within the SPROUT query engine, which is part of the MayBMS probabilistic database management system; the code was added to Release 2.1 beta of MayBMS, which itself is based on Postgresql 8.3.3 (see http://maybms.sourceforge.net). The aconf implementation is the one supported in MayBMS 2.1 beta and was not altered, and the parameter δ is 0.0001 for all the experiments.



(a)



(b)



(c)

Fig. 6. Experimental results for tractable queries.

All resources needed to reproduce our experiments (algorithms, queries, data sets, data set generators) are available at http://web.comlab.ox.ac.uk/projects/SPROUT/.

3) Experiment Design: Our experiments were designed to provide insight into the performance of our technique across a variety of datasets and queries that are representative of future applications of probabilistic databases. Since no benchmark has been established so far for query processing in probabilistic databases, and there is not even wide agreement yet on a set of most relevant use cases, we have to rely on our understanding of the possible sources of hardness in probability compution that may arise in a variety of applications.

In addition to the obvious sources of hardness, such as large data and non-hierarchical queries, which create complex DNFs, there are several subtle issues to be considered, as discussed below. Our experiments are designed to study them.

1. Tuple-independent databases versus databases with more complicated lineage. The queries in our experiments

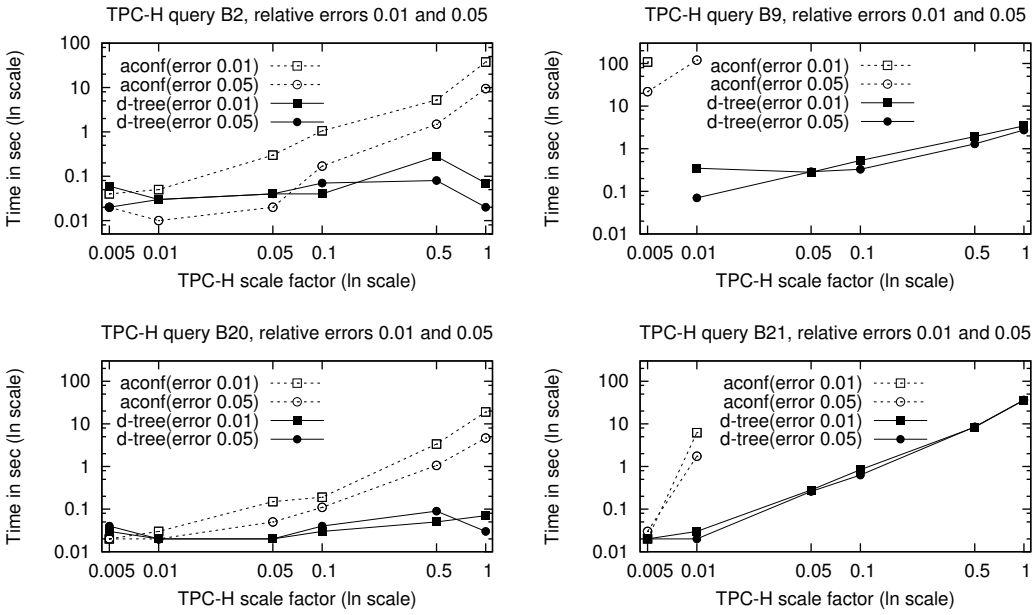


Fig. 7. Experimental results for hard TPC-H queries.

create complex "lineage" formulas. However, we focus on queries whose relational algebra part is positive since the relational difference operation is a substantial source of complexity (cf. e.g. [22]). Thus, if we start with tuple-independent relations in which each tuple is associated with its own Boolean random variable, positive relational algebra queries will only create positive DNFs. This has an effect on the algorithms; in fact, for our algorithm, mixed positive and negated variables in the conditions may possibly make confidence computation easier, because it may allow the upper- and lower-bounding mechanisms to converge more quickly.

2. Easy-hard-easy pattern. In [17], we observed such a pattern similar to those observed in combinatorial algorithms for propositional satisfiability and constraint satisfaction: When the ratio of variables to clauses is very large, then the result probability is rather small and the input to the algorithm is small: such a case tends to be easy. Similarly, if the ratio of variables to clauses is very small, then the result probability tends to be very close to 1 and lower-bounding with sufficient accuracy is easy. However, there is a critical region of variable-to-clause ratios inbetween for which probability computation is hard. For our experiments, this means that there is a pitfall in increasing the instance sizes: 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 and on many instances we will do well without difficulty.

3. Absolute versus relative approximation. When result probabilities are reasonably close to 1, then there is no great difference between 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 the d-tree algorithm is important, since relative approximation is a staple of the Karp-Luby approximation scheme (aconf). Designing a Monte Carlo algorithm for efficient *absolute* approximation is trivial.

A. TPC-H Experiments

The first broad class of experiments was performed on data generated by a modified version of the TPC-H data generator which creates tuple-independent probabilistic databases [7], that is, each tuple occurs in the database independently with a given probability. We consider modified versions of the TPC-H queries without aggregations but with confidence computation.

The queries of the TPC-H benchmark fall into two main classes: tractable queries with inequalities (six hierarchical queries used in [21] and three inequality queries used in [20]), and four #P-hard queries. Queries marked with "B" are Boolean. Two of the tractable queries are selections on the large lineitem table, all other tractable queries are joins of two large tables (e.g., lineitem with supplier, or orders, or part). The hard queries are more complex: 20B is a join on supplier, nation, partsupplier, and part, 21B is a join on supplier, lineitem, orders, and nation, 2B is a join on part, supplier, partsupplier, nation, and region, and 9B is a join on part, supplier, lineitem, partsupplier, orders, and nation.

Fig. 6 shows the running times for computing the answers to tractable queries and their confidences. Overall, d-tree performs worse than SPROUT because SPROUT learns the structure of the DNF from the query, whereas d-tree has to rediscover it on its own. The timing of the two is however comparable in almost all cases.

For hierarchical queries (Fig. 6(a) and (b)) we considered input data with probability distributions in (0,1) and also in (0,0.01). Our algorithm d-tree finishes in all cases within 100 seconds, even for computing the exact confidence. In contrast,

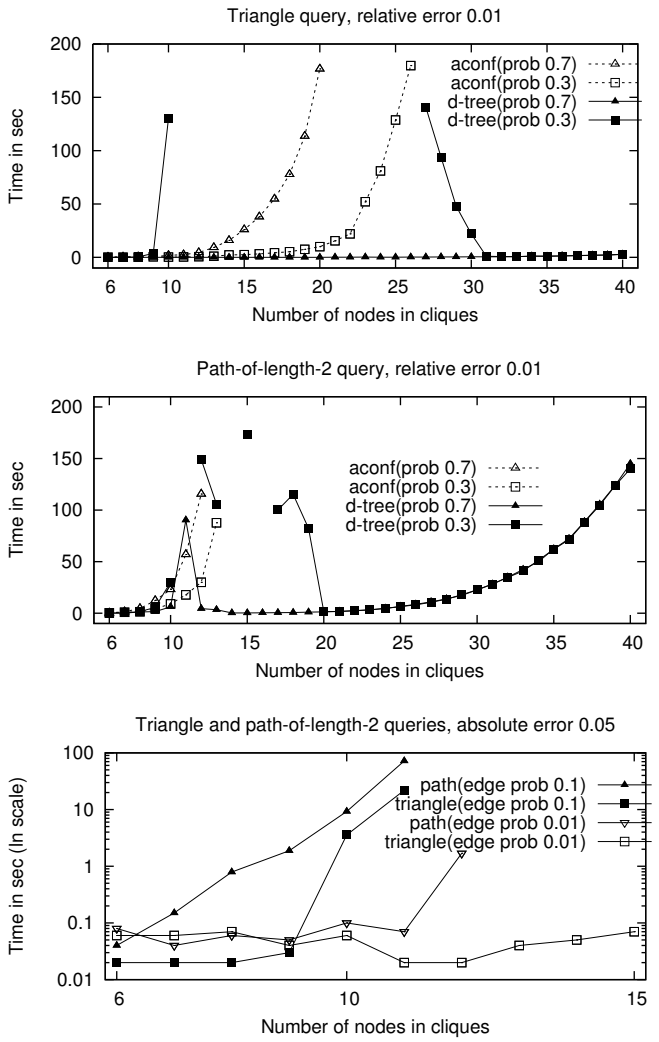


Fig. 8. Experimental results for random graphs.

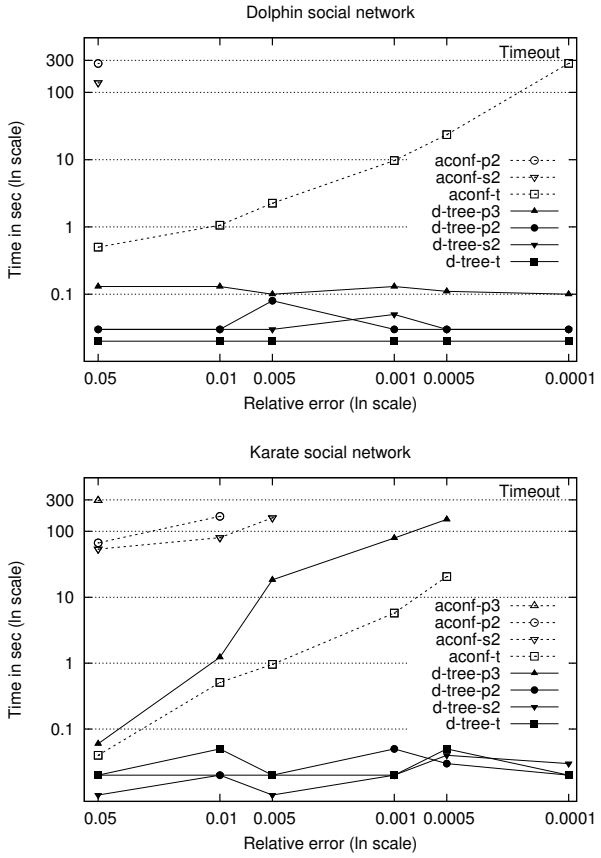


Fig. 9. Experimental results for social networks.

aconf only finishes in four out of the 12 experiments. Overall, we obtain a better timing for error 0 than for relative error 0.01, because in the former case we do not need to compute the lower and upper bounds of each leaf during compilation. This becomes more evident in the case of small probabilities. In case of queries B16 and B17 in Fig. 6(a), checking the bounds clearly pays off: For these cases, no compilation is needed, since the bounds are already approximated very well and we stop early. Without checking the bounds, we would have to construct the entire d-tree, which is then more expensive.

For tractable queries with inequalities (Fig. 6(c)), aconf does not finish in the allocated time, and d-tree closely follows SPROUT.

For all tractable queries, about 90% of the nodes in the d-tree are $\otimes$ nodes, which suggests that our approximation of lower and upper bounds for non-independent sets of clauses works very well and avoids possible exponentiality introduced by variable elimination. In addition, in case of inequality joins, the clause subsumption procedure is very effective. As explained in Section VI-B, this is vital for the overall polynomial time computation. For instance, the IQ query 6 has about 25 distinct answer tuples, each with a DNF of (on average) 10,000 clauses and 550 variables. For each answer tuple, d-tree creates (on average) 20,000 nodes, and subsumes ca. one million clauses (overall, on all branches of the d-tree).

Our algorithm d-tree performs consistently better than aconf also for hard queries. The hard queries have many joins, which ultimately lead to overall low probabilities of clauses, and with final confidences that range from 10^{-3} to 0.93, while answers have up to 500 clauses and 500 variables (query 20), up to 75,000 clauses and 150,000 variables (query 21), up to 640 clauses and 1,600 variables (query 2), and up to 350,000 clauses and 725,000 variables (query 9).

Statistics collected from d-tree traces show that in most of the cases, as the size of DNF increases, the number of nodes constructed by our algorithm also goes up. However, two scenarios may change this trend. First, in the lower and upper bound computation, with more input clauses, both the lower and upper bounds increase but maximal values of upper bounds are 1. If upper bounds reach 1 and lower bounds still increase, this can lead to quick convergence. For instance, for TPC-H query B2 and relative error 0.01, the number of nodes constructed by our algorithm reaches its peak at scale factor 0.5 and drops dramatically at scale factor 1. For larger errors, the U-turn happens even earlier. Still, for TPC-H query B2 but relative error 0.05, the maximal number of nodes appear at scale factor 0.1. Second, the DNF of some TPC-H queries (that have equality selections with constants) has the property that very few variables from one input table occur in

most of the clauses. For instance, for queries B20 and B21, there is only one variable coming from table nation. After we eliminate this variable, the remaining DNF consists of many independent clauses and our approximation approach captures this and tightens the lower and upper bounds very quickly. Therefore, the number of nodes constructed remains low and is not affected by the DNF size.

B. Random Graph and Social Networks Experiments

The second broad class of experiments deals with graph data in which edges are independently either in the graph or absent. We consider two classes of datasets modelled as block-independent disjoint tables. The first set consists of generated 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 case the membership of each edge in the graph is uniform, the probability distribution over this set of possible worlds is uniform, too, 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 Karate club [28], with 34 nodes, a classic, and the other represents friendship among a group of dolphins. The social networks generalize our 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 (very credible for dolphins), or varying degrees of friendship (very credible for karatekas).

We consider four different queries. The first two, triangle (t) and "path of length 2" (p2) were discussed in Section VI-A. The query p3 computes the probability that the graph contains at least one path of length 3. The "separation" query (s2) computes the probability that two given nodes have at most two degrees of separation.

Our experimental results for queries on random graphs and social networks are reported in Figures 8 and 9. In case of random graphs, for large edge probabilities (above 0.5), d-tree 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 triangle and path2 queries.

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 DNF of 780 variables and 9880 clauses; the path2 and path3 queries use a three-way and eight-way self-joins, respectively.

ACKNOWLEDGMENTS

The authors would like to thank the reviewers for their useful comments. Dan Olteanu acknowledges the financial support of the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under the FET-Open grant agreement FOX, number FP7-ICT-233599. Jiewen Huang was supported by a one-year scholarship from Cornell. Christoph Koch was supported by the NSF under grant IIS-0812272.

REFERENCES

[1] V. Akman. "Implementation of Karp-Luby Monte Carlo method: an exercise in approximate counting". *The Computer Journal*, **34**(3):279–282, 1991.

[2] L. Antova, T. Jansen, C. Koch, and D. Olteanu. "Fast and Simple Relational Processing of Uncertain Data". In *Proc. ICDE*, 2008.

[3] O. Benjelloun, A. D. Sarma, C. Hayworth, and J. Widom. "An Introduction to ULDBs and the Trio System". *IEEE Data Engineering Bulletin*, 2006.

[4] E. Birnbaum and E. Lozinskii. "The Good Old Davis-Putnam Procedure Helps Counting Models". *Journal of AI Research*, **10**(6):457–477, 1999.

[5] R. K. Brayton. "Factoring logic functions". *IBM J. Res. Develop.*, **31**, 1987.

[6] P. Dagum, R. M. Karp, M. Luby, and S. M. Ross. "An Optimal Algorithm for Monte Carlo Estimation". *SIAM J. Comput.*, **29**(5):1484–1496, 2000.

[7] N. Dalvi and D. Suciu. "Efficient Query Evaluation on Probabilistic Databases". *VLDB Journal*, **16**(4), 2007.

[8] A. Darwiche and P. Marquis. "A knowlege compilation map". *Journal of AI Research*, 17:229–264, 2002.

[9] M. Davis and H. Putnam. "A Computing Procedure for Quantification Theory". *Journal of ACM*, **7**(3):201–215, 1960.

[10] M. Golumbic, A. Mintza, and U. Rotics. "Read-Once Functions Revisited and the Readability Number of a Boolean Function". In *Proc. Int. Colloq. on Graph Theory*, 2005.

[11] C. P. Gomes, A. Sabharwal, and B. Selman. *Handbook of Satisfiability*, chapter Model Counting. IOS Press, 2009.

[12] E. Grädel, Y. Gurevich, and C. Hirsch. "The Complexity of Query Reliability". In *Proc. PODS*, pages 227–234, 1998.

[13] R. Jampani, F. Xu, M. Wu, L. L. Perez, C. M. Jermaine, and P. J. Haas. "MCDB: a Monte Carlo Approach to Managing Uncertain Data". In *Proc. SIGMOD*, 2008.

[14] R. M. Karp and M. Luby. "Monte-Carlo Algorithms for Enumeration and Reliability Problems". In *Proc. FOCS*, pages 56–64, 1983.

[15] R. M. Karp, M. Luby, and N. Madras. "Monte-Carlo Approximation Algorithms for Enumeration Problems". *J. Algorithms*, **10**(3):429–448, 1989.

[16] C. Koch. "Approximating Predicates and Expressive Queries on Probabilistic Databases". In *Proc. PODS*, 2008.

[17] C. Koch and D. Olteanu. "Conditioning Probabilistic Databases". *PVLDB*, 1(1), 2008.

[18] J. Li and A. Deshpande. "Consensus Answers for Queries over Probabilistic Databases". In *Proc. PODS*, 2009.

[19] D. Olteanu and J. Huang. "Using OBDDs for Efficient Query Evaluation on Probabilistic Databases". In *Proc. SUM*, 2008.

[20] D. Olteanu and J. Huang. "Secondary-Storage Confidence Computation for Conjunctive Queries with Inequalities". In *Proc. SIGMOD*, 2009.

[21] D. Olteanu, J. Huang, and C. Koch. "SPROUT: Lazy vs. Eager Query Plans for Tuple-Independent Probabilistic Databases". In *Proc. ICDE*, 2009.

[22] D. Olteanu, C. Koch, and L. Antova. "World-set Decompositions: Expressiveness and Efficient Algorithms". *Theoretical Computer Science*, 403(2-3), 2008.

[23] C. Ré, N. Dalvi, and D. Suciu. Efficient top-k query evaluation on probabilistic data. In *Proc. ICDE*, 2007.

[24] L. Trevisan. "A Note on Deterministic Approximate Counting for k-DNF". In *Proc. APPROX-RANDOM*, pages 417–426, 2004.

[25] L. Valiant. "The Complexity of Enumeration and Reliability Problems". *SIAM J. Comput.*, **8**:410–421, 1979.

[26] V. V. Vazirani. *Approximation Algorithms*. Springer, 2001.

[27] W. Wei and B. Selman. "A New Approach to Model Counting". In *Proc. SAT*, 2005.

[28] W. W. Zachary. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, **33**:452–473, 1977.