CEUR-WS.org/Vol-2100/paper19.pdf

Towards Deterministic Decomposable Circuits for Safe Queries

Mikaël Monet[1] and Dan Olteanu[2]

[1] LTCI, Télécom ParisTech, Université Paris-Saclay, France,
Inria Paris; Paris, France mikael.monet@telecom-paristech.fr
[2] University of Oxford dan.olteanu@cs.ox.ac.uk

Abstract. There exist two approaches for exact probabilistic inference of UCQs on tuple-independent databases. In the *extensional approach*, query evaluation is performed within a DBMS by exploiting the structure of the query. In the *intensional approach*, one first builds a representation of the *lineage* of the query on the database, then computes the probability of the lineage. In this paper we propose a new technique to construct lineage representations as deterministic decomposable circuits in PTIME. The technique can apply to a class of UCQs that has been conjectured to separate the complexity of the two approaches. We test our technique experimentally, and show that it succeeds on all the queries of this class up to a certain size parameter, i.e., over 20 million queries.

1 Introduction

Probabilistic databases [1] have been introduced in answer to the need to capture data uncertainty and reasoning about it. This uncertainty can come from various angles: imperfect sensor precision of scientific data, imprecise automatic processes (e.g., natural language processing, rule mining in knowledge bases), untrusted data sources (e.g., web crawling), etc. In their simplest and most common form, probabilistic databases consist of a relational database where each tuple is annotated with a probability value that is supposed to represent how confident we are about having this tuple in the database. While a traditional (deterministic) database can only *satisfy* or *violate* a Boolean query, a probabilistic database has a certain probability of satisfying it. Given a Boolean query Q the *probabilistic query evaluation problem for* Q ($\mathsf{PQE}(Q)$) then asks for the probability that the query holds on an input probabilistic database. We measure the complexity of $\mathsf{PQE}(Q)$ as a function of the input database, hence considering that the Boolean query Q is fixed. This is known as *data complexity*, and is motivated by the fact that the queries are usually much smaller than the data.

Unfortunately, even for very simple queries, $\mathsf{PQE}(Q)$ can be intractable. When Q is a union of conjunctive queries (UCQ), a dichotomy result is provided by the work of Dalvi and Suciu [2]: either Q is *safe* and $\mathsf{PQE}(Q)$ is PTIME, or Q is not safe and $\mathsf{PQE}(Q)$ is #P-hard. The algorithm to compute the probability of a safe UCQ exploits the first order structure of the query to find a so called *safe query plan* (using extended relational operators that can manipulate probabilities) and can be implemented within a DBMS. This approach is referred to as *extensional query evaluation*, or *lifted inference*.

A second approach to PQE is *intensional query evaluation* or *grounded inference*, and consists of two steps. First, compute a representation of the *lineage* of the query Q on the database D, which is a Boolean formula intuitively representing which tuples of D suffice to satisfy Q. Second, perform weighted model counting on the lineage to obtain the probability. To ensure that model counting is tractable, we use the structure of the query to represent the lineage in tractable formalisms from the field of knowledge compilation, such as read once Boolean formulas, free or ordered binary decision diagrams (OBDDs, FBDDs), deterministic decomposable normal forms (d-DNNFs), decision decomposable normal forms (dec-DNNFs), deterministic decomposable circuits (d-Ds), etc. The main advantage of this approach compared to lifted inference is that the lineage can help explain the query answer. Moreover, having the lineage in a good knowledge compilation formalism can be useful for other applications: we could for instance change the tuples' probabilities and compute the new result easily, or compute the most probable state of the database that satisfies the query.

What we call the q_9 *conjecture*, formulated by Dalvi, Jha, and Suciu [2, 3], states that for safe queries, extensional query evaluation is strictly more powerful than the knowledge compilation approach. Or in other words, that there exists a query which is safe (i.e., can be handled by the extensional approach) whose lineages on arbitrary databases cannot be computed in PTIME in a knowledge compilation formalism that allows tractable weighted model counting (i.e., cannot be handled by the intensional approach). Note that the conjecture depends on the tractable formalism that we consider. The conjecture has recently been shown by Beame, Li, Roy, and Suciu [4] to hold for the formalism of dec-DNNFs (including OBDDs and FBDDs), which captures the traces of modern model counting algorithms. Another independent result by Bova and Szeider [5] shows that the conjecture also holds when we consider the class of *deterministic structured negation normal forms* (d-SDNNFs), which are d-DNNFs that follow the structure of a *v-tree* [6]. However the question is still open for more expressive formalisms, namely, d-DNNFs and d-Ds. Maybe the conjecture fails for such expressive formalisms, i.e., maybe the reason why PQE is PTIME for safe queries is because we can build deterministic decomposable circuits in PTIME for them?

In this paper we focus on a class of queries (the $\mathcal{H}$-queries) that was conjectured in [2, 3] to separate the two approaches and that was used to prove the conjecture for dec-DNNFs [4] and d-SDNNFs [5]. Our first contribution is to develop a new technique to build d-DNNFs and d-Ds in polynomial time for the $\mathcal{H}$-queries, based on what we call *nice Boolean functions*. Because we were not able to prove that this technique works for all the safe $\mathcal{H}$-queries, our second contribution is to test this technique with the help of the SAT solver Glucose [7] on all the $\mathcal{H}$ queries up to a certain size parameter, that we generated automatically. We found no query on which it does not work. Interestingly, we found a few queries for which we can build d-Ds with a single internal negation at the very top, whereas we do not know if we can build d-DNNFs (could these queries separate UCQ(d-DNNF) and UCQ(d-D)?). We conjecture that this technique can build d-Ds for all safe $\mathcal{H}$-queries.

To do this analysis, we had to solve a task of independent interest, namely, computing explicitly the list of all inequivalent monotone Boolean functions on 7 variables. This task had previously been undertaken by Cazé, Humphries, and Gutkin [8] and by Stephen and Yusun [9]. We reused parts of the code from [8] and confirmed the number of such functions: $490, 013, 148$.

Paper structure We start our presentation with preliminaries in Section 2. We then define the $\mathcal{H}$-queries in Section 3 and review what is known about them. In Section 4 we introduce our technique, and we experimentally demonstrate its effectiveness in Section 5. Our code and all the functions are available online [10].

2 Preliminaries

We will consider in this work the most commonly used model for probabilistic databases: the *tuple-independent* model, where each tuple is annotated with a probability of being present or absent, assuming independence across tuples:

Definition 1. *A* tuple-independent (TID) database *is a pair* (D, π) *consisting of a relational instance* D *and a function* π *mapping each tuple* $t \in D$ *to a rational probability* $\pi(t) \in [0; 1]$*. A TID instance* (D, π) *defines a probability distribution* $\Pr$ *on* $D' \subseteq D$*, where* $\Pr(D') := \prod_{t \in D'} \pi(t) \times \prod_{t \in D \setminus D'} (1 - \pi(t))$*. Given a Boolean query* Q*, the* probabilistic query evaluation problem for Q *(*$\mathsf{PQE}(Q)$*) asks, given as input a TID instance* (D, π)*, the probability that* Q *is satisfied in the distribution* $\Pr$*. That is, formally,* $\Pr(Q, (D, \pi)) := \sum_{D' \subseteq D \text{ s.t. } D' \models Q} \Pr(D')$*.*

Dalvi and Suciu [2] have shown a dichotomy result on UCQs for PQE: either Q is *safe* and $\mathsf{PQE}(Q)$ is PTIME, or Q is not safe and $\mathsf{PQE}(Q)$ is #P-hard. Moreover they show that all the safe queries can be handled by the *extensional approach*, i.e., by using the structure of the query to compute the probability. Due to space constraints, we point to [2] for a presentation of their algorithm to compute the probability of a safe query, though it is not strictly necessary to understand the current paper. We denote by UCQ(P) the set of safe UCQs (hence which corresponds to the set of tractable UCQs if P $\neq$ #P).

By contrast, in the *intentional approach*, one first computes a representation of the *lineage* $\mathsf{Lin}(Q, D)$ of the query Q on the instance D:

Definition 2. *The* lineage *of a Boolean query* Q *over* D *is a Boolean formula* $\mathsf{Lin}(Q, D)$ *on the tuples of* D *mapping each Boolean valuation* $\nu : D \to \{0, 1\}$ *to* 1 *or* 0 *depending on whether* D_ν *satisfies* Q *or not, where* $D_\nu := \{t \in D \mid \nu(t) = 1\}$*.*

The lineage can be represented with any formalism that represents Boolean functions (Boolean formulas, BDDs, Boolean circuits, etc), but the crucial idea is to use a formalism that allows tractable probability computation. In this work we will specifically focus on *deterministic decomposable circuits* (d-Ds) and *deterministic decomposable normal forms* [11] (d-DNNFs).

Definition 3. *Let C be a Boolean circuit (featuring* and*,* or*,* not*, and variable gates). An* and*-gate g of C is* decomposable *if for every two input gates $g_1 \neq g_2$ of g we have $\mathsf{Vars}(g_1) \cap \mathsf{Vars}(g_2) = \emptyset$, where $\mathsf{Vars}(g)$ denotes the set of variable gates that have a directed path to g in C. We call C* decomposable *if each* and*-gate is. An* or*-gate g of C is* deterministic *if there is no pair $g_1 \neq g_2$ of input gates of g and valuation ν of the variables such that g_1 and g_2 both evaluate to* 1 *under ν. We call C* deterministic *if each* or*-gate is. A* negation normal form *(NNF) is a circuit in which the inputs of* not*-gates are always variable gates.*

Probability computation is in linear time for d-Ds (hence, for d-DNNFs): to compute the probability of a d-D, compute by a bottom-up pass the probability of each gate, where and gates are evaluated using $\times$, or gates using $+$, and not gates using $1-x$. While there does not seem to be any interest in using d-DNNFs rather that d-Ds for probabilistic databases, we are also interested by d-DNNFs from a knowledge compilation point of view, as it is currently not known if d-Ds are strictly more succinct than d-DNNFs. We write UCQ(d-DNNF) (resp., UCQ(d-D)) to denote the set of UCQs Q such that for any database instance D, we can compute in polynomial time (in data complexity) a d-DNNF (resp., d-D) representation of $\mathsf{Lin}(Q, D)$. For a study of the intensional approach using weaker formalisms for Boolean functions (read once formulas, ordered and free binary decision diagrams), see [3]. Hence we have:

$$\text{UCQ(d-DNNF)} \subseteq \text{UCQ(d-D)} \subseteq \text{UCQ(P)} \qquad (1)$$

Dalvi, Jha, and Suciu [2, 3] conjectured that the inclusion UCQ(d-D) $\subseteq$ UCQ(P) is strict, i.e., that the extensional approach is strictly more powerful than the intensional approach, and proposed a candidate query to separate these classes (named q_9 and that we define in the next section). The purpose of this paper is to study this conjecture.

3 The $\mathcal{H}$-queries

We define in this section the $\mathcal{H}$-queries and review what is known about them. The building blocks of these queries are the queries h_{ki}, which were first defined in the work of Dalvi and Suciu to show the hardness of UCQs that are not safe:

Definition 4. *Let $k \in \mathbb{N}$, $k \geq 1$. The queries h_{ki} for $0 \leq i \leq k$ are defined by:*

- $h_{k0} = \exists x \exists y\ R(x) \wedge S_1(x, y)$*;*
- $h_{ki} = \exists x \exists y\ S_i(x, y) \wedge S_{i+1}(x, y)$ *for* $1 \leq i < k$*;*
- $h_{kk} = \exists x \exists y\ S_k(x, y) \wedge T(y)$*.*

We define the $\mathcal{H}$-queries to be combinations of queries h_{ki}, as in [4]:

Definition 5. *For $k \geq 1$, we define the set of variables $\langle k \rangle := \{0, \ldots, k\}$. Given a Boolean function ϕ on variables $\langle k \rangle$, we define the Boolean query Q_k^ϕ to be the query represented by the first order formula $\phi[0 \mapsto h_{k0}, \ldots, k \mapsto h_{kk}]$, i.e., ϕ where we substituted each variable $i \in \langle k \rangle$ by the formula h_{ki}.*

The query class $\mathcal{H}_k$ (resp., $\mathcal{H}_k^+$) is then the set of queries Q_k^ϕ when ϕ ranges over all Boolean functions (resp., monotone Boolean functions) on variables $\langle k \rangle$. We finally define $\mathcal{H}$ (resp., $\mathcal{H}^+$) to be $\bigcup_{k=1}^{\infty} \mathcal{H}_k$ (resp., $\bigcup_{k=1}^{\infty} \mathcal{H}_k^+$). Observe that the queries in $\mathcal{H}^+$ are in particular UCQs.

Example 1. Let $k = 3$, and ϕ_9 be the monotone Boolean function $(2 \vee 3) \wedge (0 \vee 3) \wedge (1 \vee 3) \wedge (0 \vee 1 \vee 2)$. Then $Q_3^{\phi_9}$ represents the query $q_9 = (h_{32} \vee h_{33}) \wedge (h_{30} \vee h_{33}) \wedge (h_{31} \vee h_{33}) \wedge (h_{30} \vee h_{31} \vee h_{32}) \in \mathcal{H}_3^+$, which is safe and was conjectured in [2,3] not to be in UCQ(d-D).

To study the $\mathcal{H}$-queries, we need the following notions on Boolean functions:

Definition 6. *Let ϕ be a Boolean function on variables $\langle k \rangle$. We will always consider a valuation ν of $\langle k \rangle$ simply as the set of variables that ν maps to* 1. *We write* $\mathsf{SAT}(\phi)$ *the set of satisfying valuations of ϕ. We say that ϕ* depends on variable *$l \in \langle k \rangle$ if there exists a valuation $\nu \subseteq \langle k \rangle$ such that $\phi(\nu \cup \{l\}) \neq \phi(\nu \setminus \{l\})$. We write* $\mathsf{DEP}(\phi) \subseteq \langle k \rangle$ *for the set of variables on which ϕ depends. We call ϕ and Q_k^ϕ* nondegenerate *if* $\mathsf{DEP}(\phi) = \langle k \rangle$ *(and* degenerate *otherwise).*

Then, if ϕ is degenerate (i.e., does not depend on all its $k+1$ variables), Q_k^ϕ is safe and is in UCQ(d-DNNF) (in fact, even in UCQ(OBDD)):

Proposition 1 (Theorem 3.12 of [4], or Lemma 3.8 of [12]). *Let $k \geq 1$, and $Q_k^\phi \in \mathcal{H}_k$ with* $\mathsf{DEP}(\phi) \subsetneq \langle k \rangle$. *Then $Q_k^\phi \in$ UCQ(d-DNNF).*

This is in contrast to when ϕ is nondegenerate. Indeed, Beame, Li, Roy, and Suciu then show [4] that Q_k^ϕ do not admit polynomial sized decision decomposable NNFs (dec-DNNF). A dec-DNNF is a d-DNNF in which the determinism of or gates is restricted to simply choosing the value of a variable [13,14]. That is, each or gate is of the form $(v \wedge g) \vee (\neg v \wedge g')$ for some variable v. In fact, they show a lower bound for more general representations than dec-DNNFs, namely, for what they called *Decomposable Logic Decision Diagrams* (DLDDs), which generalise dec-DNNFs in that they allow negations at arbitrary places and also allow decomposable binary operator gates. When ϕ is monotone, another independent lower bound by Bova and Szeider [5] tells us than we cannot impose *structuredness* either (i.e., use d-SDNNFs) when ϕ is nondegenerate. These results mean that for such queries, one cannot restrict too much the expressivity of determinism. The question is then: do the nondegenerate queries have polynomial sized d-DNNFs (or d-Ds)?

Let us first see what the dichotomy theorem tells us about $\mathcal{H}$-queries that are nondegenerate. We shall restrict our attention to monotone functions now, i.e., to queries in $\mathcal{H}^+$, because the dichotomy theorem applies only to UCQs, and Q_k^ϕ is not a UCQ when ϕ is not monotone. We need to define the *CNF lattice* of ϕ:

Definition 7. *Let ϕ be a monotone Boolean function on variables $\langle k \rangle$ such that* $\mathsf{DEP}(\phi) = \langle k \rangle$, *and let $F_{\mathrm{CNF}} = C_0 \wedge \ldots \wedge C_n$ be the (unique) minimized CNF* *representing* ϕ*, where we see each clause simply as the set of variables that it contains. For* $\mathbf{s} \subseteq \langle n \rangle$*, we define* $d_{\mathbf{s}} := \bigcup_{i\in\mathbf{s}} C_i$*. Note that* $d_\emptyset$ *is* $\emptyset$*, and that we can have* $d_{\mathbf{s}} = d_{\mathbf{s}'}$ *for* $\mathbf{s} \neq \mathbf{s}'$*. The* CNF lattice *of* ϕ *is the lattice* $(L, \leq)$*, where* L *is* $\{d_{\mathbf{s}} \mid \mathbf{s} \subseteq \langle n \rangle\}$*, and where* $\leq$ *is reversed set inclusion. In particular, the top element* $\hat{1}$ *of* L_{CNF} *is* $\emptyset$*, while its bottom element* $\hat{0}$ *is* $\langle k \rangle$ *(because* ϕ *depends on all the variables, hence each variable is in at least one clause).*

Example 2. The Hasse diagram of the CNF lattice of ϕ_9 is shown in Figure 1 (ignore for now the values at the right inside the nodes).

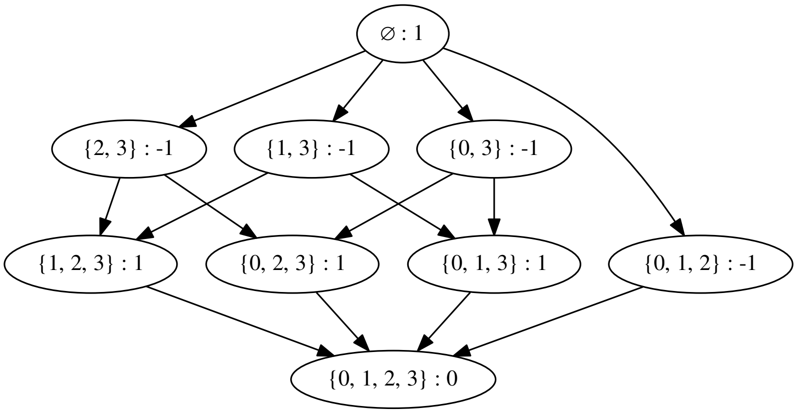

Fig. 1. CNF lattice of ϕ_9 with the values $\mu(n, \hat{1})$ for each node n.

The criterion of Dalvi and Suciu is based on the *Mobius function* [15] of the CNF lattice L of ϕ. The Mobius function $\mu : L \times L \to \mathbb{Z}$ on L is defined on pairs (u, v) with $u \leq v$ by $\mu(u, u) = 1$ and $\mu(u, v) = -\sum_{u<w\leq v} \mu(w, v)$ for $u < v$. The dichotomy theorem for UCQs then states:

1. If $\mu_L(\hat{0}, \hat{1}) \neq 0$, then Q_k^ϕ is #P-hard. Hence, if P is different from #P, we have $Q_k^\phi \notin$ UCQ(d-D).
2. If $\mu_L(\hat{0}, \hat{1}) = 0$, then Q_k^ϕ is PTIME. We do not know if $Q_k^\phi \in$ UCQ(d-DNNF) or $Q_k^\phi \in$ UCQ(d-D). But by what precedes, we know that $Q_k^\phi \notin$ UCQ(dec-DNNF) and $Q_k^\phi \notin$ UCQ(d-SDNNF).

Our goal is to investigate, when we are in the second case (ϕ is monotone, nondegenerate and safe), for which functions ϕ we can build d-DNNFs or d-Ds.

4 Nice Boolean Functions

In this section we present a technique to prove that some queries $Q_k^\phi \in \mathcal{H}_k$ are in UCQ(d-DNNF). This will in particular apply to the query q_9 (which, we recall, was conjectured not to be in UCQ(d-D)). Our goal is to rewrite ϕ as $\phi_0 \vee \ldots \vee \phi_k$, where the ϕ_i are mutually exclusive ($\phi_i \wedge \phi_j \equiv \bot$ for $i \neq j$) and depend on strict subsets $S_i \subsetneq \langle k \rangle$ of $\langle k \rangle$. When such a rewriting exists, we say that ϕ is *nice*:

Definition 8. *Let ϕ be a Boolean function on variables $\langle k \rangle$. We call ϕ* nice *if there exist strict subsets S_i of $\langle k \rangle$ and mutually exclusive Boolean functions (not necessarily monotone) ϕ_i for $0 \leq i \leq k$ such that $\mathsf{DEP}(\phi_i) = S_i$ and $\phi \equiv \bigvee_{i=0}^{k} \phi_i$.*

Observe that allowing to have more (or less) than $k+1$ functions ϕ_i would not change the definition of being nice. Also, note that if ϕ is degenerate, then ϕ is trivially nice.

Example 3. The function ϕ_9 is equivalent to the mutually exclusive disjunction $\phi_0 \vee \phi_1 \vee \phi_2 \vee \phi_3$, where $\phi_0 \equiv 0 \wedge \neg 2 \wedge 3$; $\phi_1 \equiv \neg 1 \wedge 2 \wedge 3$; $\phi_2 \equiv \neg 0 \wedge 1 \wedge 3$; and $\phi_3 \equiv 0 \wedge 1 \wedge 2$. Moreover for $0 \leq i \leq 3$ we have $\mathsf{DEP}(\phi_i) \subsetneq \langle 3 \rangle$, hence ϕ_9 is nice.

When ϕ is nice, we can express Q_k^{ϕ} as $\bigvee_{i=0}^{k} Q_k^{\phi_i}$, thus showing that $Q_k^{\phi} \in$ UCQ(d-DNNF). Indeed, given a database D, we can use Proposition 1 to construct in PTIME a d-DNNF C^{ϕ_i} representing $\mathsf{Lin}(Q_k^{\phi_i}, D)$ for each $i \in \{0, \ldots, k\}$, and then build the d-DNNF $C^{\phi} = \bigvee_{i=0}^{k} C^{\phi_i}$ that represents $\mathsf{Lin}(Q_k^{\phi}, D)$. In other words, the following holds:

Proposition 2. *Let $k \in \mathbb{N}$ and ϕ be a Boolean function on variables $\langle k \rangle$. If ϕ is nice, then $Q_k^{\phi} \in$ UCQ(d-DNNF).*

Hence $q_9 \in$UCQ(d-DNNF). This result shows that, for all queries Q_k^{ϕ} where ϕ is nice, we can compute a d-DNNF representation of their lineage in PTIME, and hence compute their probability efficiently. We do not know to which queries this technique can be applied. Moreover also we have the following corollary:

Corollary 1. *Let $k \in \mathbb{N}$ and ϕ be a Boolean function on variables $\langle k \rangle$. If $\neg\phi$ is nice, then $Q_k^{\phi} \in$ UCQ(d-D).*

We will call *co-nice* a function ϕ such that $\neg\phi$ is nice.

5 Experiments

We have presented a technique that can be used to show that some queries are in UCQ(d-DNNF) or UCQ(d-D), but we have not characterized the queries to which it applies. In this section, we present our experiments that show that for all $k \in \{1, \ldots, 6\}$, every nondegenerate monotone function ϕ for which Q_k^{ϕ} is safe is either nice or co-nice. Hence all the safe queries in $\mathcal{H}_k^+$ for $k \in \{1, \ldots, 6\}$ are in UCQ(d-D). This suggests that UCQ(d-D) = UCQ(P), or at least that any counterexample query in $\mathcal{H}^+$ must be in $\mathcal{H}_k^+$ for $k \geq 7$.

We used a machine with 40 x86 64 CPUs of 2.6 GHz and 512 GB RAM. The code was written in Python 2.7.12 and parallelized using Python's multiprocessing library. We explain briefly how we generated all the functions in $\mathcal{H}_k^+$ for $k \in \{1, \ldots, 6\}$ in Section 5.1, then explain how we tested niceness of these functions in Section 5.2.

5.1 Generating $\mathcal{H}_k^+$

We started by generating the set $R(k)$ of all monotone Boolean functions on variables $\langle k \rangle$ *up to isomorphism*, that is, up to renaming the variables. The size of $R(k)$ corresponds to the OEIS sequence A003182, which is only known up to $k = 6$ (computed in [8] and in [9]). We used parts of the code from [8] to generate all functions in $R(k)$ for k in $\{1, \ldots, 6\}$. We then filtered $R(k)$ to obtain the set of functions that are nondegenerate. Then we tested whether Q_k^ϕ is safe by computing the CNF lattice of ϕ and checking that $\mu(\hat{0}, \hat{1}) = 0$. Let us call $SND(k)$ the set of remaining functions (that is, the functions that are safe and nondegenerate). It took about 2 weeks (using the 40 CPUs) to compute the explicit lists of all the functions in $R(k)$ and $SND(k)$ for $k \in \{1, \ldots, 6\}$, and the sizes of these sets can be found in Table 1. We next explain how we tested the niceness of each function in $SND(k)$.

5.2 Testing Niceness

Let us call *boxes* the functions ϕ_i used in Definition 8. That is, ϕ is nice if and only if we can partition its satisfying valuations into $k+1$ ordered boxes (we allow some boxes to be empty), where the i-th box has a *symmetry around variable i*:

Definition 9. *Let $\nu \subseteq \langle k \rangle$ be a valuation of $\langle k \rangle$, and $l \in \langle k \rangle$ be a variable. We define the valuation* $\mathsf{Toggle}(\nu, l)$ *to be the valuation $\nu \cup \{l\}$ if $l \notin \nu$ and $\nu \setminus \{l\}$ if $l \in \nu$. We say that a set B of valuations of $\langle k \rangle$* has a symmetry around variable *l if for every valuation $\nu \subseteq \langle k \rangle$ we have $\nu \in B$ iff* $\mathsf{Toggle}(\nu, l) \in B$.

To check if ϕ is nice, we build a CNF $\mathsf{Nice}(\phi)$ that expresses exactly that $\mathsf{SAT}(\phi)$ can be partitioned nicely, i.e., $\mathsf{Nice}(\phi)$ is satisfiable if and only if ϕ is nice. We can then use a SAT solver.

Definition 10. *Let $k \geq 1$ and ϕ be a Boolean function on $\langle k \rangle$. We define the CNF* $\mathsf{Nice}(\phi)$ *as follows. Its set of variables is $\{\mathrm{x}_\nu^l \mid \nu \in \mathsf{SAT}(\phi)$ and $l \in \langle k \rangle\}$, where x_ν^l intuitively expresses that ν is put in box l. Its set of clauses is:*

1. *For each $\nu \in \mathsf{SAT}(\phi)$, the clause $\bigvee_{l=0}^{k} \mathrm{x}_\nu^l$, expressing the valuation ν must be put in at least one box;*
2. *For each $\nu \in \mathsf{SAT}(\phi)$ and $l, l' \in \{0, \ldots, k\}$ with $l \neq l'$, the clause $\neg\mathrm{x}_\nu^l \vee \neg\mathrm{x}_\nu^{l'}$, expressing that the valuation ν is in at most one box;*
3. *For each $\nu \in \mathsf{SAT}(\phi)$ and $l \in \{0, \ldots, k\}$, then:*

Table 1. Results of our experiments. The meaning of columns is explained after Proposition 3.

k	$\lvert R(k)\rvert$	$\lvert SND(k)\rvert$	$\lvert N(k)\rvert$	$\lvert co\text{-}N(k)\rvert$	$\lvert BAD(k)\rvert$
1	5	0	0	0	0
2	10	0	0	0	0
3	30	2	2	0	0
4	210	25	25	0	0
5	16,353	2,531	2,529	2	0
6	490,013,148	21,987,161	21,987,094	67	0

(a) If $\mathsf{Toggle}(\nu, l) \notin \mathsf{SAT}(\phi)$*, the clause* $\neg \mathrm{x}^l_\nu$*;*
(b) Else, the clause $\neg \mathrm{x}^l_\nu \vee \mathrm{x}^l_{\mathsf{Toggle}(\nu,l)}$*.*
This ensures that the box l *has a symmetry around* l*.*

Proposition 3. *ϕ is nice iff $\mathsf{Nice}(\phi)$ is satisfiable.*

Now for each function ϕ in $SND(k)$, we constructed the CNF formula $\mathsf{Nice}(\phi)$, and used the SAT solver Glucose [7] to determine if it is satisfiable. If $\mathsf{Nice}(\phi)$ is satisfiable then $Q^\phi_k \in$ UCQ(d-DNNF) and we store ϕ in $N(k)$. If it is not we give the formula $\mathsf{Nice}(\neg\phi)$ to Glucose. If this formula is satisfiable then ϕ is co-nice and $Q^\phi_k \in$ UCQ(d-D) (but we do not know if it is in UCQ(d-DNNF)) and we store ϕ in $co\text{-}N(k)$. If $\mathsf{Nice}(\neg\phi)$ is not satisfiable then ϕ is in $BAD(k)$ and we do not know if Q^ϕ_k is in UCQ(d-D). The results of these experiments are displayed in Table 1, and, as we found no function in $BAD(k)$, imply:

Proposition 4. *All the safe queries in $\mathcal{H}^+_k$ for $k \in \{1, \ldots, 6\}$ are in UCQ(d-D).*

We give here one of the 2 functions that are in $co\text{-}N(5)$, $\phi_{\mathrm{co-N1}} := 24 \wedge 034 \wedge 013 \wedge 12 \wedge 15 \wedge 05 \wedge 35 \wedge 23 \wedge 02 \wedge 25 \wedge 014 \wedge 45$ where we write, for instance, 014 to mean $0 \vee 1 \vee 4$. Could $\phi_{\mathrm{co-N1}}$ separate UCQ(d-DNNF) from UCQ(d-D)?

6 Conclusion

We have introduced a new technique to construct deterministic decomposable circuits for safe $\mathcal{H}$-queries and have experimentally demonstrated its effectiveness on the first 20 million such queries. We conjecture that this technique can build d-Ds for all safe $\mathcal{H}$-queries. We leave open many intriguing questions:

- For the $\mathcal{H}$-queries, can we use the DNF lattice instead of the CNF lattice to decide if the query is safe [16]?
- Can we show (unconditionally to P $\neq$ #P) that if Q^ϕ_k is not safe then ϕ is not nice?
- What is the link between our technique and the notion of *d-safety* defined in [3]?
- Do the queries in *co-N* separate UCQ(d-DNNF) from UCQ(d-D)?

We have put online our code, and the complete list of all the functions studied in Section 5.2, which we believe could be useful for people studying the $\mathcal{H}$-queries. It can be used, for instance, to enter by hand a monotone Boolean function ϕ and check if Q_k^ϕ is safe of not, draw its CNF lattice, check if ϕ is nice, etc.

Acknowledgements. We are grateful to Antoine Amarilli for careful proofreading of the article (and for the many discussions about the code), to Romain Cazé for pointing out to us his code to generate monotone Boolean functions, and to Stephen Tamon for referring us to Romain Cazé. The second author would like to acknowledge Guy van den Broeck for initial discussions on the q9 conjecture. The fact that q9 is expressible as a succinct d-D has been already known to him and has been mentioned in several of his presentations prior to this article. This work was partly funded by the Télécom ParisTech Research Chair on Big Data and Market Insights, and by the EPSRC platform grant DBOnto (L012138) that funded Mikaël's research visit at Oxford.

References

1. Suciu, D., Olteanu, D., Ré, C., Koch, C.: Probabilistic Databases. Morgan & Claypool (2011)
2. Dalvi, N., Suciu, D.: The dichotomy of probabilistic inference for unions of conjunctive queries. J. ACM **59**(6) (2012)
3. Jha, A., Suciu, D.: Knowledge compilation meets database theory: Compiling queries to decision diagrams. Theory of Computing Systems **52**(3) (2013)
4. Beame, P., Li, J., Roy, S., Suciu, D.: Exact model counting of query expressions: Limitations of propositional methods. ACM Trans. Database Syst. **42**(1) (2017)
5. Bova, S., Szeider, S.: Circuit treewidth, sentential decision, and query compilation. In: PODS. (2017)
6. Pipatsrisawat, K., Darwiche, A.: New compilation languages based on structured decomposability. In: AAAI. (2008)
7. Audemard, G., Simon, L.: Predicting learnt clauses quality in modern SAT solvers. In: IJCAI. (2009)
8. Cazé, R.D., Humphries, M., Gutkin, B.: Passive dendrites enable single neurons to compute linearly non-separable functions. PLOS Computational Biology **9**(2) (2013) Code available at `https://github.com/rcaze/PlosCB2013`.
9. Stephen, T., Yusun, T.: Counting inequivalent monotone boolean functions. Discrete Applied Mathematics **167** (2014)
10. Monet, M.: `mikael-monet.net/en/publications.html#monet2018towards`
11. Darwiche, A.: On the tractable counting of theory models and its application to truth maintenance and belief revision. J. Applied Non-Classical Logics **11**(1-2) (2001)
12. Fink, R., Olteanu, D.: Dichotomies for queries with negation in probabilistic databases. ACM Trans. Database Syst. **41**(1) (2016)
13. Huang, J., Darwiche, A.: DPLL with a trace: From SAT to knowledge compilation. IJCAI'05 (2005)
14. Huang, J., Darwiche, A.: The language of search. J. Artif. Int. Res. **29**(1) (2007)
15. Stanley, R.P.: Enumerative Combinatorics: Volume 1. 2nd edn. (2011)
16. Monet, M.: Möbius values of CNF and DNF lattices of a monotone boolean function. `http://cstheory.stackexchange.com/q/39754` (2018)