# Borders of Theories for Cooperative Querying over Uncertain Databases

Chourouk Belheouane, Stephane Jean, Brice Chardin, Allel Hadjali, Hamid Azzoune

# Borders of Theories for Cooperative Querying over Uncertain Databases

Chourouk Belheouane [1,2], Stéphane Jean[2], Brice Chardin[2], Allel Hadjali[2], Hamid Azzoune[1]
[1]LRIA, USTHB
BP 32 El Alia, Bab Ezzouar 16111, Algérie
Email: {cbelheouane, hazzoune}@usthb.dz
[2] LIAS, ISAE-ENSMA/Poitiers University
1, Avenue Clement Ader, 86960 Futuroscope Cedex, France
Email: {stephane.jean, brice.chardin, allel.hadjali}@ensma.fr

***Abstract*—In many real applications, data are intrinsically uncertain due to measurement errors, interpretability issues, information incompleteness, etc. In those uncertain databases, users usually express quality requirements when the system evaluates their queries. However, as they may not be familiar with the contents of the queried database, their queries may be failing i.e., they may return no results or results that do not satisfy the expected degree of certainty. To provide users with relevant information in order to obtain alternative satisfactory results, we introduce a cooperative approach based on the dualization concept. This approach computes a set of meaningful subqueries (MFSs and XSSs) of the initial failing query, which is of paramount importance for query reformulation and relaxation purposes. The conducted experiments show that our proposition, a Mixed Dualization Matrix-Based approach (MDMB), outperforms existing algorithms, especially for large queries.**

## I. Introduction

With the Big data era, managing data uncertainty is one of the major challenging issues that has attracted the interest of several researchers. To represent data uncertainty, most of the existing approaches use either the probability or possibility theory. In the present work, we use the possibility theory mainly because it provides a simple and qualitative way to represent uncertainty.

Querying uncertain data generates answers that are associated with certainty degrees expressing the satisfaction of the query. However, as users may overestimate the certainty of the information available, the submitted query may fail, i.e., return an empty set of answers. Instead of just reporting that no results were found, a more cooperative answer consists in identifying the causes of the failure, as well as some successful subqueries – with non-empty results.

To this end, we introduce an approach based on hypergraph dualization to improve the efficiency of cooperative query processing in the context of uncertain databases. Dualization – i.e. the computation of the transversal hypergraph [1] – has been widely used to solve several problems such as mining frequent itemsets, association rules and functional dependencies, or transforming Boolean expressions between CNF and DNF. One of the most attractive properties of dualization lies in the large panel of efficient evaluation strategies available. To the best of our knowledge, it is the first time that dualization is used in handling the failing queries problem, known also as the empty answer problem.

Our approach to tackle the empty answer problem is to compute (i) failure causes (Minimal Failing Subqueries: MFSs) which are of interest for query reformulation or relaxation, and (ii) successful subqueries (maXimal Succeeding Subqueries: XSSs) that provide non-empty alternative answers [2]. This computation procedure consists of 3 steps: 1) a binary matrix representing the satisfaction of the predicates involved in the failing query is first calculated, 2) a set of XSSs is then extracted from the matrix and finally, 3) a set of MFSs is identified using a hypergraph dualization-based method.

In this paper, we substantially develop and improve the third step of this process over our previous work [3]. The novelty of this contribution consists in identifying and defining the requirements for dualization, proving its applicability for the empty answer problem, and evaluating its performance benefits. As a result, our new Mixed Dualization Matrix-Based approach (MDMB) handles large queries (with up to 25 predicates) with acceptable response times, while previous existing approaches did not scale well w.r.t. the number of predicates. To that end, MDMB benefits from an efficient general-purpose dualization algorithm called Sparse Hypergraph Dualization (SHD) [4].

The paper is structured as follows: In Section II we discuss related work and emphasize the motivation of the present study. We introduce in Section III some basic notions along with the problem statement. Section IV is dedicated to the presentation of dualization concepts used in our approach. Our proposed Mixed Dualization Matrix-Based approach is then detailed in Section V. We present our implementation and experimental evaluation of MDMB in Section VI and conclude in Section VII.

## II. Related work

The empty answer problem has been addressed by several complementary approaches such as query auto-completion, query relaxation or database completion. In this section, we focus on works providing cooperative answers to the user based on the computation of MFSs and XSSs.

Among the first works to have addressed the MFSs and XSSs computation problem in the context of relational databases, we find the work of Godfrey [2]. His main contributions were (1) the definition of the complexity of the MFSs and XSSs computation problem and (2) the proposition of the ISHMAEL algorithm that computes either MFSs or XSSs by exploring the lattice of subqueries of the initial query. This work was adapted to the RDF context by Fokou et al. [5], for which they proposed the LBA approach to compute both MFSs and XSSs. LBA was then extended in $\alpha$LBA [6] to take into account uncertain RDF data. McSherry [7] also proposed an approach (abbreviated as MCS in this paper) based on the exploration of the subquery lattice to compute MFSs, for use during interactive query relaxation sessions. The main limitation of the aforementioned four approaches is that they require an exponential number of queries against the database (w.r.t, the number of predicates $|Q|$) which limits their scalability for large queries.

Instead of exploring the subquery lattice, Jannach [8] proposed to compute an intermediary representation of the data as a binary matrix using only $|Q|$ queries. The XSSs can then be extracted directly from this matrix, requiring no further access to the database. This idea was followed in several works. Pivert and Smits [9] proposed an approach that calculates gradual MFSs using a summary computed by a disjunctive query for a set of thresholds. For the different satisfaction thresholds involved, the summary represents fuzzy cardinalities, namely how many tuples from the database satisfy each conjunctive subquery of Q. However, just as $\alpha$LBA, their approach is designed to consider sets of thresholds, while we focus on a single certainty requirement provided by the user to reduce the size of the search space. Our previous approach, Matrix-Based Search (MBS) [3], computes XSSs and MFSs in the context of uncertain relational databases for relatively small queries ($\leq$ 16 predicates). MBS is similar to MCS in its construction of a binary matrix, but with a single disjunctive query composed of the failing query's predicates. Table I summarizes the application domains and main characteristics of the aforementioned approaches.

The main strength of MBS is that it executes a single query against the database. However, as we will show in Section VI, this approach does not scale for large queries during the computation of MFSs. Indeed, according to [3], the MFS computation time of MBS exceeds 50% of the total for queries with 16 predicates or more. Yet, large queries are generated in a number of domains such as database integration, automated query formulation or graphical query languages. As a consequence, we aim in this work at improving the MFS computation time.

TABLE I
RELATED WORK CHARACTERISTICS

| Approach | Database | #Queries | Search | Answer |
|---|---|---|---|---|
| ISHMAEL [2] | Relational | multiple | lattice | MFS or XSS |
| LBA [5] | RDF | multiple | lattice | MFS and XSS |
| $\alpha$LBA [6] | Uncertain RDF | multiple | lattice | MFS and XSS |
| MCS [7] | Recom. Syst. | multiple | lattice | MFS |
| MBS [3] | Uncertain Rel. | 1 | matrix | MFS and XSS |
| Jannach [8] | Recom. Syst. | $\|Q\|$ | matrix | MFS or XSS |
| Pivert [9] | Gradual Queries | 1 | summary | MFS and XSS |

## III. Preliminaries and Problem Statement

Among the large number of uncertainty models proposed in the literature [10] [11], we consider in this work the model based on possibilistic certainty proposed by Bosc et al [10] due to its simple and qualitative nature. Moreover, relational algebra operators are easily extended to this model [12].

### A. Certainty-Based Model

In this model, uncertainty in data is represented by a couple $(v, \alpha)$ where $v$ is the value of a tuple $t$ for an attribute $a$ and $\alpha \in ]0, 1]$ is the *certainty degree* (called *necessity degree* in the possibilistic theory). By default, a value has a certainty equals to 1. For instance, $\langle 1, John, (Paris, 0.6)\rangle$ denotes the existence of a person named *John*, who lives in *Paris* with a certainty equal to 0.6.

By associating an additional degree $\beta$ to each tuple $t$ of an uncertain relation $r$, we can represent *maybe tuples* in this model which expresses the certainty that $t$ exists in $r$. We assume that $\beta = 1$ for all tuples since a value of $\beta \neq 1$ does not result in any particular difficulty. In the case where a tuple $t$ may have different values for a given attribute $a$, the representation used consists in defining a disjunctive set of values with a degree of certainty: $(v_1 \vee \cdots \vee v_n, \alpha)$. Again, we assume that uncertainty is represented with only one value for the sake of simplicity, but our approach could be easily extended to handle multiple uncertainty values by using techniques developed in [12].

### B. MFSs and XSSs

In our approach, we deal with conjunctive queries $Q = p_1 \wedge p_2 \wedge \cdots \wedge p_d$ where each $p_i$ is a valid predicate on an uncertain relation $R$. The relation $R$ considered could itself be the result of a complex query involving all relational algebra and aggregation operators, and not only conjunctions of predicates. Let us consider the uncertain relation in Table II and a query searching for mechanics who live in Bern ($Q$ : $Job = Mechanic \wedge City = Bern$). Tuple 6 will be selected with a certainty degree equal to $min(0.3, 0.8)$ (in accordance with the calculus of necessity measures [13]). We denote by $[[Q]]_D$ the evaluation of $Q$ over an uncertain database $D$ and by $degree(r, Q)$ the certainty degree of each result $r \in [[Q]]_D$.

We consider a *threshold query* $Q_\alpha$ where only results with certainty degrees greater than or equal to a given threshold $\alpha$ are selected. The evaluation of a threshold *query* $Q_\alpha$ on an uncertain database $D$ is defined by: $[[Q_\alpha]]_D = \{r \in [[Q]]_D \mid degree(r, Q) \geq \alpha\}$. Given a threshold query $Q_\alpha = p_1 \wedge \cdots \wedge p_d$, a threshold query $Q'_\alpha = p_i \wedge \cdots \wedge p_j$ is a *subquery* of $Q_\alpha$, $Q'_\alpha \subseteq Q_\alpha$, iff $\{p_i, \cdots, p_j\} \subseteq \{p_1, \cdots, p_d\}$. If $\{p_i, \cdots, p_j\} \subset \{p_1, \cdots, p_d\}$, we say that $Q'_\alpha$ is a *proper subquery* of $Q_\alpha$ ($Q'_\alpha \subset Q_\alpha$).

A failing subquery $Q^*_\alpha$ of a threshold query $Q_\alpha$ is a subquery whose evaluation returns an empty set: $[[Q^*_\alpha]]_D = \emptyset$.

*Definition 1:* A Minimal Failing Subquery (MFS) $Q^*_\alpha$ is a failing subquery for which none of its proper subqueries is failing: $[[Q^*_\alpha]]_D = \emptyset \ \wedge \nexists \ Q'_\alpha \subset Q^*_\alpha$ such that $[[Q'_\alpha]]_D = \emptyset$.
The set of all MFSs of a query $Q_\alpha$ is denoted by $mfs(Q_\alpha)$.

Conversely, a succeeding subquery $Q^*_\alpha$ of a threshold query $Q_\alpha$ is a subquery whose evaluation returns at least one result: $[[Q^*_\alpha]]_D \neq \emptyset$.

*Definition 2:* A Maximal Succeeding Subquery (XSS) $Q^*_\alpha$ of a query $Q_\alpha$ is a succeeding subquery that is not a proper subquery of any succeeding subquery: $[[Q^*_\alpha]]_D \neq \emptyset \ \wedge \nexists \ Q'_\alpha$ such that $Q^*_\alpha \subset Q'_\alpha \wedge [[Q'_\alpha]]_D \neq \emptyset$.
The set of all XSSs of a query $Q_\alpha$ is denoted by $xss(Q_\alpha)$.

## IV. Dualization and Borders of Theories

### A. Basic Concepts of Dualization

In this section, we will introduce some aspects about dualization and its applications. The interested reader can refer to [1] or [14] for more details.

The dualization problem is the construction of the dual hypergraph. Let $\mathcal{H} = (V, F)$ be a hypergraph where $V$ is the set of vertices and $F$ the set of hyperedges. A hyperedge $E \in F$ is a non-empty subset of vertices: $E \subseteq V \wedge E \neq \emptyset$.

A subset T of V is a transversal of $\mathcal{H}$ if it intersects every hyperedge of F:

$$\forall \ E \in F, \ T \cap E \neq \emptyset.$$

A transversal T is called minimal if it is not a superset of any other transversal:

$$T \text{ of } \mathcal{H} \text{ is minimal if } \nexists \ T' \subset T \mid T' \text{ is a transversal.}$$

The dual hypergraph of $\mathcal{H}$ is then defined as the hypergraph $(V, G)$ where G is the set of all minimal transversals of $\mathcal{H}$.

Let us consider, for instance, the hypergraph $\mathcal{H} = (V, F)$ where $V = \{a, b, c, d\}$ and $F = \{\{a\}, \{b, c\}, \{c, d\}\}$. The subset $\{a, c, d\}$ is a transversal but is not minimal as its subset $\{a, c\}$ is also a transversal. The dual hypergraph of $\mathcal{H}$ is given by $dual(\mathcal{H}) = (V, \{\{a, c\}, \{a, b, d\}\})$.

When clear from context, we will omit the set of vertices and use $dual(F)$ to denote the set of hyperedges of the dual hypergraph.

Dualization have been used in many domains to answer problems such as:

- mining frequent itemsets,
- mining inclusion and functional dependencies,
- mining association and strong rules,
- computing minimal keys and hitting set,
- transforming Boolean expressions between CNF and DNF,
- computing negative and positive borders of a monotone Boolean function on a lattice,

and therefore offers many efficient implementations.

In this section, we will show that computing MFSs from XSSs is equivalent to the problem of computing a negative border from a positive one, thus allowing the use of dualization for this task. In this context, borders – MFSs and XSSs – are seen as a compact representation of respectively all failing and all non-failing subqueries in the search space, here the lattice of all possible subqueries of the original query.

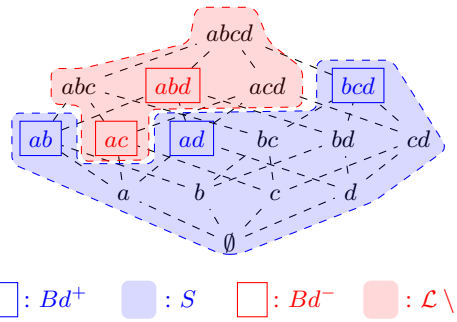


Fig. 1. Borders of a theory $S$ in a lattice

First, we introduce the computation of borders in the general case. A lattice is defined by a set $\mathcal{L}$ of symbols, called language, and a partial order relation $\preceq$ on $\mathcal{L}$ where any two elements of $\mathcal{L}$ share a unique supremum and infimum.

A monotone property $P$ over a lattice is a monotone Boolean function $P : \mathcal{L} \to \{0, 1\}$ such that $\forall a, b \in \mathcal{L}^2, a \preceq b \Rightarrow P(a) \geqslant P(b)$ (monotonically decreasing). Without loss of generality, we only consider decreasing functions since increasing functions can be accounted for by negation.

Let us now consider a set S of sentences from $\mathcal{L}$ such that S is closed under the relation $\preceq$:

$$\forall a, b \in \mathcal{L}^2, b \in s \wedge a \preceq b \Rightarrow a \in S.$$

Since $P$ is monotone, the set of sentences verifying $P$ is closed under $\preceq$. This set of sentences is called a theory $\mathcal{T}h(\mathcal{L}, P) = \{a \in \mathcal{L} \mid P(a) = 1\}$.

The border $Bd(S)$ of S consists of the union of a positive and a negative border. The positive border of S is the set of all largest elements of S and the negative border is the set of all smallest elements of $\mathcal{L} \setminus S$, more formally:

$$Bd(S) = Bd^+(S) \cup Bd^-(S),$$

where

$$Bd^+(S) = \{a \in S \mid \forall \ b \in \mathcal{L}, a \preceq b \Rightarrow b \notin S\} \qquad (1)$$

and

$$Bd^-(S) = \{a \in \mathcal{L} \setminus S \mid \forall \ b \in \mathcal{L}, b \preceq a \Rightarrow b \in S\} \qquad (2)$$

Figure 1 illustrates the notions of borders for a given closed set of sentences $S = \{\emptyset, a, b, c, d, ab, ad, bc, bd, cd, bcd\}$.

Given a positive border, the problem of computing its associated negative border is equivalent to the computation of a dual hypergraph [1].

Let $E^{\complement}$ be the complement of a hyperedge $E$ with respect to the set of vertices $V$:

$$E^{\complement} = V \setminus E$$

.

Let the complement $F^{\complement}$ of a set of hyperedges $F$ be the set of the complements of each hyperedge:

$$F^{\complement} = \{E^{\complement} \mid E \in F\}.$$

Positive and negative borders are then bound by the following equations [1]:

$$Bd^{-}(S) = dual(Bd^{+}(S)^{\complement}) \quad (3)$$

$$Bd^{+}(S) = dual(Bd^{-}(S))^{\complement} \quad (4)$$

Following the example depicted in Figure 1,

$$Bd^{+} = \{\{a, b\}, \{a, d\}, \{b, c, d\}\},$$

$$Bd^{+\complement} = \{\{c, d\}, \{b, c\}, \{a\}\}, \text{ and}$$

$$dual(Bd^{+\complement}) = \{\{a, c\}, \{a, b, d\}\} = Bd^{-}.$$

### B. Dualization and the Empty Answer Problem

For this problem instance, the language $\mathcal{L}$ is the powerset of predicates of the failing query, and the partial order is the set inclusion on predicates. Each element of $\mathcal{L}$ therefore depicts a subquery of the original query (which is itself the supremum of the lattice).

Our objective is to compute MFSs from XSSs. To the best of our knowledge, no work has addressed this problem using dualization. The considered property is then whether the subquery composed of these predicates provides non-empty results:

$$P(Q_\alpha) = ([[Q_\alpha]]_D \neq \emptyset).$$

The considered theory is then the set of succeeding subqueries of the original query $Q_\alpha$:

$$\mathcal{T}h(\mathcal{L}, P) = \{Q^*_\alpha \subseteq Q_\alpha \mid [[Q^*_\alpha]]_D \neq \emptyset\}.$$

Since $P$ is monotone, there exist positive and negative borders of this theory, and by definition – equations (1) and (2) – the positive border $Bd^{+}$ is the set of all XSSs, and the negative border $Bd^{-}$ is the set of all MFSs:

$$Bd^{+}(\mathcal{T}h(\mathcal{L}, P)) = \{XSS\},$$

$$Bd^{-}(\mathcal{T}h(\mathcal{L}, P)) = \{MFS\}.$$

Figure 2 illustrates these notions in the context of query relaxation. Applying these notations to equations (3) and (4) gives:

$$\{MFS\} = dual(\{XSS\}^{\complement}),$$

$$\{XSS\} = dual(\{MFS\})^{\complement}.$$

MFSs can therefore be computed from XSSs using hypergraph dualization. We will show in section VI that using optimized general-purpose algorithms for this computation is more efficient than relying on existing query relaxation-specific algorithms.

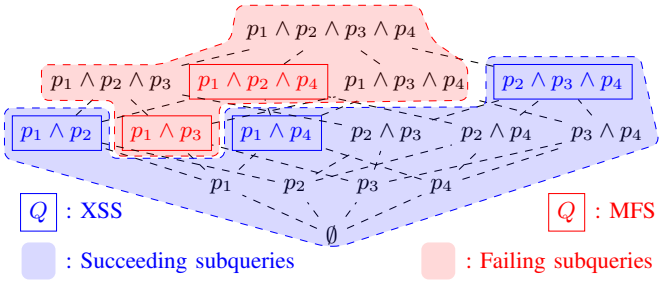


Fig. 2. MFSs and XSSs as borders in the subquery lattice

TABLE II
EXAMPLE OF A RELATION IN AN UNCERTAIN DATABASE

| id | Name | Age | City | Job |
|---|---|---|---|---|
| 1 | (John, 0.7) | (26, 0.8) | (Bern, 0.2) | (Engineer, 0.3) |
| 2 | (Bob, 0.3) | (27, 0.9) | (Bern, 0.4) | (Engineer, 0.5) |
| 3 | (Bob, 0.6) | (24, 0.3) | (Rome, 0.8) | (Mechanic, 0.9) |
| 4 | (John, 0.6) | (27, 0.2) | (Paris, 0.6) | (Engineer, 0.8) |
| 5 | (Mary, 0.7) | (25, 0.7) | (Paris, 0.6) | (Engineer, 0.8) |
| 6 | (John, 0.9) | (24, 0.8) | (Bern, 0.3) | (Mechanic, 0.8) |

## V. MDMB: A Mixed Dualization Matrix-Based Approach

The main objective of our approach is to compute the sets of MFSs and XSSs for a failing threshold query $Q_\alpha = p_1 \wedge p_2 \wedge ... \wedge p_d$ to assist the user in his understanding of the query failure regarding the uncertain database. MDMB computes the MFSs and XSSs in three steps: 1) computing a binary representation of succeeding subqueries, 2) extracting the XSSs, and 3) computing the MFSs. For completeness, steps 1 and 2 – from [3] – are outlined in this paper.

As a running example, let us consider an uncertain relation representing individuals, elaborated in Table II, and the following failing threshold query: $Q_{0.4} : Name = John \wedge Age \geqslant 26 \wedge City = Bern \wedge Job = Engineer$ $(p_1 \wedge p_2 \wedge p_3 \wedge p_4)$.

### A. Computation of the Binary Matrix

In this first step, our approach computes a binary matrix of Succeeding Subqueries of $Q_\alpha$ (SSQs) where every row is a subquery that satisfies at least one of the query predicates. In the following, this matrix SSQs set, represented by $MSSQ(Q_\alpha, D)$ is defined as: $MSSQ(Q_\alpha, D) = \{ SSQ_\alpha \subset Q_\alpha \mid \exists r \in [[SSQ_\alpha]]_D \wedge \nexists Q'_\alpha : SSQ_\alpha \subset Q'_\alpha \wedge r \in [[Q'_\alpha]]_D \}$. Based on this set, the binary matrix is defined as follows:

*Definition 3:* The binary matrix $M$ of a threshold query $Q_\alpha = p_1 \wedge p_2 \wedge ... \wedge p_d$ on an uncertain database $D$ is a two-dimensional table created with subqueries $SQ \in MSSQ(Q_\alpha, D)$ as rows and predicates $p_i \in Q$ as columns. For a subquery $SQ$ and a predicate $p_i \in Q$, if $p_i \in SQ$, then $M[SQ][p_i] = 1$, else $M[SQ][p_i] = 0$.

This binary matrix is computed using a disjunction of the predicates of the threshold failing query $Q_\alpha$. This disjunction ensures that every subquery satisfying at least one predicate and at most $|Q_\alpha| - 1$ predicates is included, which implies that the matrix contains all the XSSs of $Q_\alpha$.

TABLE III
BINARY MATRIX OF OUR THRESHOLD QUERY EXAMPLE

| *Name=John* | *Age* $\geq 26$ | *City=Bern* | *Job=Engineer* |
|---|---|---|---|
| 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 |

*Proposition 1:* The binary matrix of a threshold query $Q_\alpha$ contains its XSSs.

*Proof 1:* Let $Q^*$ be an XSS of $Q_\alpha$. By definition, $Q^* \subset Q_\alpha$. Moreover, since $Q^*$ is successful, its evaluation on $D$ contains at least one result: $\exists\, r \in [[Q^*]]_D$. Finally, as $Q^*$ is maximal, it cannot have a successful superquery: $\nexists\, {Q^*}'_\alpha \mid Q^*_\alpha \subset {Q^*}'_\alpha \wedge r \in [[{Q^*}'_\alpha]]_D$. Thus, $Q^* \in \mathit{MSSQ}(Q_\alpha, D)$.

The size of the matrix cannot exceed the size of the subquery lattice i.e., $2^d - 2$ rows (it is assumed that the initial query fails and the empty query succeeds). This binary matrix can easily be stored in main memory by using bitmap structures. Additionally, our proposed approach only runs a single query: no further access to the database is required after this step, which significantly reduces XSSs computation time.

Table III shows the computed matrix for the running example, where each row of this matrix represents an SSQ of $Q_{0.4}$. For example, the first row corresponds to the SSQ $Q'_{0.4} : Name = John \wedge Age \geqslant 26$. This SSQ is successful as the first row of Table II satisfies it.

### B. Computing XSSs from the Binary Matrix

Since the binary matrix represents a set of succeeding subqueries, it contains both maximal and non-maximal subqueries. The computation of the XSSs can be done by removing the non-maximal SSQs from it. This problem can be reduced to the computation of the *skyline* of query results (also known as *the maximum vector problem*) [15]. Among the existing skyline algorithms, in this work, we adapt the nested loop algorithm.

Our algorithm computes the XSSs of a failing threshold query $Q_\alpha$ from its binary matrix by keeping a list of the currently found XSSs. When a new SSQ $SQ$ is read from the binary matrix, $SQ$ is compared to all queries in the list of current XSSs. Three cases can occur: (1) $SQ$ is a subquery of one of the current XSSs. In this case, $SQ$ is eliminated as it cannot be an XSS, (2) $SQ$ is a superquery of some current XSSs. In this case, these queries are removed from the list of current XSSs and $SQ$ is added to this set, and (3) $SQ$ is neither a subquery nor a superquery of one of the current XSSs (i.e., these queries are not *comparable*), $SQ$ is added to the list of current XSSs.

Let $n$ be the size of the binary matrix. In the worst case, every SSQ of the binary matrix will need to be compared against every other SSQ of this matrix and our algorithm has a worst-case runtime of $\mathcal{O}(n^2)$. This complexity is polynomial in $n$ but exponential in $d$, the size of the query (whose upper-bound is $2^d - 2$). As Godfrey has shown that enumerating

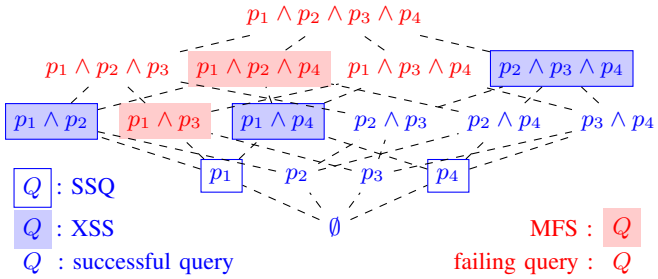


Fig. 3. MFSs, XSSs and SSQs in the lattice of subqueries

the MFSs of a failing query is NP-hard [2], this result is not surprising. In this work, which deal with large input queries, our experiments presented in Section VI show that, even for large queries (up to 25 predicates), our approach is efficient in practice.

### C. Computing MFSs from XSSs by Dualization

Following the relationship between MFSs from XSSs presented in Section IV, the computation of MFSs consists of two steps: 1) computing the complement of each XSS and 2) dualizing the hypergraph whose hyperedges consist of these complements.

Complements in step 1 are computed straightforwardly using sorted lists. For instance in our running example, the complement of $XSS_1 = p_1 \wedge p_2$ is $(XSS_1)^{\complement} = p_3 \wedge p_4$. To comply with the input format of the next step, complements are represented as sets of integers, where $(XSS_1)^{\complement} = p_3 \wedge p_4$ becomes $\{3, 4\}$.

For step 2, we chose the Sparse Hypergraph Dualization (SHD) algorithm [4], for which the authors provide a reference implementation[1]. We opted for SHD due to its efficiency, but other dualization algorithms could have been selected instead. SHD dualizes an hypergraph $(V, F)$ in $O(||F|| \times |V|)$ time and $O(||F||)$ space [4], where $||F||$ represents the total size of the hyperedges:

$$||F|| = \sum_{E \in F} |E|.$$

In the context of query relaxation, the computation of MFSs from XSSs is therefore linear in both the number of predicates and the size of the hypergraph, although the size of the hypergraph is itself exponential in the number of predicates. This algorithm also provides good results in practice even though the problem of computing the dual hypergraph is NP-hard.

Figure 3 illustrates the final output of our running example (three MFSs and two XSSs) as well as intermediary results (five SSQs) in the subquery lattice.

## VI. EXPERIMENTAL EVALUATION

In this section, we investigate the scalability of our approach, MDMB, and compare it with three algorithms from

[1]Available at http://research.nii.ac.jp/ uno/code/shd.html.

the state of the art.

**Implemented Algorithms.** MDMB is an improvement of the MBS algorithm [3] to handle larger queries. MBS therefore acts as a baseline to evaluate the benefits of our optimization. MDMB is also compared to two alternative algorithms: MCS [7] and LBA [5], described in section II. As these approaches (MCS and LBA) have been proposed in non-uncertain contexts, we have adapted these algorithms in order to compare experiments. This adaptation consists in adapting the original predicates to include the verification of the certainty threshold. For instance, the conjunction $(Name = John \wedge Age \geqslant 26)$ becomes $(Name = John \wedge degree(Name) \geqslant \alpha) \wedge (Age \geqslant 26 \wedge degree(Age) \geqslant \alpha)$.

**Datasets and Queries.** Since there exists no benchmark dedicated to the empty answer problem, we used a data generator proposed by Börzsönyi et al. [15]. This generator is widely used in skyline research to provide datasets with or without correlation. More formally, it generates databases of tuples (or points) with varying number of dimensions using one of the following three value distributions:

- *correlated* (*corr*): tuples with higher values in one dimension tend to have higher values in all other dimensions,
- *anti-correlated* (*anti*): tuples with higher values in one dimension tend to have lower values in at least one other dimension,
- *independent* (*indep*): values are generated using uniform distributions for each dimension.

Datasets are generated with standard parameters for results with all four algorithms to be comparable. Generated values range between 0 and 1, for a number of dimensions $d$ between 4 and 25. The size of the dataset (i.e., the cardinality or number of tuples) $n$ varies from 500K to 8M rows. Unless stated otherwise, the default number of dimensions is set to $d = 18$ and the default cardinality is $n$ =1M.

The datasets are generated so that each one is included in the other, in other words, the 500k dataset is a subset of the 1M dataset, which is itself a subset 2M on and so forth. The degrees of certainty were generated randomly using a uniform distribution between 0 and 1. We experimented with different indexes to decrease the running time of the queries, and found that the best performance was provided by using B-tree indexes for each couple $(d, d\_v)$, where $d$ is a dimension and $d\_v$ its associated degree of certainty.

The parameters of the generated queries are chosen to ensure that they are always failing for all query sizes on all datasets: queries select tuples with values below 0.1 for each dimension and a certainty degree above 0.9.

The number of MFSs and XSSs being a decisive parameter for the runtime of the algorithms, the three possible distributions allowed by the generator display dissimilar characteristics in that regard. Table IV gives an overview of these dissimilarities. As expected, the number of MFSs increases with the number of dimensions due to the exponential increase of the number of subqueries. Additionally, the anti-correlated and the independent datasets turn out to be the most challenging as the number of MFSs sharply increases with the number of dimensions. In the correlated dataset, the number of MFSs remains comparatively quite small due to the nature of the distribution, where low values in all dimension – as requested by the original query – are more likely present.

Table IV also illustrates the evolution of the number of MFSs with respect to the size of datasets $n$ ($d$ = 18). For correlated datasets, the number of MFSs remains constant at 153 since generated data are similar and happen not to change the failing status of subqueries. For the anti-correlated and independent generators, larger datasets necessarily have more successful queries, and the number of MFSs therefore varies, with a tendency to increase.

In addition to these synthetic datasets, we also conducted experiments on three real datasets that are commonly used to evaluate skyline algorithms[2] [16]: NBA (statistics of basketball players during regular seasons), HOUSE (money spent in one year by an American family for six different types of expenditures) and WEATHER (average monthly precipitation totals and elevation at over half a million sensor locations). Since these datasets do not include uncertainty values, certainty degrees were also randomly generated. The properties of these datasets are given in Table V. We can observe that the WEATHER dataset is the most challenging on all three criteria: cardinality, number of dimensions and number of MFSs.

**Experimental Setup.** Algorithms are implemented in Java 1.8 64 bits on top of PostgreSQL 9.5, MySQL 5.6.17 and Oracle 12c. Other Database Management Systems (DBMSs) supporting the CASE operator (required to compute the matrix) could also have been used. In this paper, we only report results for PostgreSQL since other DBMSs provide a comparable performance. Our prototype has been made available[3] with a tutorial to reproduce this evaluation.

Our experiments were conducted on a Linux-Ubuntu Server 16.04 LTS system with Intel XEON CPU E5-2630 v3 @2.4 GHz CPU and 8 GB RAM.

In the following, execution times are the average of five consecutive runs of the algorithms. To prevent a cold start effect, each algorithm is executed once before the actual measurements.

**Experiment 1: scalability w.r.t query size.** Figures 4, 5 and 6, illustrate the time to compute MFSs and XSSs using various query sizes (i.e., number of dimensions) for each synthetic dataset of 1M rows.

For the correlated dataset (Figure 4), all four algorithms provide response times in the order of a few milliseconds for small queries ($\leq$ 13 predicates), but MBS and MCS display a rapid increase of their response time when the query includes more than, respectively, 16 and 13 predicates. Comparatively, LBA takes more time to compute MFSs for small queries, but scales better by providing acceptable response times for queries with up to 25 predicates. As MCS explores a large part of the

[2]These real datasets are available at https://github.com/seanchester/SkyBench

[3]At http://www.lias-lab.fr/forge/projects/mfs4udb.

TABLE IV
STATISTICS OF SYNTHETIC DATASETS

| Dataset Dimension | | #MFSs and #XSSs vs query size, $n$= 1M | | | | | | | | | | | #MFSs and #XSSs vs dataset size, $d$= 18 | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 5 | 7 | 9 | 11 | 13 | 15 | 17 | 19 | 21 | 23 | 25 | 0.5 | 1 | 2 | 4 | 8 |
| Correlated | MFS | 10 | 21 | 36 | 55 | 78 | 105 | 136 | 171 | 210 | 253 | 300 | 153 | 153 | 153 | 153 | 153 |
| | XSS | 5 | 7 | 9 | 11 | 13 | 15 | 17 | 19 | 21 | 23 | 25 | 18 | 18 | 18 | 18 | 18 |
| Anti-correlated | MFS | 9 | 32 | 70 | 135 | 231 | 372 | 547 | 761 | 1040 | 1384 | 1791 | 724 | 645 | 617 | 832 | 1445 |
| | XSS | 8 | 15 | 24 | 38 | 56 | 83 | 137 | 211 | 294 | 387 | 517 | 113 | 175 | 307 | 444 | 535 |
| Independent | MFS | 4 | 20 | 51 | 107 | 194 | 331 | 577 | 950 | 1380 | 1987 | 2729 | 566 | 747 | 1698 | 2747 | 2804 |
| | XSS | 7 | 21 | 46 | 88 | 157 | 253 | 368 | 517 | 692 | 906 | 1136 | 296 | 432 | 552 | 555 | 472 |

subquery lattice each time a failing query is found, it does not scale well for queries that have more than 13 predicates and requires an important processing time, despite the fact that it executes less queries than LBA. Our approach MDMB outperforms all three others especially for large queries. This is mainly due to the use of dualization to compute the set of MFSs, which is highlighted by the comparison with MBS. The dualization avoids an in-memory representation of the lattice – such as in MBS – which significantly reduces the response time.

For the anti-correlated (Figure 5) and independent datasets (Figure 6), MBS gives better results then MCS and LBA for small queries but displays, as in the correlated datasets, a rapid increase in response time for queries of more then 16 predicates. As the subquery matrix is comparatively larger (since there are more XSSs), its computation takes more time. Another consequence of the augmentation of the number of MFSs is that LBA needs to execute much more queries than with the correlated dataset. Our approach MDMB outperforms all three others especially for large queries and yet does not exceed 1.6 seconds even on the most challenging dataset with the largest query ($d$=25).

Since both approaches share the same pre-processing steps up to the computation of XSSs, the difference between MBS and MDMB illustrates the importance of using dedicated dualization algorithms to compute MFSs. Indeed, for large queries on all three datasets, dualization allows MDMB to keep on providing acceptable response times where MBS did not.

**Experiment 2: scalability w.r.t dataset size.** Figures 7, 8 and 9 illustrate the time to compute the MFSs and XSSs as a function of the dataset size for respectively the correlated, anti-correlated and independent datasets ($d$ = 18). Results with MCS do not appear in these figures due to its substantial response time (respectively 279s, 372s and 275s $n$=0.5M). This experiment confirms the previous results. On the correlated datasets, MDMB, LBA and MBS scale well and only require a few milliseconds to about 1 second to terminate even on a dataset of 8M rows. The impact of our proposition is more significant on the independent and anti-correlated datasets where MDMB outperforms other algorithms. As these datasets have more MFSs, the LBA and MCS algorithms need to execute more queries, which induces a significant increase of their execution times on large datasets. Comparatively, MBS and MDMB only need to execute a single query to compute their matrix. MDMB outperforms the other approaches in these experiments. However it does not scale well with large datasets (8M). This is mainly explained by the quadratic complexity of the XSS computation as the matrix becomes larger, but also by the cost of dualization due to a larger number of XSSs and MFSs.

TABLE V
STATISTICS OF REAL DATASETS

| Dataset | Cardinality | Dimensions | #MFSs |
|---|---|---|---|
| NBA | 17 264 | 8 | 7 |
| HOUSE | 127 931 | 6 | 13 |
| WEATHER | 566 268 | 15 | 48 |

TABLE VI
PERFORMANCE ON REAL DATASETS

| Algorithm | NBA | | HOUSE | | WEATHER | |
|---|---|---|---|---|---|---|
| | msec. | speedup | msec. | speedup | sec. | speedup |
| LBA | 7.2 | 4.5× | 13.4 | 5.15× | 0.19 | 9.5× |
| MCS | 1.4 | 0.88× | 8.8 | 3.38× | 4.45 | 222.5× |
| MDMB | 1.6 | - | 2.6 | - | 0.02 | - |
| MBS | 0.6 | 0.38× | 1.4 | 0.54× | 0.10 | 5× |

**Experiment 3: applicability to real datasets.** Table VI gives the performance of the algorithms on three real datasets. Since NBA and HOUSE are relatively small and have few dimensions, we do not obtain significant performance gains with our new approach MDMB compared to MBS. This can be explained by the small size of both the lattice and the datasets which reduces the response time and the search space of MBS. On WEATHER, the most challenging real dataset, we observe that MDMB, MBS and LBA provide the MFSs and XSSs of the queries in a few milliseconds. As with synthetic datasets, MCS executes less queries than LBA but still needs more than 4 seconds to return the results as it explores several times a large part of the subquery lattice, which has $2^{15}$ nodes. Even if the size of this dataset is still small compared to the synthetic ones, MDMB outperforms other algorithms by a factor of $5\times$ to $222\times$.

## VII. CONCLUSION

In this paper, we have proposed an efficient approach, called MDMB, that computes both the MFSs and XSSs of failing threshold queries over uncertain databases. The key notion of MDMB is the dualization concept borrowed from graph theory. As shown in the experimental evaluation, this

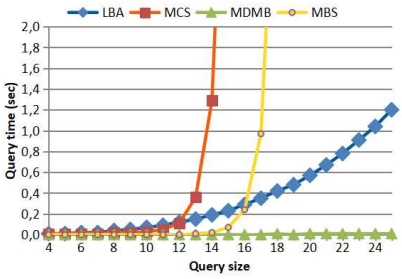


Fig. 4. time vs query size, corr, n=1M

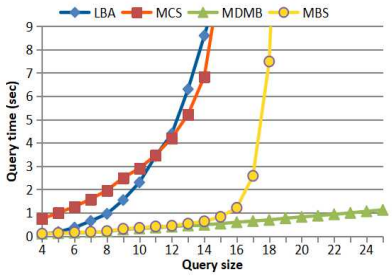


Fig. 5. time vs query size, anticorr, n=1M

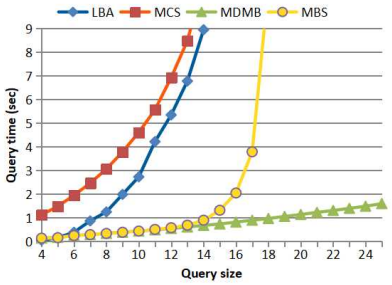


Fig. 6. time vs query size, indep, n=1M

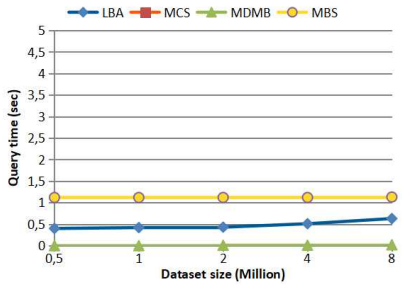


Fig. 7. time vs dataset size, corr, d=18

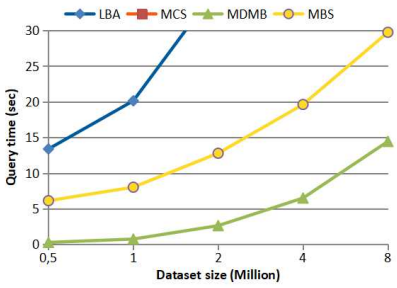


Fig. 8. time vs dataset size, anticorr, d=18

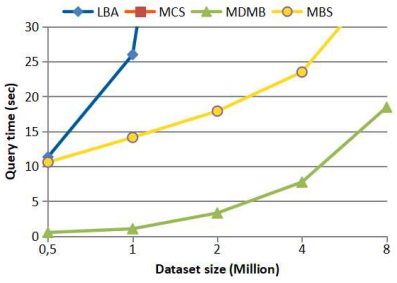


Fig. 9. time vs dataset size, indep, d=18

modification greatly improves the computation time over our previous MBS approach [3].

We have also compared MDMB with two algorithms from the state of the art: LBA and MCS. MCS can only be used for relatively small datasets and queries with few predicates as it needs to explore several times a large part of the subquery lattice, while LBA and MBS return the MFSs and XSSs in a few milliseconds on real datasets and on the less challenging synthetic datasets. The benefit of MDMB appears when using large input queries, where LBA and MBS do not provide acceptable performance for queries with 16 predicates or more, while MDMB gives good response time even for the most challenging queries.

## Acknowledgment

We would like to acknowledge the worthy assistance and technical support received from Mickaël Baron, Research Engineer at ISAE-ENSMA.

## References

[1] H. Mannila and H. Toivonen, "Levelwise search and borders of theories in knowledge discovery," *Data Min. Knowl. Discov.*, vol. 1, no. 3, pp. 241–258, 1997.

[2] P. Godfrey, "Minimization in Cooperative Response to Failing Database Queries," *International Journal of Cooperative Information Systems*, vol. 6, no. 2, pp. 95–149, 1997.

[3] C. Belheouane, S. Jean, A. Hadjali, and H. Azzoune, "Handling failing queries over uncertain databases," in *2017 IEEE International Conference on Fuzzy Systems, FUZZ-IEEE 2017, Naples, Italy, July 9-12, 2017*, 2017, pp. 1–6.

[4] K. Murakami and T. Uno, "Efficient algorithms for dualizing large-scale hypergraphs," *Discrete Applied Mathematics*, vol. 170, pp. 83–94, 2014.

[5] G. Fokou, S. Jean, A. Hadjali, and M. Baron, "Handling Failing RDF Queries: From Diagnosis to Relaxation," *Knowledge and Information Systems (KAIS)*, vol. 50, no. 1, pp. 167–195, 2017.

[6] I. Dellal, S. Jean, A. Hadjali, B. Chardin, and M. Baron, "On addressing the empty answer problem in uncertain knowledge bases," in *Database and Expert Systems Applications - 28th International Conference, DEXA 2017, Lyon, France, August 28-31, 2017, Proceedings, Part I*, 2017, pp. 120–129.

[7] D. McSherry, "Incremental Relaxation of Unsuccessful Queries," in *Advances in Case-Based Reasoning*, 2004, vol. 3155, pp. 131–148.

[8] D. Jannach, "Fast Computation of Query Relaxations for Knowledge-based Recommenders," *AI Communications*, pp. 235–248, 2009.

[9] O. Pivert and G. Smits, "How to efficiently diagnose and repair fuzzy database queries that fail," in *Fifty Years of Fuzzy Logic and its Applications*, 2015, pp. 499–517.

[10] P. Bosc, O. Pivert, and H. Prade, "An uncertain database model and a query algebra based on possibilistic certainty," in *SoCPaR*, 2010.

[11] P. Sen, A. Deshpande, and L. Getoor, "PrDB: Managing and Exploiting Rich Correlations in Probabilistic Databases," *The VLDB Journal*, vol. 18, no. 5, pp. 1065–1090, 2009.

[12] O. Pivert and H. Prade, "A certainty-based model for uncertain databases," *IEEE Transactions on Fuzzy Systems*, pp. 1181–1196, 2015.

[13] D. Dubois and H. Prade, "Necessity measures and the resolution principle," *IEEE Trans. on Systems, Man and Cybernetics*, 1987.

[14] D. Gunopulos, R. Khardon, H. Mannila, S. Saluja, H. Toivonen, and R. S. Sharm, "Discovering all most specific sentences," *ACM Trans. Database Syst.*, vol. 28, no. 2, pp. 140–174, 2003.

[15] S. Börzsönyi, D. Kossmann, and K. Stocker, "The skyline operator," in *ICDE*, 2001, pp. 421–430.

[16] S. Chester, D. Šidlauskas, I. Assent, and K. S. Bøgh, "Scalable Parallelization of Skyline Computation for Multi-core Processors," in *ICDE 2015*, 2015, pp. 1083–1094.