Worst-case Analysis for Interactive Evaluation of Boolean Provenance

Antoine Amarilli
LTCI, Télécom Paris, Institut Polytechnique de Paris

Yael Amsterdamer
Bar-Ilan University

Abstract

In recent work, we have introduced a framework for fine-grained consent management in databases, which combines Boolean data provenance with the field of interactive Boolean evaluation. In turn, interactive Boolean evaluation aims at unveiling the underlying truth value of a Boolean expression by frugally probing the truth values of individual values. The required number of probes depends on the Boolean provenance structure and on the (a-priori unknown) probe answers. Prior work has analyzed and aimed to optimize the *expected* number of probes, where expectancy is with respect to a probability distribution over probe answers. This paper gives a novel *worst-case* analysis for the problem, inspired by the decision tree depth of Boolean functions.

Specifically, we introduce a notion of *evasive provenance expressions*, namely expressions, where one may need to probe all variables in the worst case. We show that read-once expressions are evasive, and identify an additional class of expressions (acyclic monotone 2-DNF) for which evasiveness may be decided in PTIME. As for the more general question of finding the optimal strategy, we show that it is coNP-hard in general. We are still able to identify a sub-class of provenance expressions that is "far from evasive", namely, where an optimal worst-case strategy probes only $\mathrm{O}(\log n)$ out of the n variables in the expression, and show that we can find this optimal strategy in polynomial time.

1 Introduction

There is a large body of work on Boolean provenance for database queries (e.g., [4, 5, 10, 11, 16, 25, 26, 35]). The basic paradigm is that one starts by associating a Boolean variable with every tuple in the input database; then, for queries in a given language (say, Select-Project-Join-Union), we explain how the query operators can be extended to annotate their results with Boolean expressions depending on the input tuples. For instance, for the join operation, a tuple obtained by the joining two tuples annotated by x and y will be annotated by $x \wedge y$. If we union two relations and the same tuple occurs in both with annotations by x and y, then the tuple in the (set) union of the two relations will be annotated by $x \vee y$. A basic property of such provenance structures, dating back to [26], is that these provenance expressions describe the *Possible Worlds* that make the query true: a truth valuation v of the Boolean variables satisfies the provenance expression annotating a tuple t if and only if t appears in the query result when running the query over the sub-database consisting only of the tuples annotated by variables that v maps to True.

Building on these provenance structures, we have proposed in recent work a framework for consent management in shared databases [17]. The setting is such that multiple peers share data but do not necessarily agree upfront to every kind of use of their data – rather, explicit permission is needed when the data is to be used in a particular way, e.g., shared with a third party. This gets more complex when data is transformed via queries: when we wish to use a query result, it is challenging to determine whose consent should be asked, because the result may rely on multiple tuples originally contributed by multiple peers. The model proposed in [17] follows the same possible worlds semantics that we have mentioned above: namely, consent is granted with respect to an output tuple t if and only if, on the sub-database of the input database containing all tuples for which consent was granted, the output t is part of the query result. This means that the problem of consent management reduces to the problem of *Interactive Evaluation of Boolean provenance expressions*, i.e., that of determining the truth value of these expressions by probing individual variables to reveal their truth values. Interactive Boolean Evaluation has been extensively studied, and has mostly focused on the following problem: given probabilities on the truth values of the Boolean variables (namely, the probability of obtaining an affirmative probe answer for each variable), we must determine a probing strategy (i.e., which variables to probe and in what order) so that the expected number of probes is minimized. There is a significant body of work on the topic (e.g., [2, 6, 8, 13, 15, 17, 29, 31, 44]) and in [17] we have shown how to leverage this work for our application of consent management.

In this paper, we start from Boolean provenance and study the complexity of interactively evaluating it, but focus on a different optimization goal. Namely, rather than optimizing the expected number of probes, we focus on optimizing the worst-case number of probes, i.e., minimizing the maximal number of probes that may be needed on *any* possible sequence of probe answers. In contrast to expected-case analysis, worst-case analysis requires no probability estimation for probe answers, and is useful when one aims for "cautious" strategies in terms of the number of probes.

Example 1.1. *Consider a database with two tuples, one has the Boolean provenance $x \wedge y$ and the other has the provenance $x \vee z$ (for now, regardless of how this database could be created). If we probe x first, i.e., discover its truth value, we will make at most two probes in total: if x is True we need not probe z to evaluate the second expression; and if x is False we need not probe y to evaluate the first expression. This strategy optimizes the worst-case number of probes for the given provenance. In contrast, a non-optimal strategy would probe y and z first and in the worst case would require additionally querying x to reach full evaluation.*

A natural way to capture interactive evaluation strategies is by representing the Boolean expression as a *Binary Decision Diagram* [9]. Intuitively, BDDs are acyclic graphs where each (inner) node corresponds to a variable to be probed next, where edges correspond to possible probe answers, and where leaves give the result of evaluating the function. The worst-case number of probes for a given strategy is the depth (i.e. longest root-to-leaf path) of the BDD representation of the strategy. Worst-case optimization for interactive evaluation then amounts to finding a BDD with minimal depth for the given provenance expressions. We can always find a trivial BDD where this depth is the number of variables of the expression, by asking about all variables, but in some cases we may be able to achieve a lower depth.

We define two related decision problems. We first ask: can we tell, by looking at a given provenance expression, whether the worst-case number of probes is simply the number of variables in the expression; or equivalently, is there no BDD for the expression whose depth is smaller than the number of variables? Following existing work on Boolean functions [46, 12], we call *evasive* the provenance expressions that have this trait, i.e., that require probing all their variables in the worst case, and call the decision problem `DEC-BDD-EVASIVE`. Evasiveness implies a lower bound for the worst-case performance of probing strategies: if a provenance expression is evasive, then there is no hope of improving on the trivial BDD that queries all variables in some order. Second, we introduce the problem `OPT-BDD-DEPTH` of finding the optimal strategy, namely the BDD with minimal depth for given provenance expressions. We also study its phrasing as a decision problem, called `DEC-BDD-DEPTH`, where we must decide whether the depth of the optimal BDD for the expression (which we call *the expression depth*) is less than some given k: this problem generalizes `DEC-BDD-EVASIVE`. We study these problems for multiple classes of provenance expressions of interest. Our main complexity results are the following.

General Provenance Expressions We start by considering general provenance expression, which in particular may involve negation (this can be generated for queries with negation/difference). We show that `DEC-BDD-DEPTH` and `OPT-BDD-DEPTH` are coNP-hard. Since these problems are intractable in the most general settings, we then turn to analyzing sub-classes of interest.

Read-once Provenance Multiple lines of previous work have established the importance of *read-once provenance*, namely expressions where no variable occur more than once. Generalizing to multiple provenance expressions (of multiple output tuples), we may distinguish between sets of expressions that are *overall read-once* (namely no variable occur twice even across expressions) and sets where each individual expression is read-once but variables may reoccur in different expressions. For overall read-once provenance expressions, or ones that may be written in an equivalent overall read-once form, we show that they are always evasive. This means that we cannot improve the worst-case probing complexity of these expressions. By contrast, we show that read-once provenance is not always evasive when it is not overall read-once.

Monotone Provenance A class of queries whose provenance is often studied is that of SPJU (Select-Project-Join-Union queries). For such queries, the provenance includes no negation and may be represented as a monotone DNF formula. It is open whether our decision problems are tractable for this class, and we show that the problem is far from trivial. Specifically, we show that there exists a sequence of such provenance expressions that are “far from evasive”, namely for which there is an optimal probing strategy that is exponentially

Acquisitions

Acquired	Acquiring	Date	
A2Bdone	Zazzer	7/11/2020	a_0
microBarg	Fiffer	1/5/2017	a_1
fPharm	Fiffer	1/2/2016	a_2
Optobest	microBarg	8/8/2015	a_3

Education

Alumni	Institute	Year	
Usha Koirala	U. Melbourne	2017	e_0
Pavel Lebedev	U. Melbourne	2017	e_1
Nana Alvi	U. Sau Paolo	2010	e_2
Nana Alvi	U. Melbourne	2017	e_3
Gao Yawen	U. Sau Paolo	2010	e_4
Amaal Kader	U. Cape Town	2005	e_5

Roles

Organization	Role	Member	
A2Bdone	Founder	Usha Koirala	r_0
A2Bdone	Founding member	Pavel Lebedev	r_1
A2Bdone	Founding member	Nana Alvi	r_2
microBarg	Co-founder	Nana Alvi	r_3
microBarg	Co-founder	Gao Yawen	r_4
microBarg	CTO	Amaal Kader	r_5

Table 1: Example annotated database.

better than the trivial strategy: it only probes $\mathrm{O}(\log n)$ variables in the worst case, where n is the number of variables in the expression. For this particular sequence, the optimal strategy may be efficiently computed.

Acyclic monotone 2-DNF Provenance We finally identify a sub-class of interest, which we call *acyclic monotone 2-DNF*. These are monotone 2-DNF formulas, where the graph obtained by treating each term as an edge between its variables is acyclic. For this class, we show a polynomial-time algorithm to check whether a given expression is evasive. The algorithm is based on a non-evasiveness pattern, namely a pattern that occurs in the graph if and only if the corresponding expression is non-evasive.

The rest of this paper is organized as follows. We start with an overview of our model in Section 2. Our formal results and proofs are detailed in Section 3. We overview related work in Section 4 and conclude in Section 5.

2 Model

We introduce the model for the problems that we study in this paper. We start by recalling the notion of Boolean provenance as well as a representation form for Boolean expressions called Binary Decision Diagrams that will be useful in our proofs. We then define the decision problems that we study in the remainder of the paper.

2.1 Boolean Provenance

We assume familiarity with standard relational database terminology [1]. Let X be a set of Boolean variables. We recall the notion of an annotated database [25, 26], a relational database where each tuple is annotated by a variable from X (also called X-database). In our context, the Boolean variables stand for tuple consent.

```
SELECT DISTINCT a.Acquired, e.Institute
FROM Acquisitions AS a, Roles AS r, Education AS e
WHERE a.Acquired = r.Organization AND
      r.Member = e.Alumni AND a.Date >= 2017.01.01 AND
      r.Role LIKE '%found%' AND e.YEAR <= year(a.Date)
```

Figure 1: Query over the example database

Definition 2.1 (Annotated database)**.** *An* annotated database $\bar{D} = (D, L)$ *consists of a relational database D and of a labeling function L :* $\mathrm{tuples}(D) \mapsto X$ *mapping each tuple t in D to a variable* $\mathrm{L}(t) \in X$*. We will denote $n := |X|$.*

The premise is that there is a hidden truth associated with each variable, i.e., a ground truth correctness for the corresponding tuple.

Definition 2.2 (Truth Valuation)**.** *A* valuation *is a function* $\mathrm{val} : X \mapsto \{\mathit{True}, \mathit{False}\}$*. Given an annotated database $\bar{D} = (D, L)$, denote by $D_{\mathrm{val}} := \{t \in D \mid \mathrm{val}(L(t)) = \mathit{True}\}$.*

Example 2.3. *Table 1 outlines an annotated database with three relations:* Acquisitions*, including data on companies acquired by other companies;* Roles*, including data on roles of different organization members; and* Education*, including data on university alumni. The right-most column shows the variable in X annotating the tuple, and standing for the event that consent is granted for using this tuple (in response to a specific usage request).*

The provenance annotations of the input database can be propagated to the query output, so that Boolean expressions annotating tuples in the query output reflect the possible worlds of the input annotated database for which the output tuple appears in the query result. In particular, Select-Project-Join-Union (SPJU) queries are known to yield provenance in the form of monotone Boolean expressions (without negation), which are computable in DNF (i.e., as disjunctions of conjunctions) in PTIME [26]. When relational difference is allowed (SPJUD queries), the provenance expressions may also include negation [4] and may no longer be in DNF.

For a Boolean expression φ using only variables from X, we will denote by $\mathrm{val}(\varphi)$ the truth value resulting from replacing each occurrence of a variable x in φ with its truth value $\mathrm{val}(x)$. We can then define:

Definition 2.4 (Provenance for Query results)**.** *Given an annotated database $\bar{D} = (D, L)$ and a query Q over D, the query result is $Q(\bar{D}) = (Q(D), L')$ where $Q(D)$ is the standard query result and $L' : Q(D) \to \mathrm{Bool}[X]$ where $\mathrm{Bool}[X]$ is the semiring of all Boolean expressions over X [4, 25]. For every $t \in Q(D)$, and every valuation* val *over X, we require that the expression $\varphi = L'(t)$ is such that $\mathrm{val}(\varphi) = \mathit{True}$ if and only if $t \in D_{\mathrm{val}}$ (Def 2.2).*

Example 2.5. *Consider the SPJU query in Figure 1 over the DB in Table 1, which returns companies acquired since 2017 along with institutes in which founders of these companies had studied. The query results are shown in Figure 2 along with their Boolean provenance expressions. In the event that the first two tuples in the Acquisitions relation are assigned False ($\mathrm{val}(a_0) = \mathrm{val}(a_1) = \mathit{False}$) there are no query results (With consent) since the join of the Acquisitions table with the other tables would necessarily be empty. If, alternatively, the first tuple*

Acquired	Institute	
A2Bdone	U. Melbourne	$(a_0 \wedge r_0 \wedge e_0) \vee (a_0 \wedge r_1 \wedge e_1) \vee (a_0 \wedge r_2 \wedge e_3)$
A2Bdone	U. Sau Paolo	$(a_0 \wedge r_2 \wedge e_2)$
microBarg	U. Melbourne	$(a_1 \wedge r_3 \wedge e_3)$
microBarg	U. Sau Paolo	$(a_1 \wedge r_3 \wedge e_2) \vee (a_1 \wedge r_4 \wedge e_4)$

Table 2: Result of the example query.

of each of the three relations is assigned True (i.e., $\text{val}(a_0) = \text{val}(r_0) = \text{val}(e_0) =$ *True), the query first result tuple* `(A2Bdone, U. Melbourne)`*, derived from these input tuples, has consent.*

2.2 Interactive Evaluation via BDDs

We focus here on *evaluating* provenance expressions, namely revealing the truth values of expressions by probing individual variables to gradually reveal the underlying valuation. In the settings of [17], probes correspond to asking tuple owners for their consent to use a tuple that they have contributed to a shared database. Naturally, the choice of which variables to probe is non-trivial, and different sequences of probes may lead to revealing the truth values of the provenance expressions much faster than others.

Example 2.6. *Consider the first tuple of the query result in Table 2. Probing* a_0 *corresponds to asking consent to use the first tuple of the Acquisitions relation. If the answer is negative, then* $\text{val}(a_0) =$ *False and we can conclude that the first result tuple is assigned False (in the context of consent management, we do not have consent to use this tuple). If by contrast* $\text{val}(a_0) =$ *True then we still do not know the consent status with respect to the result tuple, and we need to continue probing. There are many other probing strategies, e.g., we may start by probing* $\text{val}(a_1)$*.*

A deterministic strategy of which variable to probe, based on the truth values revealed so far, can be generally modelled in terms of a Boolean Decision Diagram (BDD). In a BDD, each inner node stands for the current variable to probe and its outgoing edges reflect the strategy for the next steps if the observed value (probe answer) is True and False respectively. The leaves reflect the truth value by the end of the evaluation. We state the definition in full generality for a set of Boolean functions.

Definition 2.7 (Binary Decision Diagram (BDD))**.** *A Boolean expression* $\varphi \in \text{Bool}[X]$ *is* constant *equal to* b*, for* $b \in \{\mathit{True}, \mathit{False}\}$*, if for every valuation* val *of* X*, we have* $\text{val}(\varphi) = b$*.*

For $x \in X$ *and* $b \in \{\mathit{True}, \mathit{False}\}$*, the* instantiation of x to b in φ*, written* $\varphi_{x=b}$*, is the Boolean function on* $X \setminus \{x\}$ *obtained by assigning* b *to (all occurrences of)* x *in* φ*. We generalize this to a set of Boolean functions* Φ *by* $\Phi_{x=b} = \{\varphi_{x=b} \mid \varphi \in \Phi\}$*.*

A BDD on a non-empty set of Boolean functions $\Phi \subseteq \text{Bool}[X]$ *is a labeled directed acyclic graph (DAG)* $G = (V, E, L_V)$ *and distinguished* root node $v \in V$*, where* V *is a set of nodes labeled by* $L_V : V \to X \cup (\Phi \to \{\mathit{True}, \mathit{False}\})$ *and* $E \subseteq V \times V \times \{\mathit{True}, \mathit{False}\}$ *is a set of directed edges labeled by a Boolean value. The DAG is inductively defined as follows.*

- *If Φ has at least one variable x, a BDD for Φ may consist of a root node v with $L_V(v) = x$, an outgoing edge labeled by True and connecting v to the root of a BDD for $\Phi_{x=\mathit{True}}$, and similarly an outgoing edge and child BDD for $x =$ False. The two child BDDs can share some nodes and edges.*

- *Alternatively, a BDD for Φ may be a single root node $\{v\}$. In this case, we require that each function in Φ is a constant Boolean expression, and $L_V(v)$ is a function assigning each $\varphi \in \Phi$ to its constant value in $\{\mathit{True}, \mathit{False}\}$.*

For convenience, when $\Phi = \{\varphi\}$ is a singleton, we identify the label $\varphi \mapsto b$ on each leaf with the constant value b it assigns to the single formula φ. (End of Def. 2.7.)

In the present work, we are interested in the number of variables that need to be probed in the worst-case, as reflected by the depth of the BDD.

Definition 2.8 (Depth)**.** *Given a BDD $G = (V, E, L_V)$, the* depth *of the BDD is the maximal number of edges on a directed path in G from the root node to a leaf. If $|V| = 1$ then the depth is 0.*

Given a set of Boolean expressions $\Phi \subseteq \mathrm{Bool}[X]$, its depth is defined as the minimal depth of any BDD for Φ.

An immediate observation is that if we restrict ourselves to strategies that do not make obviously useless probes, i.e., do not issue the same probe twice, then a BDD on $\Phi \subseteq \mathrm{Bool}[X]$ has depth at most $n = |X|$. We call a set of Boolean expressions as *evasive* if its depth is n, i.e., if there is no BDD with a better depth than that of the naive BDD querying all variables in order.

Example 2.9. *The AND function $x_1 \wedge \cdots \wedge x_n$ is evasive. Any decision tree for this function has height n, since in particular we must observe all the variables to establish that their conjunction is True. OR is also evasive, by similar arguments.*

Example 2.10. *$\psi_0 = (w \wedge x) \vee (x \wedge y) \vee (y \wedge z)$ is non-evasive. A strategy that first observes x and y will at most observe 3 variables: if $\mathrm{val}(x) = \mathrm{val}(y) =$ True, the evaluation is complete. Otherwise, assume w.l.o.g that $\mathrm{val}(x) =$ False; then $w \wedge \mathit{False} = \mathit{False}$, so we do not need to observe w. The same reasoning applies to z if $\mathrm{val}(y) =$ False.*

Problem definition We are given a set of Boolean expressions $\Phi \subseteq \mathrm{Bool}[X]$ (where $|X| = n$). The `OPT-BDD-DEPTH` problem is defined as the optimization problem of finding a strategy that corresponds to a BDD of minimal depth for Φ. The corresponding decision problem, `DEC-BDD-DEPTH`, is defined as deciding whether the depth of Φ is $\leq k$, for a given $k < n$. Finally, `DEC-BDD-EVASIVE` is the problem of deciding whether Φ is evasive.

Observe that we can reduce `DEC-BDD-EVASIVE`, in PTIME, to the complement of `DEC-BDD-DEPTH`, by deciding `DEC-BDD-DEPTH` for $k = n - 1$, and taking the inverse. In contrast, `DEC-BDD-DEPTH` may not reduce in PTIME to `OPT-BDD-DEPTH`, since this depends on the representation of the optimal strategy: the size of a BDD may be exponential in n so computing the value for `DEC-BDD-DEPTH` from the output of `OPT-BDD-DEPTH` is not in PTIME.

3 Results

We next detail our technical results. We start by considering the general case of arbitrary Boolean expressions, and show that deciding DEC-BDD-DEPTH is generally intractable. We then turn to the case of read-once provenance, extensively studied in the context of, e.g., probabilistic databases [27, 39, 41]. We show that overall read-once Boolean provenance is always evasive. Then, we show a class of monotone Boolean expression that are "far" from evasive, and for which we can provide an optimal probing strategy. Finally, we identify a class of provenance expressions called monotone acyclic graph DNFs, for which we can decide evasiveness in PTIME.

3.1 General Provenance Expressions

We first observe that DEC-BDD-DEPTH is intractable when given a non-monotone Boolean formula, already for CNF and DNF.

Proposition 3.1. *Deciding DEC-BDD-DEPTH is coNP-hard, even if the input Boolean expression is in DNF/CNF and the depth upper bound is $k = 0$.*

Proof. We first show this for CNFs. We reduce from the NP-hard problem of deciding, given a CNF φ, if φ is satisfiable, i.e., it is not the constant False function. To do this, first check if φ is falsifiable, which is easily done in PTIME by checking that it has one non-trivial clause. If φ is not falsifiable, answer that φ is (vacuously) satisfiable. Otherwise, if φ is falsifiable, then it is satisfiable iff it not constant, i.e., if its depth is greater than 0. We have thus reduced the NP-hard problem of Boolean satisfiability for a CNF to the complement of DEC-BDD-DEPTH, establishing that DEC-BDD-DEPTH is coNP-hard. Similarly, for DNF we can check in PTIME if the expression is satisfiable, and then the DNF is falsifiable iff the expression is not a constant True, i.e., its depth is 0. □

This also implies that the problem OPT-BDD-DEPTH is coNP-hard, since if we got an optimal 0-depth BDD (which, unlike general BDDs, can be compactly represented) we could decide that the depth is ≤ 0. In contrast, this still does not imply the hardness of verifying evasiveness, which remains open.

3.2 Read-once Provenance

Several works in the field of Probabilistic Databases [27, 39, 41] have studied queries that yield *read-once* provenance (or provenance that can be compiled to read-once form), i.e., where each variable occurs at most once in each provenance expression. The reason is that read-once expressions enable easy computation of probabilities. The work of [17] further lists cases when the provenance has *overall* read-once form, i.e., when variables do not repeat across the provenance expressions of different output tuples.

In our context, we show that overall read-once sets of Boolean expressions are evasive. We call a read-once expression *non-simplifiable* if it is either the constant False or True, or contains no occurrences of constants. Indeed, if that is not the case, then we can simplify the expression further by rules such as False $\wedge\, \varphi =$ False, yielding a smaller read-once expression.

Example 3.2. *Recall the provenance in Table 2. The second and third expressions are non-simplifiable read-once expressions, i.e., they contain no constants and no variable repetitions (also across these two expressions). The first expression is not read-once, as a_0 repeats in every term; but it is equivalent to the read-once expression $a_0 \wedge ((r_0 \wedge e_0) \vee (r_1 \wedge e_1) \vee (r_2 \wedge e_3))$; and similarly for the fourth expression. Finally, the entire query result is not overall read-once since variables such as a_0, a_1, r_3 repeat across Boolean expressions.*

We can show the following by a recursion on the structure of the Boolean expression.

Proposition 3.3. *Given a set of Boolean expressions Φ over a set of variables X, if Φ is overall read-once and each $\varphi \in \Phi$ is non-simplifiable, then Φ is evasive.*

Note an easy corollary: sets of Boolean expressions that are not overall read-once but are equivalent to an overall read-once set are also evasive, as evasiveness is preserved under equivalence.

Proof. We prove this by induction on the number of variables in Φ. The key observation is that for any non-simplifiable read-once Boolean formula φ with n variables, for every variable x of that formula, then one of $\varphi_{x=\mathrm{True}}$ and $\varphi_{x=\mathrm{False}}$ is a non-simplifiable read-once Boolean formula with $n-1$ variables. Indeed, if the one occurrence of x is positive, then if it is as part of a $\vee$-operation (resp., a $\wedge$-operation), then replacing x by False (resp., by True) and simplifying yields a Boolean function that is still read-once, is clearly non-simplifiable, and has $n-1$ variables. If the one occurrence of x is negative, then we do the same but replacing it respectively by True and False instead. Since Φ is overall read-once, we can perform this induction for each of its expressions separately. □

Read-once provenance is not necessarily evasive if variables may be repeated *across* expressions: reconsider Example 1.1, and observe that each expression is non-simplifiable read-once, and x repeats across expressions. We have shown that probing one variable can always be avoided, i.e., the expression set is not evasive.

3.3 Monotone Boolean Expressions

Next, we consider monotone Boolean expressions, without negation. These expressions are of particular interest, since we can show a two-way correspondence between the provenance of SPJUs and the class of monotone Boolean k-DNF formulas, where every term (conjunction) consists of at most k (unique) variables.

Proposition 3.4.

1. *For each SPJU query Q there exists a value k such that for every X-database $\bar{D}$, the provenance of each tuple in $Q(\bar{D})$ may be represented in a monotone k-DNF form. This form may further be constructed in PTIME in data complexity (i.e., as a function of $\bar{D}$).*

2. *Conversely, for every monotone k-DNF formula φ, there exists an SPJ query Q depending only on k, an X-database $\bar{D}$ whose size is linear in φ such that the query output $Q(\bar{D})$ is a singleton tuple whose provenance is equivalent to φ.*

Proof (sketch). The first part of the proposition holds with k being the maximal number of relations joined by a conjunctive query within Q. To observe that this is the case, note that conjunctions in the provenance construction are associated with joins and disjunctions are associated with projection and union.

For the second part, we again exploit the correspondence between query and Boolean operations. Given a monotone k-DNF formula φ, consider a DB $\bar{D}$ with two relations. Relation R encodes the variables of φ, where for each variable x in φ we have a tuple $R(x)$ (using x as a value) annotated by x. Relation S encodes the terms of φ such that for each term $x_1 \wedge x_2 \wedge \cdots \wedge x_k$ in φ we have a tuple $S(x_1, x_2, \ldots, x_k)$ (using $x_1, x_2, \ldots, x_k$ as values) and annotated by x_1. If a term is of size $< k$ we can repeat one of its variables to obtain an equivalent term of size exactly k. The query Q is a binary CQ fixed for k (having no unions, using only equality joins and a projection on all variables): $\mathrm{ans}() : -S(z_1, \ldots, z_k), R(z_1), \ldots, R(z_k)$. By this construction, each tuple in the join result corresponds to a term in φ and has a provenance of the form of a conjunction $x_1 \wedge x_2 \wedge \cdots \wedge x_k$, where every x_i stands for an original variable, and the repeated occurrence of x_1 by the join with S is absorbed. Then projecting out all the variables yields the disjunction of these conjunctions, which is equivalent to φ. $\square$

The hardness result of Prop. 3.1 does not immediately apply to the case of monotone functions, which we leave open. Let us instead study bounds that we can obtain in special cases. For instance, to prove that an expression in DNF is True, we need to prove, for one of its terms (conjunctions), that each of its variables are True; and similarly for CNF clauses for proving False. Therefore, the depth of a monotone expression is bounded from below by the maximum size (number of distinct variables) of a term in its DNF and of a clause in its CNF (assuming the expressions are simplified to avoid subsumed terms/clauses). As stated by the following theorem, there exists a class of Boolean expressions that have a depth linear in this bound, and exponentially smaller than the number of variables.

Theorem 3.5. *For arbitrarily large integers n, there is a monotone Boolean expression of size $\Theta(n)$, such that in DNF its term size (number of distinct variables in a conjunction) is at most $k = \mathrm{O}(\log n)$, and its depth is $2k - 1 = \mathrm{O}(\log n)$.*

Proof (sketch). Define the monotone DNF formula $\psi_0 = (w \wedge x) \vee (x \wedge y) \vee (y \wedge z)$ (as in Example 2.10) and recursively define $\psi_{i+1} = (u_i \wedge \psi_i) \vee (u_i \wedge v_i) \vee (v_i \wedge \psi'_i)$ where u_i, v_i are fresh variables and ψ'_i is obtained by consistently replacing the variables of ψ_i by fresh variables. In DNF, the resulting term size is $k = i + 2$. We can show that $|\mathrm{vars}(\psi_i)|$ is exponential in i since we double the expression size at each level, and that there exists a BDD ψ_i whose expected cost is $\mathrm{O}(i)$ (by probing first the u_i, v_i variables). $\square$

The theorem implies that exists a class of Boolean expressions that are "far" from evasive, namely, the optimal strategy probes only a logarithmic number of variables in the worst case. Moreover, the optimal number of probes for this class is close to the overall lower bound on the number of probes for any k-DNF. Our proof is constructive: not only do we show the existence of an optimal strategy, but the proof also provides the strategy itself.

3.4 Monotone Acyclic Graph DNFs

As a first step towards understanding monotone DNFs in general, we next focus on a subclass of Boolean expressions which we call *monotone acyclic graph DNFs*. The subclass is defined as follows:

Definition 3.6 (Monotone acyclic graph DNFs)**.** *A monotone graph DNF φ is a monotone DNF where each term includes at most two variables (i.e., 2-DNF). We identify φ to a graph G whose nodes are the variables of X and which has one edge $\{x, y\}$ per term $x \wedge y$. We say that φ is* acyclic *if G is. For convenience, we will often abuse notation and identify φ and G (but paying attention to the fact that G does not represent terms with one single variable), and we will often consider G as a rooted tree, by picking some root.*

Example 3.7. *The expression $\psi_0 = (w \wedge x) \vee (x \wedge y) \vee (y \wedge z)$, shown in Ex. 2.10 to be non-evasive, is a monotone acyclic graph DNF.*

Characterizing which queries guarantee this form of provenance is left open. However, given provenance of this shape, we can show the following.

Theorem 3.8. *Given a monotone acyclic graph DNF, deciding* `DEC-BDD-EVASIVE` *is in PTIME.*

We prove this theorem in the rest of the section. The high-level idea is that we identify a *non-evasiveness pattern*, defined recursively on the structure of the graph DNF. We explain that finding such a pattern is in PTIME, and that such a pattern is present iff the formula is non-evasive, leading to the conclusion of Theorem 3.8.

Our theorem holds for any acyclic monotone graph DNF, but for convenience, we will eliminate some anomalies that can be handled separately (and in PTIME), by assuming:

- All the variables of X occur in φ (otherwise, it is trivially non-evasive).
- The graph is connected (otherwise, the formula is evasive iff each connected component is).
- There are no subsumed terms, i.e., if we have a singleton term x, then there are no terms $x \wedge y$ (as these can be removed).

We now present the pattern that characterizes non-evasiveness:

Definition 3.9. *Let φ be a monotone acyclic graph DNF, and let x be a variable. Let T be the graph of φ represented as a tree rooted at x. For y a variable of φ, we denote by $\varphi_{x:y}$ the monotone acyclic graph DNF obtained by restricting φ to the variables in the subtree of T rooted at y, and the terms involving only these variables.*

A non-evasiveness pattern Π *rooted at x for φ is a labeled tree inductively defined as follows:*

- *If x is a variable that does not appear in any term, then Π is a single leaf node n labeled by x.*

- *Otherwise,* Π *has a root node* n *labeled with* x *and with the following children: for each variable* y *co-occurring in a term with* x *(i.e., every child of* x *in* T*), we choose a grandchild* w_y *of* y *(we require that such a grandchild exists), and* n *has a child node consisting of a non-evasiveness pattern rooted at* w_y *for* $\varphi_{x:w_y}$*.*

Example 3.10. *The formula* ψ_0 *in Example 2.10 has a non-evasiveness pattern, for instance the one with root labeled* x *and with one child labeled* z*. Generalizing from this example to the "path-shaped" monotone acyclic graph DNFs of the form* $\psi_n = (x_0 \wedge x_1) \vee (x_1 \wedge x_2) \vee \cdots \vee (x_{n-1} \wedge x_n)$*, there is a non-evasiveness pattern iff* n *is divisible by* 3*.* ψ_0 *is the path expression with* $n = 3$*, and is indeed non-evasive.*

The existence of a non-evasiveness pattern for φ can be checked in PTIME recursively: we successively root φ at each variable x and compute bottom-up, for each variable y, if there is a non-evasiveness pattern rooted at y. Note also that the choice of root of a non-evasiveness pattern Π among the variables occurring in Π is inessential: for any node y labeling a node of Π, we can re-root T and Π at y and obtain a non-evasiveness pattern rooted at y.

We will show that a non-evasiveness pattern for φ exists iff φ is non-evasive. We first show the easy direction:

Lemma 3.11. *Given a monotone acyclic graph DNF* φ*, if* φ *has a non-evasiveness pattern* Π*, then it is not evasive.*

Proof. We show the claim by induction on the number of nodes of Π. If Π is just a singleton node, then φ is a formula where x does not appear in any term, so does not need to be queried. Otherwise, let x be the variable labeling the root node n of Π, let $y_1, \ldots, y_m$ be the children of x in T (with $m \geq 1$), and let $w_1, \ldots, w_m$ be the variables labeling the corresponding child nodes of n in Π; let $z_1, \ldots, z_m$ be their respective parents (each z_i is a child of y_i).

Let us build a BDD witnessing that φ is not evasive by asking about all the variables $y_1, \ldots, y_m$. If they all evaluate to False, then all terms involving x are falsified, so we need not query x and the remaining depth is less than the number of remaining variables. Hence, to conclude, it suffices to argue that the formula is not evasive when some y_i evaluates to true, say $\varphi_{y_{i_0}=\text{True}}$.

Now, in $\varphi_{y_{i_0}=\text{True}}$, the term $y_i \wedge z_i$ simplifies to the singleton term z_i, which subsumes the term $z_i \wedge w_i$. Thus, after performing partial evaluation, the resulting formula has a connected component that is precisely $\varphi_{z_i:w_i}$. Now, the subtree of Π rooted at n_i is by definition a non-evasiveness pattern for $\varphi_{z_i:w_i}$ rooted at w_i. Hence, by induction hypothesis, we know that $\varphi_{z_i:w_i}$ is not evasive, so $\varphi_{y_{i_0}=\text{True}}$ has a connected component that is not evasive and is itself non-evasive, concluding the proof. □

We also claim that the converse holds, via:

Lemma 3.12. *If a monotone acyclic graph DNF* φ *has no non-evasiveness pattern, then for any variable* x *of* φ*, there exists* $b \in \{\mathit{True}, \mathit{False}\}$ *such that, after performing partial evaluation on* $\varphi_{x=b}$*, no connected component has a non-evasiveness pattern.*

This implies that if φ does not have a non-evasiveness pattern, then it is evasive. Indeed, applying repeatedly the above lemma justifies that any BDD has a branch leading always to a formula without a non-evasiveness pattern, until we reach a formula with no variables (which is constant and evasive). Let us prove the lemma:

Proof. Root φ at the variable x, yielding a tree T. We say that a variable y of T is *special* if there is a non-evasiveness pattern for $\varphi_{x:y}$ rooted at y. Because we assumed φ has no subsumed terms, all leaves of T are special, and we know by definition of a non-evasiveness pattern that for every special variable x and child y of x there is a grandchild of y that is also special. We make the following observation:

Observation 3.13. *If x has a special child and a special grandchild, then the whole graph DNF φ has a non-evasiveness pattern in which x does not appear.*

Proof. Let y be the special child and z the special grandchild of x, and let Π_y and Π_z be respectively the non-evasiveness pattern of $\varphi_{x:y}$ rooted at y and the non-evasiveness pattern of $\varphi_{x:z}$ rooted at z. We proceed by induction on the height of T. The base case is when T has height at most 3: then the special nodes z and y cannot have any child and must be leaves, and we have a suitable non-evasiveness pattern for φ with one variable, say y, as the root, and the other as its only child.

For the induction case, there are two subcases. First, if z is not a child of y in T, then we conclude that φ has a non-evasiveness pattern rooted at y. We build it from the non-evasiveness pattern Π_y rooted at y for $\varphi_{x:y}$, but we take into account the additional child x of y when rooting φ at y by adding Π_z as a child node of the root in Π_y. Note that x does not occur as a node label in the result.

Second, if z is a child of y, then y is not a leaf. As y is special and z is a neighbor of y, the definition of Π_y ensures that there must be a grandchild y' of z that is special. Letting w be the intermediate node between z and y', it witnesses that z is not a leaf, so by definition of Π_z there must be a grandchild z' of w that is also special. We can thus repeat the argument on the subtree of T rooted at w, which is of strictly smaller height. By induction, this subtree has a non-evasiveness pattern Π_w that does not involve w. Now, Π_w is a non-evasiveness pattern for the whole φ, because we can connect the rest of the tree T to w without breaking the existing non-evasiveness pattern (as it does not involve w). □

We now prove the lemma by contradiction. Assume there is a variable x such that for each $b \in \{\text{False}, \text{True}\}$ some connected component of $\varphi_{x=b}$ has a non-evasiveness pattern, and let us show this implies φ has a non-evasiveness pattern. Let T be the tree obtained by rooting φ at x. We wish to show that x has a special child and a special grandchild, to conclude using the previous observation.

Let us first take $b = \text{False}$. Let us consider the monotone acyclic graph DNF $\varphi_{x=\text{False}}$, perform partial evaluation to remove x and all terms involving x, and root the child subtrees $T_1, \ldots, T_n$ of T at the variables $y_1, \ldots, y_n$ that co-occur with x in φ: these are the connected components to consider. We know by hypothesis that some connected component T_i of $\varphi_{x=\text{False}}$ has a non-evasiveness pattern Π_i. If Π_i does not use y_i as a label, then we can use it as non-evasiveness pattern for all of φ, because adding back the rest of the tree T does not break the definition of a non-evasiveness pattern (it does not add any new neighbor to consider), and this immediately concludes. Hence, it suffices to consider the case where Π_i uses y_i as a label, and we can re-root it to a non-evasiveness pattern of T_i rooted at y_i. Hence, y_i is special in T.

Now consider the monotone acyclic graph DNF $\varphi_{x=\text{True}}$. Performing partial evaluation, the neighbors $y_1, \ldots, y_n$ become singleton terms, and the other terms involving them are subsumed. The singleton connected components corresponding to the $y_1, \ldots, y_n$ cannot have non-evasiveness patterns because they appear in singleton terms. The other connected components are the subtrees $T_1, \ldots, T_m$ rooted at the grandchildren $z_1, \ldots, z_m$ of the root of T. By hypothesis, one of them, say T_j, has a non-evasiveness pattern, and as in the previous paragraph has a non-evasiveness pattern rooted at the grandchild z_j, so that z_j is special in T. We can now use the existence of the special child y_i of T and the special grandchild z_j of T to establish using Observation 3.13 that T has a non-evasiveness pattern. This concludes the proof. □

To conclude, we have shown in Lemma 3.11 that indeed, if a non-evasiveness pattern occurs in a monotone acyclic graph DNF φ, we have a strategy that always probes less variables than the number of variables of φ, i.e., φ is non-evasive. Lemma 3.12 proves, for the other direction, that if φ has no non-evasiveness pattern, then it is indeed evasive. This completes the proof of Theorem 3.8.

4 Related Work

Interactive Boolean Evaluation and Consent Management The problem of Interactive Boolean Evaluation [2, 8, 15] and BDD optimization has been extensively studied in multiple contexts. These include system testing, e.g., [8, 44] (where it is called *Sequential System Testing* or *Sequential Diagnosis*), BDD design, e.g., [13, 18, 31] (where it is also called *Discrete Function Evaluation*), active learning [24] (where it is a particular case of *Bayesian Active Learning*) and its connection to other problems such as Stochastic Set Cover [15, 29] (where it is termed *Stochastic Boolean Function Evaluation (SBFE)*). Boolean Evaluation is also a component of *Consent Management* [6, 17], where the choice of variables to observe coincides with requests to attain consent to use specific input tuples. Many of these studies focus on optimization goals different than ours, most notably the expected number of observed variables under a probabilistic model [2, 6, 8, 13, 15, 17, 29, 31, 44], corresponding to the *expected* path length in a BDD; but other alternatives such as minimizing the size of the BDD have been considered [19].

Worst-case/depth analysis of BDDs and related problems have been studied in additional contexts, e.g., testing edges to decide about graph properties (e.g., [12, 28, 40, 46]) or letters to decide string properties [12]. The Boolean functions considered are thus very different from ours. There is also work on computing BDDs with minimum depth for input-output sample pairs (e.g., [13, 18]), but where a Boolean formula is not a-priori known.

Probabilistic databases A large body of work studied query evaluation in probabilistic databases [14, 32, 36, 43, 45], which resembles our work in the use of data provenance [25, 26] to track the connection between input and query output and the approach of identifying classes of queries that yield provenance expressions of favorable structure. In particular, prior work on probabilistic databases has characterized query classes where provenance may be transformed into read-once form for its useful properties such as easy probability computation [20, 27, 39, 41]. In our context, we have studied the evasiveness of such expressions (Section 3.2).

Data Provenance Provenance for query results has been extensively studied, with multiple models and applications [4, 5, 3, 10, 11, 16, 23, 22, 25, 26, 30, 35, 42]. Specifically, as discussed above the shape of the Boolean provenance may depend on the used data model (e.g., relational, graph databases, nested data) and query language (positive relational algebra, difference, Datalog). This, in turn, may affect the problem analysis, as we have shown for monotone Boolean provenance yielded by queries without negation.

Crowdsourced databases In practical scenarios, the worst-case number of observed variables may be translated to the worst-case number of questions posed to a human expert or crowd members, which should be accounted for, e.g., in optimization or by proper budget assignment. Works on crowd data sourcing such as [7, 21, 33, 34, 37, 38] considered different problems that typically also involve some optimization of the questions posed to crowd members. Specifically, the work of [7] considered the cleaning of an input database in order to fix query results, which involves the propagation of Boolean correctness from the input to output tuples; but the problem they solve is completely different and hence so are their analysis and results.

5 Conclusion and Future Work

We have studied in this paper the problem of *evaluating* provenance expressions, namely repeatedly probing individual variables to reveal their truth values in order to reveal the truth value of entire expressions. We have focused here on a worst-case analysis of the problem based on BDD representation of the provenance. We have shown that computing the number of required probes is generally intractable, but identified several classes of provenance expressions for which the problem becomes tractable. In some of these classes, we can identify that the expressions are evasive and one cannot do better than simply probing all variables, and in others, we can in fact do exponentially better than this naive approach.

The motivation for our study originates in our work on consent management [17] where we have used interactive Boolean evaluation of provenance expressions. Yet our techniques and analysis may have applications in other domains as well, such as data cleaning and view maintenance. In future work we will further explore such application domains, and will continue characterizing the query classes and corresponding classes of Boolean expressions that admit efficient interactive evaluation algorithms.

Acknowledgements We thank the anonymous reviewers for their insightful comments. This work was partly funded by the Israel Science Foundation (grant

No. 2015/21), by a grant from the Israel Ministry of Science and Technology, and by the grants ANR-18-CE23-0003-02 ("CQFD") and ANR-19-CE48-0019 ("EQUUS").

References

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

[2] S. R. Allen, L. Hellerstein, D. Kletenik, and T. Ünlüyurt. Evaluation of monotone DNF formulas. *Algorithmica*, 77(3), 2017.

[3] Y. Amsterdamer, S. B. Davidson, D. Deutch, T. Milo, J. Stoyanovich, and V. Tannen. Putting lipstick on Pig: Enabling database-style workflow provenance. *PVLDB*, 5(4), 2011.

[4] Y. Amsterdamer, D. Deutch, and V. Tannen. On the limitations of provenance for queries with difference. In *TaPP'11*, 2011.

[5] Y. Amsterdamer, D. Deutch, and V. Tannen. Provenance for aggregate queries. In *PODS*, 2011.

[6] Y. Amsterdamer and O. Drien. PePPer: Fine-Grained Personal Access Control via Peer Probing. In *ICDE*, 2019.

[7] M. Bergman, T. Milo, S. Novgorodov, and W. C. Tan. Query-oriented data cleaning with oracles. In *SIGMOD*, 2015.

[8] E. Boros and T. Ünlüyurt. Sequential testing of series-parallel systems of small depth. In *Computing tools for modeling, optimization and simulation.* Springer, 2000.

[9] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *Computers, IEEE Transactions on*, 100(8):677–691, 1986.

[10] P. Buneman, S. Khanna, and W. C. Tan. Why and where: A characterization of data provenance. In *ICDT*, 2001.

[11] J. Cheney, L. Chiticariu, and W. C. Tan. Provenance in databases: Why, how, and where. *Foundations and Trends in Databases*, 1(4), 2009.

[12] C. E. Chronaki. A survey of evasiveness: Lower bounds on the decision-tree complexity of boolean functions. Technical report, Citeseer, 1990.

[13] F. Cicalese, E. S. Laber, and A. M. Saettler. Diagnosis determination: decision trees optimizing simultaneously worst and expected testing cost. In *ICML*, 2014.

[14] N. N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. *VLDB J.*, 16(4), 2007.

[15] A. Deshpande, L. Hellerstein, and D. Kletenik. Approximation algorithms for stochastic Boolean function evaluation and stochastic submodular set cover. In *SODA*, 2014.

[16] D. Deutch, T. Milo, S. Roy, and V. Tannen. Circuits for datalog provenance. In *ICDT*, 2014.

[17] O. Drien, A. Amarilli, and Y. Amsterdamer. Managing Consent for Data Access in Shared Databases. In *ICDE*, 2021.

[18] A. Fiat and D. Pechyony. Decision trees: More theoretical justification for practical algorithms. In *ALT*, 2004.

[19] A. Fiat and D. Pechyony. Decision trees: More theoretical justification for practical algorithms. In *Algorithmic Learning Theory*, 2004.

[20] R. Fink and D. Olteanu. On the optimal approximation of queries using tractable propositional languages. In *ICDT*, 2011.

[21] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. CrowdDB: answering queries with crowdsourcing. In *SIGMOD*, 2011.

[22] B. Glavic and G. Alonso. Perm: Processing provenance and data on the same data model through query rewriting. In *ICDE*, 2009.

[23] B. Glavic and G. Alonso. Provenance for nested subqueries. In *EDBT*, 2009.

[24] D. Golovin and A. Krause. Adaptive submodularity: Theory and applications in active learning and stochastic optimization. *JAIR*, 42, 2011.

[25] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, 2007.

[26] T. Imielinski and W. L. Jr. Incomplete information in relational databases. *JACM*, 31(4), 1984.

[27] A. K. Jha and D. Suciu. Knowledge compilation meets database theory: Compiling queries to decision diagrams. *Theory Comput. Syst.*, 52(3), 2013.

[28] J. Kahn, M. E. Saks, and D. Sturtevant. A topological approach to evasiveness. In *FOCS*, 1983.

[29] H. Kaplan, E. Kushilevitz, and Y. Mansour. Learning with attribute costs. In *STOC*, 2005.

[30] G. Karvounarakis, Z. G. Ives, and V. Tannen. Querying data provenance. In *SIGMOD*, 2010.

[31] O. Keren. Reduction of average path length in binary decision diagrams by spectral methods. *IEEE TOCS*, 57, 2008.

[32] D. Koller. Probabilistic relational models. In *ILP*, 1999.

[33] G. Li, C. Chai, J. Fan, X. Weng, J. Li, Y. Zheng, Y. Li, X. Yu, X. Zhang, and H. Yuan. CDB: A crowd-powered database system. *PVLDB*, 11(12), 2018.

[34] A. Marcus, E. Wu, D. Karger, S. Madden, and R. Miller. Crowdsourced databases: query processing with people. In *CIDR*, 2011.

[35] D. Olteanu and J. Zavodny. Factorised representations of query results: size bounds and readability. In *ICDT*, 2012.

[36] K. Papaioannou, M. Theobald, and M. H. Böhlen. Supporting set operations in temporal-probabilistic databases. In *ICDE*, 2018.

[37] A. G. Parameswaran, H. Park, H. Garcia-Molina, N. Polyzotis, and J. Widom. Deco: declarative crowdsourcing. In *CIKM*, 2012.

[38] A. G. Parameswaran, A. D. Sarma, H. Garcia-Molina, N. Polyzotis, and J. Widom. Human-assisted graph search: it's okay to ask questions. *PVLDB*, 4(5), 2011.

[39] S. Roy, V. Perduca, and V. Tannen. Faster query answering in probabilistic databases using read-once functions. In *ICDT*, 2011.

[40] R. Scheidweiler and E. Triesch. A lower bound for the complexity of monotone graph properties. *SIAM J. Discrete Math.*, 27(1), 2013.

[41] P. Sen, A. Deshpande, and L. Getoor. Read-once functions and query evaluation in probabilistic databases. *PVLDB*, 3(1), 2010.

[42] P. Senellart, L. Jachiet, S. Maniu, and Y. Ramusat. ProvSQL: Provenance and probability management in PostgreSQL. *PVLDB*, 11(12), 2018.

[43] D. Suciu, D. Olteanu, C. Ré, and C. Koch. *Probabilistic Databases.* Morgan & Claypool, 2011.

[44] T. Ünlüyurt. Sequential testing of complex systems: a review. *Discrete Applied Mathematics*, 142(1-3), 2004.

[45] G. Van den Broeck and D. Suciu. Query processing on probabilistic data: A survey. *Found. Trends DBs*, 7(3-4), 2017.

[46] A. C.-C. Yao. Monotone bipartite graph properties are evasive. *SIAM Journal on Computing*, 17(3):517–520, 1988.