Counting Problems over Incomplete Databases

Marcelo Arenas
PUC & IMFD Chile
marenas@ing.puc.cl

Pablo Barceló
PUC & IMFD Chile
pbarcelo@ing.puc.cl

Mikaël Monet
IMFD Chile
mikael.monet@imfd.cl

ABSTRACT

We study the complexity of various fundamental counting problems that arise in the context of incomplete databases, i.e., relational databases that can contain unknown values in the form of labeled nulls. Specifically, we assume that the domains of these unknown values are finite and, for a Boolean query q, we consider the following two problems: given as input an incomplete database D, (a) return the number of completions of D that satisfy q; or (b) return or the number of valuations of the nulls of D yielding a completion that satisfies q. We obtain dichotomies between #P-hardness and polynomial-time computability for these problems when q is a self-join–free conjunctive query, and study the impact on the complexity of the following two restrictions: (1) every null occurs at most once in D (what is called *Codd tables*); and (2) the domain of each null is the same. Roughly speaking, we show that counting completions is much harder than counting valuations (for instance, while the latter is always in #P, we prove that the former is not in #P under some widely believed theoretical complexity assumption). Moreover, we find that both (1) and (2) reduce the complexity of our problems. We also study the approximability of these problems and show that, while counting valuations always has a fully polynomial randomized approximation scheme, in most cases counting completions does not. Finally, we consider more expressive query languages and situate our problems with respect to known complexity classes.

CCS CONCEPTS

• **Theory of computation → Complexity theory and logic**; **Incomplete, inconsistent, and uncertain databases**; • **Mathematics of computing →** *Approximation algorithms.*

KEYWORDS

Incomplete databases, closed-world assumption, counting complexity, FPRAS

ACM Reference Format:
Marcelo Arenas, Pablo Barceló, and Mikaël Monet. 2020. Counting Problems over Incomplete Databases. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS'20), June 14–19, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3375395.3387656

PODS'20, June 14–19, 2020, Portland, OR, USA

ACM ISBN 978-1-4503-7108-7/20/06…$15.00
https://doi.org/10.1145/3375395.3387656

1 INTRODUCTION

Context. In the database literature, *incomplete databases* are often used to represent missing information in the data; see, e.g., [1, 30, 44]. These are traditional relational databases whose active domain can contain both constants and *nulls*, the latter representing unknown values [25]. There are many ways in which one can define the semantics of such a database, each being equally meaningful depending on the intended application. Under the so called *closed-world assumption* [1, 37], a standard, complete database $\nu(D)$ is obtained from an incomplete database D by applying a *valuation* ν which replaces each null $\perp$ in D with a constant $\nu(\perp)$. The goal is then to reason about the space formed by all valuations ν and completions $\nu(D)$ of D.

Decision problems related to querying incomplete databases have been well studied already. Consider for instance the problem $\mathsf{Certainty}(q(\bar{x}))$ which, for a fixed query $q(\bar{x})$, takes as input an incomplete database D and a tuple $\bar{a}$ and asks whether $\bar{a}$ is an answer to q for every possible completion/valuation of D. By now, we have a deep understanding of the complexity of these kind of decision problems for different choices of query languages, including conjunctive queries (CQs) and FO queries [2, 25]. However, having the answer to this question is sometimes of little help: what if it is not the case that q is certain on D? Can we still infer some useful information? This calls for new notions that could be used to measure the certainty with which q holds, notions which should be finer than those previously considered. This is for instance what the recent work in [31] does by introducing a notion of *best answer*, which are those tuples $\bar{a}$ for which the set of completions of D over which $q(\bar{a})$ holds is maximal with respect to set inclusion.

A fundamental complementary approach to address this issue can be obtained by considering some *counting problems* related to incomplete databases; more specifically, determining the number of completions/valuations of an incomplete database that satisfy a given Boolean query q. These problems are relevant as they tell us, intuitively, how close is q from being certain over D, i.e., what is the level of *support* that q has over the set of completions/valuations of D. Surprisingly, such counting problems do not seem to have been studied for incomplete databases. A reason for this omission in the literature might be that, in general, it is assumed that the domain over which nulls can be interpreted is infinite, and thus incomplete databases might have an infinite number of completions/valuations. However, in many scenarios it is natural to assume that the domain over which nulls are interpreted is finite, in particular when dealing with uncertainty in practice [4, 6, 7, 11, 21, 38]. By assuming this we can ensure that the number of completions and valuations are always finite, and thus that they can be counted. This is the setting that we study.

In this paper, we will focus only on Boolean queries. We consider this as a necessary first step in understanding the complexity of

some relevant counting problems over incomplete databases. Besides, as shown in the paper, the case of Boolean queries is rich and complex enough to deserve its own investigation, providing very valuable information for the general case where queries can have arbitrary tuples as answers.

Problems studied. We focus on the problems #Comp(q) and #Val(q) for a Boolean query q, which take as input an incomplete database D together with a finite set dom($\bot$) of constants for every null $\bot$ occurring in D, and ask the following: How many completions, resp., valuations, of D satisfy q? More formally, a *valuation* ν of D is a mapping that associates to every null $\bot$ of D a constant $\nu(\bot)$ in dom($\bot$). Then, given a valuation ν of D, we denote by $\nu(D)$ the database that is obtained from D after replacing each null $\bot$ with $\nu(\bot)$. Besides, in this paper we consider set semantics, so repeated tuples have to be removed from $\nu(D)$. For #Comp(q) we count all databases of the form $\nu(D)$ such that q holds in $\nu(D)$. Instead, for #Val(q) we count the number of valuations ν such that q holds in $\nu(D)$. It is easy to see that these two values can differ, as there might be different valuations ν, ν' of D leading to the same completion, i.e., $\nu(D) = \nu'(D)$. We think that both problems are meaningful: while #Comp(q) determines the support for q over the databases represented by D, we have that #Val(q) further refines this by incorporating the support for a particular completion that satisfies q over the set of valuations for D.

The problems we study are analogous to the ones studied in other uncertainty scenarios; in particular, in *probabilistic databases* [19, 39] and *inconsistent databases* [9, 12, 13]. More specifically, we deal with the problems #Comp(q) and #Val(q) focusing on obtaining dichotomy results for them in terms of counting complexity classes, as well as studying the existence of randomized algorithms that approximate their results under probabilistic guarantees. For the dichotomies, we concentrate on self-join-free Boolean conjunctive queries (sjfBCQs). This assumption simplifies the mathematical analysis, while at the same defines a setting which is rich enough for many of the theoretical concepts behind these problems to appear in full force. Notice that a similar assumption is used in several works that study counting problems over probabilistic and inconsistent databases; see, e.g., [18, 32].

To refine our analysis, we study two restrictions of the problems #Comp(q) and #Val(q) based on natural modifications of the semantics, and analyze to what extent these restrictions simplify our problems. For the first restriction we consider incomplete databases in which each null occurs exactly once, which corresponds to the well-studied setting of *Codd tables* – as opposed to *naïve tables* where nulls are allowed to have multiple occurrences. We denote the corresponding problems by #Val$_{Cd}$(q) and #Comp$_{Cd}$(q). For the second restriction, we consider *uniform* incomplete databases in which all the nulls share the same domain – as opposed to the basic *non-uniform* setting in which all nulls come equipped with their own domain. We denote the corresponding problems by #Val^u(q) and #Comp^u(q). When both restrictions are in place, we denote the problems by #Val$^u_{Cd}$(q) and #Comp$^u_{Cd}$(q).

Our dichotomies for exact counting. We provide almost complete characterizations of the complexity of counting valuations and completions satisfying a given sjfBCQ q, when the input is a Codd table or a naïve table, and is a non-uniform or a uniform incomplete database (hence we have eight cases in total). The only case we have not yet completely solved is that of counting valuations in the uniform setting over Codd tables, i.e., the problem #Val$^u_{Cd}$(q). Our seven dichotomies express that these problems are either tractable or #P-hard, and that the tractable cases can be fully characterized by the absence of certain forbidden *patterns* in q. A pattern is simply an sjfBCQ which can be obtained from q essentially by deleting atoms, renaming relation symbols, deleting occurrences of variables and reordering the variables in atoms (the exact definition of this notion is given in Section 3). Our characterizations are presented in Table 1. By analyzing this table we can draw some important conclusions as explained next.

#Comp(q) and #Val(q) are computationally difficult: For very few sjfBCQs q the aforementioned problems can be solved in polynomial time. Take as an example the uniform setting over naïve tables. Then #Val^u(q) is #P-hard as long as q contains the pattern $R(x,x)$, or $R(x) \wedge S(x,y) \wedge T(y)$, or $R(x,y) \wedge S(x,y)$. That is, as long as there is an atom in q that contains a repeated variable x, or a pair (x,y) of variables that appear in an atom and both x and y appear in some other atoms in q. By contrast, for this same setting, #Comp^u(q) is #P-hard as long as q contains the pattern $R(x,y)$ or $R(x,x)$, that is, as long as there is an atom in q that is not of arity one.

#Val(q) is always easier than #Comp(q): In all of the possible versions of our problem, the tractable cases for #Val(q) are a strict superset of the ones for #Comp(q). For instance, we have that #Comp$^u_{Cd}$($\exists x \exists y\, R(x,y)$) is hard, while #Val$^u_{Cd}$($\exists x \exists y\, R(x,y)$) is tractable (because #Val^u($\exists x \exists y\, R(x,y)$) is).

Even counting completions is hard: While counting the total number of valuations for an incomplete database can always be done in polynomial time, observe from Table 1 that #Comp$^u_{Cd}$($\exists x \exists y\, R(x,y)$) is #P-hard, and thus that simply counting the completions of a uniform Codd table with a single binary relation R is #P-hard. Moreover, we show that in the non-uniform case a single unary relation suffices to obtain #P-hardness.

Codd tables help but not much: We show that counting valuations is easier for Codd tables than for naïve tables. In particular, there is always an sjfBCQ q such that counting the valuations that satisfy q is #P-hard, yet it becomes tractable when restricted to the case of Codd tables. However, for counting completions, both in the uniform and non-uniform setting, the sole restriction to Codd tables presents no benefits: for every sjfBCQ q, we have that #Comp(q) (resp., #Comp^u(q)) is #P-hard if and only if #Comp$_{Cd}$(q) (resp., #Comp$^u_{Cd}$(q)) is #P-hard.

Non-uniformity complicates things: All versions of our problems become harder in the non-uniform setting. This means that in all cases there is an sjfBCQ q for which countings valuations is tractable on uniform incomplete databases, but becomes #P-hard assuming non-uniformity, and an analogous result holds for counting completions.

Our dichotomies for approximate counting. Although #Val(q) can be #P-hard, we prove in the paper that good randomized approximate algorithms can be designed for this problem. More precisely, we give a general condition under which #Val(q) admits a *fully polynomial-time randomized approximation scheme* [27]

	Counting valuations		Counting completions	
	Non-uniform	**Uniform**	**Non-uniform**	**Uniform**
Naïve	$R(x,x)$ $R(x) \wedge S(x)$	$R(x,x)$ $R(x) \wedge S(x,y) \wedge T(y)$ $R(x,y) \wedge S(x,y)$	$R(x)$	$R(x,x)$ $R(x,y)$
Codd	$R(x) \wedge S(x)$	$R(x) \wedge S(x,y) \wedge T(y)$?	$R(x)$	$R(x,x)$ $R(x,y)$

Table 1: Our dichotomies for counting valuations and completions of sjfBCQs**. For each one of the eight cases, if an** sjfBCQ q **has as a pattern a query mentioned in that case, then the problem is** #P**-hard (and** #P**-complete for counting valuations, as well as for counting completions over Codd tables). In turn, for each case except for counting valuations under uniform Codd tables, if an** sjfBCQ q **does not have as a pattern any of the queries mentioned in that case, then the problem is in** FP**.**

(FPRAS). This condition applies in particular to all *unions of Boolean conjunctive queries*. Remarkably, we show that this no longer holds for #Comp(q); more precisely, there exists an sjfBCQ q such that #Comp(q) does not admit an FPRAS under a widely believed complexity theoretical assumption. More surprisingly, even counting the completions of uniform incomplete database containing a single binary relation does not admit an FPRAS under such an assumption (and in the non-uniform case, a single unary relation suffices). Generally, for sjfBCQs, we obtain seven dichotomies for our problems between polynomial-time computability of exact counting and non admissibility of an FPRAS. The only case that we did not solve yet is that of $\#\mathrm{Comp}^u_{Cd}(q)$.

Beyond #P**.** It is easy to see that the problem of counting valuations is always in #P. This is no longer the case for counting completions, and in fact we show that, under a complexity theoretical assumption, there is an sjfBCQ q for which $\#\mathrm{Comp}^u(q)$ is not in #P. This does not hold if restricted to Codd tables, however, as we prove that $\#\mathrm{Comp}_{Cd}(q)$ is always in #P.

For reasons that we explain in the paper, a suitable complexity class for the problem #Comp(q) is SpanP, which is defined as the class of counting problems that can be expressed as the number of different accepting outputs of a nondeterministic Turing machine running in polynomial time. While we have not managed to prove that there is an sjfBCQ q for which #Comp(q) is SpanP-complete, we show in the paper that this is the case for the problem of counting completions for the negation of an sjfBCQ, even in the uniform setting; that is, we show that $\#\mathrm{Comp}^u(\neg q)$ is SpanP-complete for some sjfBCQ q.

Organization of the paper. We start with the main terminology used in the paper in Section 2, and then present in Section 3 our three dichotomies on #Val(q) when q is an sjfBCQ, and the input incomplete database can be Codd or not, and the domain can be uniform or not. We then establish the four dichotomies on #Comp(q) in Section 4. In Section 5, we study the approximability complexity of our problems. We then give in Section 6 some general considerations about the exact complexity of the problem #Comp(q) going beyond #P. In Section 7, we discuss related work and explain the differences with the problems considered in this paper. Last, we provide some conclusions and mention possible directions for future work in Section 8. Due to the lack of space, only a few proofs are provided in the body of the paper. All missing proofs can be found in the full version of the article [8].

2 PRELIMINARIES

Relational databases and conjunctive queries. A *relational schema* σ is a finite non-empty set of relation symbols written R, S, T, …, each with its associated *arity*, which is denoted by $\mathrm{arity}(R)$. Let Consts be a countably infinite set of constants. A *database* D over σ is a set of *facts* of the form $R(a_1, \ldots, a_{\mathrm{arity}(R)})$ with $R \in \sigma$, and where each element $a_i \in \mathrm{Consts}$. For $R \in \sigma$, we denote by $D(R)$ the subset of D consisting of facts over R. Such a set is usually called a *relation* of D.

A Boolean query q is a query that a database D can *satisfy* (written $D \models q$) or not (written $D \not\models q$). A *Boolean conjunctive query* (BCQ) over σ is an FO formula of the form

$$\exists \bar{x} \left(R_1(\bar{x}_1) \wedge \ \ldots \ \wedge R_m(\bar{x}_m)\right), \tag{1}$$

where all variables are existentially quantified, and where for each $i \in [1, m]$, we have that R_i is a relation symbol in σ and $\bar{x}_i$ is a tuple of variables with $|\bar{x}_i| = \mathrm{arity}(R_i)$. To avoid trivialities, we will always assume that $m \geqslant 1$, i.e., the query has at least one atom, and also that $\mathrm{arity}(R_i) \geqslant 1$ for all atoms. For simplicity, we typically write a BCQ q of the form (1) as

$$R_1(\bar{x}_1) \wedge \ \ldots \ \wedge R_m(\bar{x}_m),$$

and it will be implicitly understood that all variables in q are existentially quantified. As usual, we define the semantics of a BCQ in terms of *homomorphisms*. A homomorphism from q to a database D is a mapping from the variables in q to the constants used in D such that $\{R_1(h(\bar{x}_1)), \ldots, R_m(h(\bar{x}_m))\} \subseteq D$. Then, we have $D \models q$ if there is a homomorphism from q to D. A *self-join–free BCQ* (sjfBCQ) is a BCQ such that no two atoms use the same relation symbol.

Incomplete databases. Let Nulls be a countably infinite set of nulls (also called *labeled nulls* in the literature), which is disjoint with Consts. An *incomplete database* over schema σ is a pair $D = (T, \mathrm{dom})$, where T is a database over σ whose facts contain elements in $\mathrm{Consts} \cup \mathrm{Nulls}$, and where dom is a function that associates to every null $\perp$ occurring in D a subset $\mathrm{dom}(\perp)$ of Consts.

$(\nu(\bot_1), \nu(\bot_2))$	(a,a)	(a,b)	(b,a)	(b,b)	(c,a)	(c,b)
$\nu(D)$	S	S	S	S	S	S
	$a\ \ b$	$a\ \ b$	$a\ \ b$	$a\ \ b$	$a\ \ b$	$a\ \ b$
	$a\ \ a$	$a\ \ a$	$b\ \ a$	$b\ \ a$	$c\ \ a$	$c\ \ a$
			$a\ \ a$		$a\ \ a$	
$\nu(D) \models q$?	Yes	Yes	Yes	No	Yes	No

Figure 1: The six valuations of the (non-uniform) incomplete database $D = (T, \mathsf{dom})$ with $T = \{S(a,b), S(\bot_1, a), S(a, \bot_2)\}$ from Example 2.2, and their corresponding completions. The Boolean conjunctive query q is $\exists x\, S(x,x)$.

Intuitively, T is a database that can mention both constants and nulls, while dom tells us where nulls are to be interpreted. Following the literature, we call T a *naïve table* [25].

An incomplete database $D = (T, \mathsf{dom})$ can represent potentially many complete databases, via what are called *valuations*. A valuation of D is simply a function ν that maps each null $\bot$ occurring in T to a constant $\nu(\bot) \in \mathsf{dom}(\bot)$. Such a valuation naturally defines a *completion of* D, denoted by $\nu(T)$, which is the complete database obtained from T by substituting each null $\bot$ appearing in T by $\nu(\bot)$. It is understood, since a database is a *set* of facts, that $\nu(T)$ does not contain duplicate facts. By paying attention to completions of incomplete databases that are generated exclusively by applying valuations to them, we are sticking to the so called *closed-world* semantics of incompleteness [1, 37]. This means that the databases represented by an incomplete database $D = (T, \mathsf{dom})$ are not open to adding facts that are not "justified" by the facts in T.

Example 2.1. Let $D = (T, \mathsf{dom})$ be the incomplete database consisting of the naïve table $T = \{S(\bot_1, \bot_1), S(a, \bot_2)\}$, and where $\mathsf{dom}(\bot_1) = \{a, b\}$ and $\mathsf{dom}(\bot_2) = \{a, c\}$. Let ν_1 be the valuation mapping $\bot_1$ to b and $\bot_2$ to c. Then $\nu_1(T)$ is $\{S(b,b), S(a,c)\}$. Let ν_2 be the valuation mapping both $\bot_1$ and $\bot_2$ to a. Then $\nu_2(T)$ is $\{S(a,a)\}$. On the other hand, the function ν mapping $\bot_1$ and $\bot_2$ to b is not a valuation of D, because $b \notin \mathsf{dom}(\bot_2)$. □

When every null occurs at most once in T, then D is what is called a *Codd table* [16]; for instance, the incomplete database in Example 2.1 is not a Codd table because $\bot_1$ occurs twice. We also consider *uniform* incomplete databases in which the domain of every null is the same. Formally, a uniform incomplete database is a pair $D = (T, \mathsf{dom})$, where T is a database over σ and dom is a subset of Consts. The difference now is that a valuation ν of D must simply satisfy $\nu(\bot) \in \mathsf{dom}$ for every null of D.

We will often abuse notation and use D instead of T; for instance, we write $\nu(D)$ instead of $\nu(T)$, or $R(a,a) \in D$ instead of $R(a,a) \in T$, or again $D(R)$ instead of $T(R)$.

Counting problems on incomplete databases. We will study two kinds of counting problems for incomplete databases: problems of the form $\#\mathsf{Val}(q)$, that count the number of *valuations* ν that yield a completion $\nu(D)$ satisfying a given BCQ q, and problems of the form $\#\mathsf{Comp}(q)$, that count the number of *completions* that satisfy q. The query q is assumed to be fixed, so that each query gives rise to different counting problems, and we are considering the *data complexity* [45] of these problems.

Before formally introducing our problems, let us observe that they are well defined if we assume that the set of constants to which a null can be mapped to is finite. Hence, for the (default) case of an incomplete database $D = (T, \mathsf{dom})$, we assume that $\mathsf{dom}(\bot)$ is always a finite subset of Consts. Similarly, for the case of a uniform incomplete database $D = (T, \mathsf{dom})$, we assume that dom is a finite subset of Consts. Finally, given a Boolean query q, we use notation $\mathsf{sig}(q)$ for the set of relation symbols occurring in q. With these ingredients, we can define our problems for the (default) case of incomplete naïve tables and a Boolean query q.

PROBLEM :	$\#\mathsf{Val}(q)$
INPUT :	An incomplete database D over $\mathsf{sig}(q)$
OUTPUT :	Number of valuations ν of D with $\nu(D) \models q$

PROBLEM :	$\#\mathsf{Comp}(q)$
INPUT :	An incomplete database D over $\mathsf{sig}(q)$
OUTPUT :	Number of completions $\nu(D)$ of D with $\nu(D) \models q$

We also consider the uniform variants of these problems, in which the input D is a uniform incomplete database over $\mathsf{sig}(q)$, and the restriction of these problems where the input is a Codd table instead of a naïve table. We then use the terms $\#\mathsf{Val}^u(q)$, $\#\mathsf{Comp}^u(q)$ when restricted to the uniform case, $\#\mathsf{Val}_{Cd}(q)$, $\#\mathsf{Comp}_{Cd}(q)$ when restricted to Codd tables, and $\#\mathsf{Val}^u_{Cd}(q)$, $\#\mathsf{Comp}^u_{Cd}(q)$ when both restrictions are applied.

As we will see, even though the problems $\#\mathsf{Val}(q)$ and $\#\mathsf{Comp}(q)$ look similar, they are of a different computational nature; this is because two distinct valuations can produce the same completion of an incomplete database. In the following example, we illustrate this phenomenon.

Example 2.2. Let q be the Boolean conjunctive query $\exists x\, S(x,x)$, and D be the (non-uniform) incomplete database $D = (T, \mathsf{dom})$, with $T = \{S(a,b),\ S(\bot_1, a), S(a, \bot_2)\}$, $\mathsf{dom}(\bot_1) = \{a, b, c\}$ and $\mathsf{dom}(\bot_2) = \{a, b\}$. We have depicted in Figure 1 the six valuations of D together with the completions that they define. Out of these six valuations ν, only four are such that $\nu(D) \models q$, so that we have $\#\mathsf{Val}(q)(D) = 4$. Moreover, there are only 3 distinct completions of D that satisfy q, so $\#\mathsf{Comp}(q)(D) = 3$. □

Counting complexity classes. Given two problems A, B, we write $A \leqslant^{\mathrm{p}}_{\mathrm{T}} B$ when A reduces to B under polynomial-time Turing reductions. When both A and B are counting problems, we

write $A \leqslant^{\mathrm{p}}_{\mathrm{par}} B$ when A can be reduced to B under polynomial-time *parsimonious* reductions, i.e., there exists a polynomial-time computable function f that transforms an input x of A to an input $f(x)$ of B such that $A(x) = B(f(x))$. We say that a counting problem is in FP when it can be solved in polynomial time. We will consider the counting complexity class #P [42] of problems that can be expressed as the number of accepting paths of a nondeterministic Turing machine running in polynomial time. Following [42, 43], we define #P-hardness using Turing reductions. It is clear that FP $\subseteq$ #P. This inclusion, on the other hand, is widely believed to be strict. Therefore, proving that a counting problem is #P-hard implies that it cannot be solved in polynomial time under such an assumption.

Graphs. In our reductions, we will often depart from hard problems that are defined over graphs. Unless mentioned otherwise, by *graph* we mean a pair $G = (V, E)$, where V is a finite set of *nodes*, and E is a set whose elements are of the form $\{u, v\}$ for $u, v \in V$ and $u \neq v$. Notice then that such graphs are *undirected*, cannot contain *self-loops*, and cannot contain multiple edges between any two nodes.

3 DICHOTOMIES FOR COUNTING VALUATIONS

In this section, given a fixed sjfBCQ q, we study the complexity of the problem of computing, given an incomplete database D, the number of valuations ν of D such that $\nu(D)$ satisfies q. Recall that we have four cases to consider for this problem depending on whether we focus on naïve or on Codd tables, where nulls are restricted to appear at most once, and whether we focus on non-uniform or uniform incomplete databases, where nulls are restricted to have the same domain. Our specific goal then is to understand whether the problem is tractable (in FP) or #P-hard in these scenarios, depending on the shape of q.

To this end, the shape of an sjfBCQ q will be characterized by the presence or absence of certain specific patterns. In the following definition, we introduce the necessary terminology to formally talk about the presence of a pattern in a query.

Definition 3.1. Let q, q' be sjfBCQs. We say that q' is a *pattern* of q if q' can be obtained from q by using an arbitrary number of times and in any order the following operations: deleting an atom, deleting an occurrence of a variable, renaming a relation to a fresh one, renaming a variable to a fresh one and reordering the variables in an atom.[1]

Example 3.2. Recall that we always omit existential quantifiers in Boolean queries. Then we have that $q' = R'(u, u, y) \wedge S'(z)$ is a pattern of $q = R(u, x, u) \wedge S'(y, y) \wedge T(x, s, z, s)$. Indeed, q' can be obtained from q by deleting atom $T(x, s, z, s)$, renaming $R(u, x, u)$ as $R'(u, x, u)$ to obtain $R'(u, x, u) \wedge S'(y, y)$, reordering the variables in $R'(u, x, u)$ to obtain $R'(u, u, x) \wedge S'(y, y)$, renaming variable y into z to obtain $R'(u, u, x) \wedge S'(z, z)$, deleting the second variable occurrence in $S'(z, z)$ to obtain $R'(u, u, x) \wedge S'(z)$, and finally renaming variable x into y to obtain q'. □

In the following general lemma, we show that if q' is a pattern of q, then each one of the problems considered in this section is as hard for q as it is for q'. Recall in this result that unless stated otherwise, our problems are defined for naïve tables.

Lemma 3.3. *Let q, q' be* sjfBCQs *such that q' is a pattern of q. Then we have* $\#\mathrm{Val}(q') \leqslant^{\mathrm{p}}_{\mathrm{par}} \#\mathrm{Val}(q)$ *and* $\#\mathrm{Val}^u(q') \leqslant^{\mathrm{p}}_{\mathrm{par}} \#\mathrm{Val}^u(q)$. *Moreover, the same results hold if we restrict to Codd tables.*

The idea is then to show the #P-hardness of our problems for some simple patterns, which then we combine with Lemma 3.3 and with some tractability proofs to obtain the desired dichotomies. Our findings are summarized in the first two columns of Table 1 in the introduction. We first focus on the two dichotomies for the non-uniform setting in Section 3.1, and then we move to the case of uniform incomplete databases in Section 3.2. We explicitly state when a #P-hardness result holds even in the restricted setting in which there is a fixed domain over which nulls are interpreted. In other words, when there is a fixed domain A such that the incomplete databases used in the reductions are of the form $D = (T, \mathrm{dom})$ and $\mathrm{dom}(\bot) \subseteq A$, for each null $\bot$ of T.

3.1 The complexity on the non-uniform case

In this section, we study the complexity of the problems $\#\mathrm{Val}(q)$ and $\#\mathrm{Val}_{Cd}(q)$, providing dichotomy results in both cases. We start by proving the #P-hardness results needed for these dichotomies. We first show that $\#\mathrm{Val}(R(x, x))$ is #P-hard by actually proving that hardness holds already in the uniform case.

Proposition 3.4. *$\#\mathrm{Val}^u(R(x, x))$ is* #P*-hard and, hence, $\#\mathrm{Val}(R(x, x))$ is also* #P*-hard. This holds even in the restricted setting in which all nulls are interpreted over the same fixed domain* $\{1, 2, 3\}$.

Proof. We reduce from the problem of counting the number of 3-colorings of a graph $G = (V, E)$, which is #P-hard [26]. For every node $v \in V$ we have a null $\bot_v$, and for every edge $\{u, v\} \in E$ we have the facts $R(\bot_v, \bot_u)$ and $R(\bot_u, \bot_v)$. The domain of the nulls is $\{1, 2, 3\}$. It is then clear that the number of valuations of the constructed database that do not satisfy $R(x, x)$ is exactly the number of 3-colorings of G. Since the total number of valuations can be computed in PTIME, this concludes the reduction. □

The next pattern that we consider is $R(x) \wedge S(x)$. This time, we can show #P-hardness of the problem even for Codd databases.

Proposition 3.5. *$\#\mathrm{Val}_{Cd}(R(x) \wedge S(x))$ is* #P*-hard.*

Already with Propositions 3.5 and 3.4, we have all the relevant hard patterns for the non-uniform setting. We start by proving our dichotomy result for naïve tables, which is our default case.

Theorem 3.6 (dichotomy). *Let q be an* sjfBCQ*. If $R(x, x)$ or $R(x) \wedge S(x)$ is a pattern of q, then $\#\mathrm{Val}(q)$ is* #P*-complete. Otherwise, $\#\mathrm{Val}(q)$ is in* FP.

Proof. The #P-hardness part of the claim follows from the last two propositions and from Lemma 3.3. We explain why the problems are in #P right after this proof. When q does not have any of these two patterns then all variables have exactly one occurrence in q. This implies that every valuation ν of D is such that $\nu(D)$ satisfies q (except when one relation is empty, in which case the result is simply zero). We can obviously compute the total number of valuations in FP by simply multiplying the sizes of the domains of every null in D. □

[1] We remind the reader that we assume all sjfBCQs to contain at least one atom and that all atoms must contain at least one variable.

Notice that in this theorem, the membership of $\#\mathrm{Val}(q)$ in #P can be established by considering a nondeterministic Turing Machine M that, with input a non-uniform incomplete database D, guesses a valuation ν of D and verifies whether $\nu(D)$ satisfies q. This machine works in polynomial time as we can verify whether $\nu(D)$ satisfies q in polynomial time (since q is a fixed FO query). Then given that $\#\mathrm{Val}(q)(D)$ is equal to the number of accepting runs of M with input D, we conclude that $\#\mathrm{Val}(q)$ is in #P. Obviously, the same idea shows that $\#\mathrm{Val}_{Cd}(q)$ is in #P. But with this restriction we obtain more tratable cases, as shown by the following dichotomy result.

Theorem 3.7 (dichotomy). *Let q be an* sjfBCQ. *If $R(x) \wedge S(x)$ is a pattern of q, then $\#\mathrm{Val}_{Cd}(q)$ is* #P*-complete. Otherwise, $\#\mathrm{Val}_{Cd}(q)$ is in* FP.

Proof. We only need to prove the tractability claim, since hardness follows from Proposition 3.5 and Lemma 3.3. We will assume without loss of generality that D contains no constants, as we can introduce a fresh null with domain $\{c\}$ for every constant c appearing in D, and the result is again a Codd table, and this does not change the output of the problem. Let q be $R_1(\bar{x}_1) \wedge \ldots \wedge R_m(\bar{x}_m)$. Observe that since q does not have $R(x) \wedge S(x)$ as a pattern then any two atoms cannot have a variable in common. But then, since D is a Codd table we have

$$\#\mathrm{Val}_{Cd}(q)(D) \;=\; \prod_{i=1}^{m} \#\mathrm{Val}_{Cd}(R_i(\bar{x}_i))(D(R_i)).$$

Hence it is enough to show how to compute $\#\mathrm{Val}_{Cd}(R_i(\bar{x}_i))(D(R_i))$ for every $1 \leqslant i \leqslant m$. Let $\bar{t}_1, \ldots, \bar{t}_n$ be the tuples of $D(R_i)$. Let us write $\rho(\bar{t}_j)$ for the number of valuations of the nulls appearing in $\bar{t}_j$ that do not match $\bar{x}_i$. Clearly, $\#\mathrm{Val}_{Cd}(R_i(\bar{x}_i))(D(R_i)) = \prod_{\perp \text{ appears in } D(R_i)} |\mathrm{dom}(\perp)| - \prod_{j=1}^{n} \rho(\bar{t}_j)$, so we only have to show how to compute $\rho(\bar{t}_j)$ for $1 \leqslant j \leqslant n$. Since we can easily compute the total number of valuations of $\bar{t}_j$, it is enough to show how to compute the number of valuations of $\bar{t}_j$ that match $\bar{x}_i$. For every variable x that appears in $\bar{x}_i$, compute the size of the intersection of the corresponding nulls in $\bar{t}_j$, and denote it s_x. Then the number of valuations of $\bar{t}_j$ that match $\bar{x}_i$ is simply $\prod_{x \text{ appears in } \bar{x}_i} s_x$. This concludes the proof. □

At this stage, we have completed the first column of Table 1, and we also know that $R(x, x)$ is a hard pattern in the uniform setting for naïve tables (but not for Codd tables, by Theorem 3.7). In the next section, we treat the uniform setting.

3.2 The complexity on the uniform case

We start our investigation with the case of naïve tables. In Proposition 3.4, we already showed that $\#\mathrm{Val}^u(R(x, x))$ is #P-hard. In the following proposition, we identify two other simple queries for which this problem is still intractable.

Proposition 3.8. *$\#\mathrm{Val}^u(R(x) \wedge S(x, y) \wedge T(y))$ and $\#\mathrm{Val}^u(R(x, y) \wedge S(x, y))$ are both* #P*-hard. This holds even in the restricted setting in which all nulls are interpreted over the same fixed domain $\{0, 1\}$.*

Proof. We reduce both problems from the problem of counting the number of independent sets in a graph (denoted by #IS), which is #P-complete [36]. We start with $\#\mathrm{Val}^u(R(x) \wedge S(x, y) \wedge T(y))$. Let $q = R(x) \wedge S(x, y) \wedge T(y)$ and $G = (V, E)$ be a graph. Then we define an incomplete database D as follows. For every node $v \in V$, we have a null $\perp_v$, and the uniform domain is $\{0, 1\}$. For every edge $\{u, v\} \in E$, we have facts $S(\perp_u, \perp_v)$ and $S(\perp_v, \perp_u)$ in D. Finally, we have facts $R(1)$ and $T(1)$ in D. For a valuation ν of the nulls, consider the corresponding subset S_ν of nodes of G, given by $S_\nu = \{t \in V \mid \nu(\perp_t) = 1\}$. This is a bijection between the valuations of the database and the node subsets of G. Moreover, we have that $\nu(D) \not\models q$ if and only if S_ν is an independent set of G. Since the total number of valuations of D is $2^{|V|}$, we have that the number of independent sets of G is equal to $2^{|V|} - \#\mathrm{Val}^u(q)(D)$. Hence, we conclude that $\#\mathrm{IS} \leqslant_{\mathrm{T}}^{\mathrm{p}} \#\mathrm{Val}^u(q)$. The idea is similar for $\#\mathrm{Val}^u(R(x, y) \wedge S(x, y))$: we encode the graph with the relation S in the same way, and this time we add the fact $R(1, 1)$. □

As shown in the following result, it turns out that the three aforementioned patterns are enough to fully characterize the complexity of counting valuations for naïve tables in the uniform setting.

Theorem 3.9 (dichotomy). *Let q be an* sjfBCQ. *If $R(x, x)$ or $R(x) \wedge S(x, y) \wedge T(y)$ or $R(x, y) \wedge S(x, y)$ is a pattern of q, then $\#\mathrm{Val}^u(q)$ is* #P*-complete. Otherwise, $\#\mathrm{Val}^u(q)$ is in* FP.

The #P-completeness part of the claim follows directly from what we have proved already. Here, the most challenging part of the proof is actually the tractability part. We only present a simple example to give an idea of the proof technique. We will use the following definition. Given $n, m \in \mathbb{N}$, let us write $\mathrm{surj}_{n \to m}$ for the number of surjective functions from $\{1, \ldots, n\}$ to $\{1, \ldots, m\}$. By an inclusion–exclusion argument, one can show that $\mathrm{surj}_{n \to m} = \sum_{i=0}^{m-1} (-1)^i \binom{m}{i} (m - i)^n$ (for instance, see [3]). It is clear that this can be computed in FP, when n and m are given in unary.

Example 3.10. Let q be the sjfBCQ $R(x) \wedge S(x)$, and D be an incomplete database over relations R, S. Notice that q does not have any of the patterns mentioned in Theorem 3.9. We will show that $\#\mathrm{Val}^u(q)$ is in FP. Since q contains only two unary atoms we can also assume without loss of generality that the input D is a Codd table (otherwise all valuations are satisfying).

Since we can compute in FP the total number of valuations, it is enough to show how to compute the number of valuations of D that do not satisfy q. Let dom be the uniform domain, d be its size, n_R (resp., n_S) be the number of nulls in $D(R)$ (resp., in $D(S)$) and C_R (resp., C_S) be the set of constants occurring in $D(R)$ (resp., in $D(S)$), with c_R (resp., c_S) its size. We can assume without loss of generality that $C_R \cap C_S = \emptyset$, as otherwise all the valuations are satisfying, and this is computable in PTIME. Furthermore, we can also assume that $C_R \cup C_S \subseteq \mathrm{dom}$, since we can remove the constants that are not in dom, as these can never match.

Let $M := \mathrm{dom} \setminus (C_R \cup C_S)$, and m its size (i.e., with our assumptions we have $m = d - c_R - c_S$). Fix some subsets $M' \subseteq M$ and $R' \subseteq C_R$. The quantity $\mathrm{surj}_{n_R \to |M'| + |R'|}$ then counts the number of valuations of the nulls of $D(R)$ that span exactly $M' \cup R'$. Moreover, letting ν_R be a valuation of the nulls of $D(R)$ that spans exactly $M' \cup R'$, the quantity $(d - c_R - |M'|)^{n_S}$ is the number of ways to extend ν_R into a valuation ν of all the nulls of D so that $\nu(D) \not\models q$: indeed, every null of $D(S)$ can take any value in $\mathrm{dom} \setminus (C_R \cup M')$. The number of valuations of D that do not satisfy q is then (keeping

in mind that a null in $D(R)$ cannot take a value in C_S):

$$\sum_{\substack{M' \subseteq M \\ R' \subseteq C_R}} \mathsf{surj}_{n_R \to |M'|+|R'|} \times (d - c_R - |M'|)^{n_S}$$

and since the summands only depends on the sizes of M' and R', this is equal to

$$\sum_{\substack{0 \leqslant m' \leqslant m \\ 0 \leqslant r' \leqslant c_R}} \binom{m}{m'} \binom{c_R}{r'} \mathsf{surj}_{n_R \to m'+r'} \times (d - c_R - m')^{n_S}$$

This last expression can clearly be computed in PTIME.[2] □

We conclude this section by turning our attention to the case of Codd tables. Notice that none of the results proved so far provides a hard pattern in this case. We identify in the following proposition a simple query for which the problem is intractable.

Proposition 3.11. $\#\mathsf{Val}^u_{Cd}(R(x) \wedge S(x,y) \wedge T(y))$ *is* #P*-hard.*

In Proposition 3.8, we proved that $\#\mathsf{Val}^u(R(x) \wedge S(x,y) \wedge T(y))$ is #P-hard in the general case where naïve tables are allowed. Hence, Proposition 3.8 was in fact a consequence of Proposition 3.11, where only Codd tables are allowed. However, we decided to provide a separate proof for Proposition 3.8, because this result includes another intractable case, and both cases in Proposition 3.8 can be established via a simple reduction from counting independent sets. By contrast, Proposition 3.11 requires of a more complicated proof (we reduce from #IS on bipartite graphs and use a Turing reduction with $(n/2+1)^2$ calls to the oracle, where n is the number of nodes of the input graph, to form a system of linear equations which we then invert to recover the number of independent sets).

Up to this point, we have not been able to prove that when an sjfBCQ q does not contain $R(x) \wedge S(x,y) \wedge T(y)$ as a pattern, it holds that $\#\mathsf{Val}^u_{Cd}(q)$ is in FP. Thus, the possibility of having a dichotomy in this case is left as a problem for future research. Nevertheless, we can still observe that restricting to Codd tables simplifies the problem of counting valuations in the non-uniform setting. Indeed, considering the query $R(x,x)$, counting valuations is #P-hard for naïve tables, while it is in FP for Codd tables by Theorem 3.7.

4 DICHOTOMIES FOR COUNTING COMPLETIONS

In this section, we study the complexity of the problems of counting completions satisfying an sjfBCQ q, in the four cases that can be obtained by considering naïve or Codd tables and non-uniform or uniform domains. We will again use the notion of pattern as introduced in Definition 3.1. Our first step is to show that Lemma 3.3, which we used in the last section for the problems or counting valuations, extends to the problems of counting completions.

Lemma 4.1. *Let q, q' be* sjfBCQs *such that q' is a pattern of q. Then we have that* $\#\mathsf{Comp}(q') \leqslant^{\mathrm{p}}_{\mathrm{par}} \#\mathsf{Comp}(q)$ *and* $\#\mathsf{Comp}^u(q') \leqslant^{\mathrm{p}}_{\mathrm{par}} \#\mathsf{Comp}^u(q)$*. Moreover, the same results hold if we restrict to the case of Codd tables.*

[2]Note that in the sum we do not need to specify that $m' + r' \leqslant n_R$, as when $a < b$ we have $\mathsf{surj}_{a \to b} = 0$.

We will then follow the same general strategy as in the last section, i.e., prove hardness for some simple patterns and combine these with Lemma 4.1 and tractability proofs to obtain dichotomies. Our findings are summarized in the last two columns of Table 1 in the introduction. We start in Section 4.1 with the non-uniform cases and continue in Section 4.2 with the uniform cases. Again, we explicitly state when a #P-hardness result holds even in the restricted setting in which there is a fixed domain over which nulls are interpreted.

4.1 The complexity on the non-uniform case

Here, we study the complexity of the problems $\#\mathsf{Comp}(q)$ and $\#\mathsf{Comp}_{Cd}(q)$, providing dichotomy results in both cases. In fact, it turns out that these problems are #P-hard for all sjfBCQs. To prove this, it is enough to show that the problem $\#\mathsf{Comp}_{Cd}(R(x))$ is hard, that is, even counting the completions of a single unary table is #P-hard in the non-uniform setting.

Proposition 4.2. $\#\mathsf{Comp}_{Cd}(R(x))$ *is* #P*-hard.*

Proof. We provide a polynomial-time parsimonious reduction from the problem of counting the vertex covers of a graph, which we denote by #VC. Let $G = (V, E)$ be a graph. We construct a Codd table D using a single unary relation R such that the number of completions of D equals the number of vertex covers of G. For every edge $e = \{u, v\}$ of G, we have one null $\bot_e$ with $\mathsf{dom}(\bot_e) = \{u, v\}$ and the fact $R(\bot_e)$. Let a be a fresh constant. For every node $u \in V$ we have a null $\bot_u$ with $\mathsf{dom}(\bot_u) = \{u, a\}$ and the fact $R(\bot_u)$. Last, we add the fact $R(a)$. We now show that the number of completions of D equals the number of vertex covers of G.

Let $\mathrm{VC}(G)$ be the set of vertex covers of G. For a valuation ν of D, define the set $S_\nu := \{u \in V \mid R(u) \in D\}$. Since the fact $R(a)$ is in every completion of D, it is clear that the number of completions of D is equal to $|\{S_\nu \mid \nu \text{ is a valuation of } D\}|$. We claim that $\mathrm{VC}(G) = \{S_\nu \mid \nu \text{ is a valuation of } D\}$, which shows that the reduction works. ($\subseteq$) Let $C \in \mathrm{VC}(G)$, and let us show that there exists a valuation ν of D such that $S_\nu = C$. For a null of the form $\bot_e$ with $e = \{u, v\} \in E$, assuming wlog that $u \in C$, we define $\nu(\bot_e)$ to be u. For a null of the form $\bot_u$ with $u \in V$, we define $\nu(\bot_u)$ to be u if $u \in C$ and a otherwise. It is then clear that $S_\nu = C$. ($\supseteq$) Let ν be a valuation of D, and let us show that S_ν is a vertex cover. Assume by contradiction that there is an edge $e = \{u, v\}$ such that $e \cap S_\nu = \emptyset$. By definition of D, we must have $\nu(\bot_e) \in \{u, v\}$, so that one of u or v must be in S_ν, hence a contradiction. Therefore, we conclude that $\#\mathrm{VC} \leqslant^{\mathrm{p}}_{\mathrm{par}} \#\mathsf{Comp}_{Cd}(R(x))$. □

Recall from Section 2 that, to avoid trivialities, we assume all sjfBCQs to contain at least one atom and that the arity of every atom is $\geqslant 1$ (that is, all atoms have at least one variable). Using Lemma 3.3, this allows us to obtain the following dichotomy result.

Theorem 4.3 (Dichotomy). *For every* sjfBCQ q*, it holds that* $\#\mathsf{Comp}(q)$ *and* $\#\mathsf{Comp}_{Cd}(q)$ *are* #P*-hard.*

Notice here that we do not claim membership in #P; in fact, we will come back to this issue in Section 6 to show that this is unlikely to be true for naïve tables. However, we can still show that membership in #P holds for Codd tables. We then obtain:

Theorem 4.4 (Dichotomy). *For every* sjfBCQ q, *the problem* $\#\mathrm{Comp}_{Cd}(q)$ *is* #P*-complete.*

4.2 The complexity on the uniform case

We now investigate the complexity of $\#\mathrm{Comp}^u(q)$ and $\#\mathrm{Comp}^u_{Cd}(q)$. Recall that in the non-uniform case, even counting the completions of a single unary table is a #P-hard problem. This no longer holds in the uniform case, as we will show that $\#\mathrm{Comp}^u(q)$ is in FP for every sjfBCQ that is defined over a schema consisting exclusively of unary relation symbols.

Such a positive result, however, cannot be extended much further. In fact, we show next that $R(x,x)$ and $R(x,y)$ are hard patterns (and, thus, we also conclude that the problem of counting the completions of a single binary table is a #P-hard problem). Moreover, #P-hardness holds even if restricted to one of the following settings: (a) Naïve tables where nulls are interpreted over a fixed domain, and (b) Codd tables.

Proposition 4.5. *We have that:*

(a) $\#\mathrm{Comp}^u(R(x,x))$ *and* $\#\mathrm{Comp}^u(R(x,y))$ *are both* #P*-hard, even when nulls are interpreted over the same fixed domain* $\{0,1\}$.
(b) $\#\mathrm{Comp}^u_{Cd}(R(x,x))$ *and* $\#\mathrm{Comp}^u_{Cd}(R(x,y))$ *are* #P*-hard.*

Proof. We only present the proof of (a) here. The proof of (b) requires more work and can be found in the extended version [8]. We reduce from #IS, the problem of counting the number of independent sets of a graph.

Let $G = (V,E)$ be a graph. We will construct an incomplete database D containing a single binary predicate R such that each completion of D satisfies $R(x,x)$ and the number of completions of D is $2^{|V|} + \#\mathrm{IS}(G)$, thus establishing hardness for the two queries. For every node $u \in V$, we have a null $\bot_u$ with $\mathrm{dom}(\bot_u) = \{0,1\}$. We then construct the naïve table D as follows:

- for every node $u \in V$ we add to D the fact $R(u, \bot_u)$;
- then for every edge $\{u,v\} \in E$, we add the facts $R(\bot_u, \bot_v)$ and $R(\bot_v, \bot_u)$ to D; and
- last, we add the facts $R(0,0)$, $R(0,1)$, $R(1,0)$, and $R(\bot,\bot)$, where $\bot$ is a fresh null.

It is clear that every completion of D satisfies $R(x,x)$.

Let us now count the number of completions of D. First, we observe that, thanks to the facts of the form $R(u,\bot_u)$, for $u \in V$, for every two valuations ν, ν' that do not assign the same value to the nulls of the form $\bot_u$, it is the case that $\nu(D) \neq \nu(D')$. We then partition the completions of D into those that contain the fact $R(1,1)$, and those that do not contain $R(1,1)$. Because of the facts of the form $R(u,\bot_u)$, for $u \in V$, and thanks to the fact $R(\bot,\bot)$ which becomes $R(1,1)$ when we assign 1 to $\bot$, there are exactly $2^{|V|}$ completions of D that contain $R(1,1)$. Moreover, it is easy to see that there are $\#\mathrm{IS}(G)$ valuations ν of D that assign 0 to $\bot$ and that yield a completion not containing $R(1,1)$. Indeed, one can check that a valuation of D that assigns 0 to $\bot$ yields a completion not containing $R(1,1)$ if and only if the set $\{u \in V \mid \nu(\bot_u) = 1\}$ is an independent set of G. Therefore, we conclude that $\#\mathrm{IS} \leqslant^{\mathrm{p}}_{\mathrm{T}} \#\mathrm{Comp}^u(q)$, where q can be $R(x,x)$ or $R(x,y)$. □

The patterns in Proposition 4.5 suffice to characterize the complexity of $\#\mathrm{Comp}^u(q)$ and $\#\mathrm{Comp}^u_{Cd}(q)$.

Theorem 4.6 (Dichotomy). *Let* q *be an* sjfBCQ. *If* $R(x,x)$ *or* $R(x,y)$ *is a pattern of* q, *then* $\#\mathrm{Comp}^u(q)$ *and* $\#\mathrm{Comp}^u_{Cd}(q)$ *are* #P*-hard. Otherwise, these problems are in* FP.

The proof of the tractability part of this theorem is combinatorial and very technical, and can be found in the extended version [8], where we also give several examples to provide the main intuitions. Note that, as in the last section, we do not claim membership in #P. However, and also as in the last section, we can show that these problems are in #P for Codd tables, which allows us to obtain our last dichotomy for exact counting.

Theorem 4.7 (Dichotomy). *Let* q *be an* sjfBCQ. *If* $R(x,x)$ *or* $R(x,y)$ *is a pattern of* q, *then* $\#\mathrm{Comp}^u_{Cd}(q)$ *is* #P*-complete. Otherwise, this problem is in* FP.

5 APPROXIMATING THE NUMBERS OF VALUATIONS AND COMPLETIONS

As we saw in the previous sections, counting valuations and completions of an incomplete database are usually intractable problems. However, this does not necessarily rule out the existence of efficient approximation algorithms for such counting problems, in particular if some source of randomization is allowed. In this section, we investigate this question by focusing on the well-known notion of Fully Polynomial-time Randomized Approximation Scheme (FPRAS) for counting problems [27]. Formally, let Σ be a finite alphabet and $f : \Sigma^* \to \mathbb{N}$ be a counting problem. Then f is said to have an FPRAS if there is a randomized algorithm $\mathcal{A} : \Sigma^* \times (0,1) \to \mathbb{N}$ and a polynomial $p(u,v)$ such that, given $x \in \Sigma^*$ and $\varepsilon \in (0,1)$, algorithm $\mathcal{A}$ runs in time $p(|x|, 1/\varepsilon)$ and satisfies the following condition:

$$\Pr\big(|f(x) - \mathcal{A}(x,\varepsilon)| \leqslant \varepsilon f(x)\big) \quad \geqslant \quad \frac{3}{4}.$$

Observe that the property of having an FPRAS is closed under polynomial-time parsimonious reductions, that is, if we have an FPRAS for a counting problem A and for counting problem B we have that $B \leqslant^{\mathrm{p}}_{\mathrm{par}} A$, then we also have an FPRAS for B.

In the following sections, we investigate the existence of FPRAS for the problems of counting valuations and completions of an incomplete database. We first deal with counting valuations in Section 5.1, where we show a general condition under which this problem has an FPRAS (which will apply, in particular, to all Boolean conjunctive queries). Then, in Section 5.2, we show that the situation is quite different for counting completions, as in most cases this problem does not admit an FPRAS.

5.1 Approximating the number of valuations

To prove the main result of this section, we need to consider the counting complexity class SpanL [5]. Given a finite alphabet Σ, an NL-transducer M over Σ is a nondeterministic Turing Machine with input and output alphabet Σ, a read-only input tape, a write-only output tape (where the head is always moved to the right once a symbol in Σ is written in it, so that the output cannot be read by M), and a work-tape of which, on input x, only the first $c \cdot \log(|x|)$ cells can be used for a fixed constant $c > 0$ (so that the space used by M is logarithmic). Moreover, $y \in \Sigma^*$ is said to be an output of M with input x, if there exists an accepting run of M with input x such that y is the string in the output tape when M halts. Then

a function $f : \Sigma^* \to \mathbb{N}$ is said to be in SpanL if there exists an NL-transducer M over Σ such that for every $x \in \Sigma^*$, the value $f(x)$ is equal to the number of distinct outputs of M with input x. In [5], it was proved that SpanL $\subseteq$ #P, and also that this inclusion is strict unless NL = NP.

Very recently, the authors of [10] have shown that every problem in SpanL has an FPRAS.

Theorem 5.1 ([10, Corollary 3]). *Every problem in* SpanL *has an FPRAS.*

By using this result, we can give a general condition on a Boolean query q under which #Val(q) has an FPRAS, as this condition ensures that #Val(q) is in SpanL. More precisely, a Boolean query q is said to be *monotone* if for every pair of (complete) databases D, D' such that $D \subseteq D'$, if $D \models q$, then $D' \models q$. Moreover, q is said to have *bounded minimal models* if there exists a constant C_q (that depends only on q) satisfying that for every (complete) database D, if $D \models q$, then there exists $D' \subseteq D$ such that $D' \models q$ and the number of facts in D' is at most C_q. Finally, the *model checking problem* for q, denoted by MC(q), is the problem of deciding, given a (complete) database D, whether $D \models q$. Then q is said to have a model checking in a complexity class $\mathcal{C}$ if MC(q) $\in \mathcal{C}$. With this terminology, we can state the main result of this section.

Proposition 5.2. *Assume that a Boolean query q is monotone, has model checking in nondeterministic linear space, and has bounded minimal models. Then* #Val(q) *is in* SpanL.

In particular, given that a union of Boolean of conjunctive queries satisfies the three properties of the previous proposition, we conclude from Theorem 5.1 that #Val(q) can be efficiently approximated by using a randomized algorithm if q is a union of BCQs.[3]

Corollary 5.3. *If q is a union of BCQs, then* #Val(q) *has an FPRAS. Hence, for such a query q, we have that each one of the problems* #Val$^u(q)$, #Val$_{Cd}(q)$, #Val$^u_{Cd}(q)$ *admits an FPRAS.*

We prove in the next section that the good properties stated in Proposition 5.2 do not hold for counting completions.

5.2 Approximating the number of completions

In this section, we prove that the problem of counting completions of an incomplete database is much harder in terms of approximation than the problem of counting valuations. In this investigation, two randomized complexity classes play a fundamental role. Recall that RP is the class of decision problems L for which there exists a polynomial-time probabilistic Turing Machine M such that: (a) if $x \in L$, then M accepts with probability at least 3/4; and (b) if $x \notin L$, then M does not accept x. Moreover, BPP is defined exactly as RP but with condition (b) replaced by: (b') if $x \notin L$, then M accepts with probability at most 1/4. Thus, BPP is defined as RP but allowing errors for both the elements that are and are not in L. It is easy to see that RP $\subseteq$ BPP. Besides, it is known that RP $\subseteq$ NP, and this inclusion is widely believed to be strict. Finally, it is not known whether BPP $\subseteq$ NP or NP $\subseteq$ BPP, but it is widely believed that NP is not included in BPP.

[3] As a matter of fact, this holds even for the larger class of unions of BCQs with *inequalities* (that is, atoms of the form $x \neq y$), as such queries also satisfy the aforementioned three properties.

The non-uniform case. Recall that #IS is the problem of counting the number of independent sets of a graph. This problem will play a fundamental role when showing non-approximability of counting completions in the non-uniform case. More precisely, the following is known about the approximability of #IS.

Theorem 5.4 ([20, Theorem 3.1]). *The problem* #IS *does not admit an FPRAS unless* NP = RP.

In the proof of Proposition 4.2, we considered the problem #VC of counting the number of vertex covers of a graph $G = (V, E)$, and showed that #VC $\leqslant^{\mathrm{p}}_{\mathrm{par}}$ #Comp$_{Cd}(R(x))$. By observing that $S \subseteq V$ is an independent set of G if and only if $V \setminus S$ is a vertex cover of G, we can conclude that #IS(G) = #VC(G) and, thus, the same reduction from the proof of Proposition 4.2 establishes that #IS $\leqslant^{\mathrm{p}}_{\mathrm{par}}$ #Comp$_{Cd}(R(x))$. Therefore, from the fact that the reduction in Lemma 4.1 is also parsimonious and preserves the property of being a Codd table, and the fact that the existence of an FPRAS is closed under polynomial-time parsimonious reductions, we obtain the following result from Theorem 5.4.

Theorem 5.5 (Dichotomy). *For every* sjfBCQ q*, it holds that* #Comp$_{Cd}(q)$ *does not admit an FPRAS unless* NP = RP (*and, hence, the same holds for* #Comp(q)).

The uniform case. Recall that from Theorem 4.6, we know that if an sjfBCQ q contains neither $R(x, x)$ nor $R(x, y)$ as a pattern, then #Comp$^u(q)$ is in FP. Thus, the question to answer in this section is whether #Comp$^u(q)$ and #Comp$^u_{Cd}(q)$ can be efficiently approximated if q contains any of these two patterns. For the case of naïve tables, we will give a negative answer to this question. Notice that, this time, our reduction from #IS in Proposition 4.5 is not parsimonious, so we cannot use Theorem 5.4 as we did for the non-uniform case. Instead, we will rely on the following well-known fact: if there exists a BPP algorithm for a problem that is NP-complete, then NP $\subseteq$ BPP, which implies that NP = RP [28].

Proposition 5.6. *Neither* #Comp$^u(R(x, x))$ *nor* #Comp$^u(R(x, y))$ *admits an FPRAS unless* NP = RP. *This holds even in the restricted setting in which all nulls are interpreted over the same fixed domain* $\{1, 2, 3\}$.

Proof. Let $G = (V, E)$ be a graph. First, we explain how to construct an incomplete database D containing a single binary relation R, with uniform domain $\{1, 2, 3\}$, and such that (a) all completions of D satisfy both queries; (b) if G is 3-colorable then D has 8 completions; and (c) if G is not 3-colorable then D has 7 completions. For every node $u \in V$ we have a null $\bot_u$. The database D consists of the following three disjoint sets of facts:

- For every edge $\{u, v\} \in E$, we have the two facts $R(\bot_u, \bot_v)$ and $R(\bot_v, \bot_u)$; we call these the *encoding facts*.
- We have the facts $R(1, 2), R(2, 1), R(2, 3), R(3, 2), R(1, 3)$, and $R(3, 1)$; we call these the *triangle facts*;
- We have six fresh nulls $\bot_1, \bot'_1, \bot_2, \bot'_2, \bot_3, \bot'_3$ and the facts $R(\bot_i, \bot'_i)$ and $R(\bot'_i, \bot_i)$ for $1 \leqslant i \leqslant 3$; we call these the *auxiliary facts*;
- Last, we have a fact $R(c, c)$, where c is a fresh constant.

It is clear that all the completions of D satisfy both queries, so we only need to prove (b) and (c). Observe that a candidate completion

of D can be equivalently seen as an undirected graph, *possibly with self-loops*, over the nodes $\{1, 2, 3\}$ (we omit the fact $R(c, c)$ since it is in every completion) and that contains the triangle. Thanks to the auxiliary facts, it is easy to show that all such graphs with at least one self-loop can be obtained as a completion of D. For instance, the completion that is triangle with a self-loop only on 1 can be obtained by assigning 1 to all the nulls in the coding facts, assigning 1 to $\perp_1$, $\perp'_1$, $\perp_2$ and $\perp_3$ and assigning 2 to $\perp'_2$ and $\perp'_3$. There are 7 such completions. Then, the completion whose graph is the triangle with no self-loops is obtainable if and only if G is 3-colorable (we assign a 3-coloring to the nulls in the coding facts, and assign 1 to $\perp_i$ and 2 to $\perp'_i$ for every $i \in \{1, 2, 3\}$). This indeed proves (b) and (c). Next, we show that any FRPAS with $\varepsilon = 1/16$ for counting the number of completions of D would yield a BPP algorithm to solve 3-colorability, thus implying NP = RP since 3-colorability is an NP-complete problem.

Let $\mathcal{A}$ be an FPRAS for $\#\mathrm{Comp}^u(q)$, with q being $R(x, x)$ or $R(x, y)$, and let us define a BPP algorithm $\mathcal{B}$ for 3-colorability using $\mathcal{A}$. On input graph G, algorithm $\mathcal{B}$ does the following. First, it computes in polynomial time the naïve table D as described above. Then $\mathcal{B}$ calls $\mathcal{A}$ with input $(D, 1/16)$, and if $\mathcal{A}(D, 1/16) \geqslant 7.5$, then $\mathcal{B}$ accepts, otherwise $\mathcal{B}$ rejects. We now prove that $\mathcal{B}$ is indeed a BPP algorithm for 3-colorability. Assume first that G is 3-colorable. Then by (b) and by definition of what is an FPRAS, we have that $\Pr\big(|8-\mathcal{A}(D, 1/16)| \leqslant 8/16\big) \geqslant \frac{3}{4}$. This implies in particular that $\Pr\big(\mathcal{A}(D, 1/16) \geqslant 8-8/16\big) \geqslant \frac{3}{4}$. Since $8 - 8/16 = 7.5$ we conclude that if G is 3-colorable, then $\mathcal{B}$ accepts with probability at least $3/4$. Next, assume that G is not 3-colorable. Then by (c) we have that $\Pr\big(|7 - \mathcal{A}(D, 1/16)| \leqslant 7/16\big) \geqslant \frac{3}{4}$. This implies in particular that $\Pr\big(\mathcal{A}(D, 1/16) \leqslant 7 + 7/16\big) \geqslant \frac{3}{4}$. Since $7 + 7/16 < 7.5$, this implies in particular that $\Pr\big(\mathcal{A}(D, 1/16) < 7.5\big) \geqslant \frac{3}{4}$. From this, we conclude that if G is not 3-colorable, then $\mathcal{B}$ rejects with probability at least $3/4$. This concludes the proof of the proposition. □

By observing again that the reduction in Lemma 4.1 is parsimonious, and that the existence of an FPRAS is closed under parsimonious reductions, we obtain that $\#\mathrm{Comp}^u(q)$ cannot be efficiently approximated if q contains $R(x, x)$ or $R(x, y)$ as a pattern.

Theorem 5.7 (Dichotomy). *Let q be an* sjfBCQ. *If q has $R(x, x)$ or $R(x, y)$ as a pattern, then $\#\mathrm{Comp}^u(q)$ does not admit an FPRAS unless* NP = RP. *Otherwise, this problem is in* FP (*by Theorem 4.6*).

Up until now, we do not know if this result still holds for Codd tables, or if it is possible to design an FPRAS in this setting. We leave this question open for future research.

6 ON THE GENERAL LANDSCAPE: BEYOND #P

Recall that, when studying the complexity of counting completions for sjfBCQs in Section 4, we did not show that these problems are in #P for naïve tables. The goal of this section is then twofold. First, we want to give formal evidence that we indeed could not show membership in #P in Section 4. Second, we want to identify a counting complexity class that is more appropriate to describe the complexity of $\#\mathrm{Comp}(q)$, but in a more general setting which is not based on some syntactic restrictions imposed on q. More precisely, for the second goal, we consider the complexity of the model checking problem $\mathrm{MC}(q)$ for q, which is defined in Section 5.1 as the problem of deciding, given a (complete) database D, whether $D \models q$. In this section, all upper bounds will be proved for the most general scenario of non-uniform naïve tables, while all lower bounds will be proved for the most restricted scenario of uniform naïve tables with the fixed domain $\{0, 1\}$.

To meet our first goal, we need to define the complexity class SPP introduced in [22, 24, 35]. Given a nondeterministic Turing Machine M and a string x, let $\mathrm{accept}_M(x)$ (resp., $\mathrm{reject}_M(x)$) be the number of accepting (resp., rejecting) runs of M with input x, and let $\mathrm{gap}_M(x) = \mathrm{accept}_M(x) - \mathrm{reject}_M(x)$. Then a language L is said to be in SPP [22] if there exists a polynomial-time nondeterministic Turing Machine M such that: (a) if $x \in L$, then $\mathrm{gap}_M(x) = 1$; and (b) if $x \notin L$, then $\mathrm{gap}_M(x) = 0$. In this way, SPP is the smallest class that can be defined in terms of the gap function gap_M. It is conjectured that NP $\not\subseteq$ SPP as, for example, for every known polynomial-time nondeterministic Turing Machine M accepting an NP-complete problem, the function gap_M is not bounded. In the following proposition, we show how this conjecture helps us to reach our first goal.

Proposition 6.1. *There exists an* sjfBCQ q *such that* $\#\mathrm{Comp}^u(q)$ *is not in* #P *unless* NP ⊆ SPP.

To meet our second goal, we need to introduce one last counting complexity class. The class SpanP [29] is defined exactly as the class SpanL introduced in Section 5.1, but considering *polynomial-time* nondeterministic Turing machines with output, instead of *logarithmic-space* nondeterministic Turing machines with output. It is straightforward to prove that #P ⊆ SpanP. Besides, it is known that #P = SpanP if and only if NP = UP [29].[4] Therefore, it is widely believed that #P is properly included in SpanP. The following easy observation can be seen as a first hint that SpanP is a good alternative to describe the complexity of counting completions.

Observation 6.2. *If q is a Boolean query such that* $\mathrm{MC}(q)$ *is in* P, *then* $\#\mathrm{Comp}(q)$ *is in* SpanP.

Notice that this result applies to all sjfBCQs and, more generally, to all FO Boolean queries. In fact, this results applies to even more expressive query languages such as Datalog [1]. More surprisingly, in the following theorem we show that $\#\mathrm{Comp}^u(q)$ can be SpanP-complete for an FO query q and, in fact, already for the negation of an sjfBCQ.

Theorem 6.3. *There exists an* sjfBCQ q *such that* $\#\mathrm{Comp}^u(\neg q)$ *is* SpanP*-complete under polynomial-time parsimonious reductions.*

This theorem gives evidence that SpanP is the right class to describe the complexity of counting completions for FO queries (and even for queries with model checking in polynomial time). It is important to notice that SpanP-hardness is proved in Theorem 6.3 by considering parsimonious reductions. This is a delicate issue because from the main result in [40], it is possible to conclude that every counting problem that is #P-hard (even under

[4] Recall that UP is the class Unambiguous Polynomial-Time introduced in [41], and that $L \in$ UP if and only if there exists a polynomial-time nondeterministic Turing Machine M such that if $x \in L$, then $\mathrm{accept}_M(x) = 1$, and if $x \notin L$, then $\mathrm{accept}_M(x) = 0$.

polynomial-time parsimonious reductions) is also SpanP-hard under polynomial-time Turing reductions, so a more restrictive notion of reduction has to be used when proving that a counting problem is SpanP-hard [29].

We conclude this section by considering an even more general scenario where queries have model checking in NP. Interestingly, in this case SpanP is again the right class to describe the complexity not only of counting completions, but also of counting valuations.

Theorem 6.4. *If q is a Boolean query with* $\mathrm{MC}(q) \in \mathrm{NP}$*, then both* $\#\mathrm{Val}(q)$ *and* $\#\mathrm{Comp}(q)$ *are in* SpanP*. Moreover, there exists such a Boolean query q for which* $\#\mathrm{Val}^u(q)$ *is* SpanP*-complete under polynomial-time parsimonious reductions (and for* $\#\mathrm{Comp}^u(q)$*, we can even take q to be the negation of an* sjfBCQ*, hence with model checking in* P*, as given by Theorem 6.3).*

7 RELATED WORK

There are two main lines of work that must be compared to what we do in this paper. In both cases the goal is to go beyond the traditional notion of *certain answers* that so far had been used almost exclusively to deal with query answering over uncertain data. We discuss them here, explain how they relate to our problems and what are the fundamental differences.

Best answers and 0-1 laws for incomplete databases. Libkin has recently introduced a framework that can be used to measure the certainty with which a Boolean query holds on an incomplete database, and also to compare query answers (for a non-Boolean query) [31]. For a Boolean query q, incomplete database D, and integer k, he defines the quantity $\mu^k(q, D)$ as $\frac{|\mathrm{Supp}^k(q,D)|}{|V^k(D)|}$, where $V^k(D)$ denotes the set of valuations of D with domain $\{1, \ldots, k\}$, and $\mathrm{Supp}^k(q, D)$ denotes the set of valuations $v \in V^k(D)$ such that $v(D) \models q$; hence, $\mu^k(q, D)$ represents the relative frequency of valuations v in $\{1, \ldots, k\}$ for which the query is satisfied. He then shows that, for a very large class of queries (namely, *generic queries*), the value $\mu^k(q, d)$ always tends to 0 or 1 as k tends to infinity (and the same results holds when considering completions instead of valuations). This means that, intuitively, over an infinite domain the query q is either almost certainly true or almost certainly false.

He also studies the complexity of finding best answers for a non Boolean query q. As mentioned in the introduction, a tuple $\overline{a}$ is a better answer than another tuple $\overline{b}$ when for every valuation v of D, if we have $\overline{b} \in q(v(d))$ then we also have $\overline{b} \in q(v(d))$. A best answer is then an answer such that there is no other answer strictly better than it (under inclusion of the sets of satisfying valuations). He studies the complexity of comparing answers under this semantics, and that of computing the set of best answers (see also [23]).

There are several crucial differences between this previous work and ours. First, Libkin does not study the complexity of computing $\mu^k(q, d)$. We do this under the name $\#\mathrm{Val}^u(q)$; moreover, we also study the setting in which the domains are not uniform. Second, knowing that a tuple is the best answer might not tell us anything about the size of its "support", i.e., the number of valuations that support it. In particular, a best answer is not necessarily an answer which has the biggest support. Finally, under the semantics of better answers it does not matter if we look at the completions or at the valuations (i.e., a tuple is a best answer with respect to inclusion of valuations iff it is the best answer with respect to completions); while we have shown that it does matter for counting problems.

Counting problems for probabilistic databases and consistent query answering. Remarkably, counting problems have received considerable attention in other database scenarios where uncertainty issues appear. As mentioned in the introduction, this includes the settings of probabilistic databases and inconsistent databases. In the former case, uncertainty is represented as a probability distribution on the possible states of the data [19, 39]. There, query answering amounts to computing a weighted sum of the probabilities of the possible states of the data that satisfy a query q. We call this problem $\mathrm{Prob}(q)$. In the case of inconsistent databases, we are given a set Σ of constraints and a database D that does not necessarily satisfy Σ; cf. [9, 12, 13]. Then the task is to reason about the set of all *repairs* of D with respect to Σ [9]. In our context, this means that one wants to count the number of repairs of D with respect to Σ that satisfy a given query q. When q and Σ are fixed, we call this problem $\#\mathrm{Repairs}(q, \Sigma)$.

Both $\mathrm{Prob}(q)$ and $\#\mathrm{Repairs}(q, \Sigma)$ have been intensively studied already. To start with, counting complexity dichotomies have been obtained for the problem $\#\mathrm{Repairs}(q, \Sigma)$; e.g., [32] gives a dichotomy for this problem when q is an sjfBCQ and σ consists of primary keys, and [33] extends this result to CQs with self-joins but only for unary keys constraints. We also mention [15], where the problem of counting repairs such that a particular input tuple is in the result of the query on the repair is studied. A seemingly close counting problem for probabilistic databases is the problem $\mathrm{Prob}(q)$ over *block independent disjoint* (BIDs) databases. We do not define it formally here, but counting repairs under primary keys can be seen as a special case of this problem, where the tuples in a "block" all have the same probability, and where the sum of the probabilities sum to 1 (and in BIDs this sum is allowed to be < 1, meaning that a block can be completely erased). Dichotomies for this problem have been obtained in [18] for sjfBCQs. Counting complexity dichotomies for other models of probabilistic databases also exist; e.g., for *tuple-independent* probabilistic databases in which each fact is assigned an independent probability of being part of the actual dataset. Interestingly, dichotomies in this case hold for arbitrary unions of BCQs, and thus not just for sjfBCQs [19].

In some cases, one can use a problem of the form $\#\mathrm{Repairs}(q, \Sigma)$ (or $\mathrm{Prob}(q)$) to show the hardness of a problem of the form $\#\mathrm{Val}(q')$. For instance, we explain in the extended version of the article [8] how the #P-hardness of $\#\mathrm{Val}_{Cd}(R(x) \wedge S(x))$ can be deduced from that of $\#\mathrm{Repairs}(R'(\underline{y}, x) \wedge S'(\underline{z}, x))$.[5] In general however, the problems $\#\mathrm{Repairs}(q, \Sigma)$ and $\mathrm{Prob}(q)$ seem to be unrelated to our problems, for the following reasons. First, in our setting the nulls can appear anywhere, so there is no notion of primary keys here; hence it seems unlikely that one can design a generic reduction from the problem of counting valuations/completions to the problem of counting repairs. In fact, it would perfectly make sense to study our counting problems where we add constraints such a functional dependencies. Second, in the BID and counting repairs problems, each "valuation" (repair) gives a different complete database, while

[5] Where we write, for instance, $R'(\underline{y}, x)$, to mean that the first attribute of R is a primary key.

in our case we have seen that this is not necessarily the case. In particular, problems of the form #Comp(q) have no analogues in these settings, whereas we have seen that they behave very differently in our setting.

Concerning approximation results, it is known that the problems #Repairs(q, Σ) and Prob(q) admit an FPRAS in some important settings. In particular, when q is a union of BCQs, this holds for #Repairs(q, Σ) when Σ is a set of primary keys [15], and for Prob(q) over BID and tuple-independent probabilistic databases [18]. We observe here that this is reminescent of our Corollary 5.3, which shows that problems of the form #Val(q) have an FPRAS for every union of BCQs.

8 FINAL REMARKS

Our work aims to be a first step in the study of counting problems over incomplete databases. The main conclusion behind our results is that the counting problems studied in this paper are particularly hard from a computational point of view, especially when compared to more positive results obtained in other uncertainty scenarios; e.g., over probabilistic and inconsistent databases. As we have shown, a particularly difficult problem in our context is that of counting completions, even in the uniform setting where all nulls have the same domain. In fact, Proposition 4.5 shows that this problem is #P-hard even in very restricted scenarios, and Proposition 5.6 that it cannot be approximated by an FPRAS. It seems then that the only way in which one could try to tackle this problem is by developing suitable tractable heuristics, without provable quantitative guarantees, but that work sufficiently well in practical scenarios. An example of this could be developing algorithms that compute "under-approximations" for the number of completions of a naïve table satisfying a certain sjfBCQ q. Notice that a related approach has been proposed by Console et al. for constructing under-approximations of the set of certain answers by applying methods based on many-valued logics [17].

We plan to continue working on several interesting problems that are left open in this paper. First of all, we would like to pinpoint the complexity of #Comp(q) when q is an sjfBCQ; in particular, whether this problem is SpanP-complete for at least one such a query. We also want to study whether the non-existence of FPRAS for #Comp^u(q) established in Proposition 5.6 continues to hold over Codd tables. We would also like to develop a more thorough understanding of the role of fixed domains in our dichotomies. In several cases, that we have explicitly stated, our lower bounds hold even if nulls in tables are interpreted over a fixed domain. Still, in some cases we do not know whether this holds. These include, e.g., Proposition 3.11, Proposition 4.2, and Proposition 4.5 for the case of Codd tables. Finally, it would also be interesting to study these counting problems under bag semantics (instead of the set semantics used in this paper), study counting problems for non-Boolean queries as in [23, 31], and consider arbitrary BCQs as opposed to only sjfBCQs.

ACKNOWLEDGMENTS

The third author would like to thank Antoine Amarilli for reminding him of the paper [14] in [34] and for suggesting to use #BIS in the proof of Proposition 3.11. This work was partly funded by the Millennium Institute for Foundational Research on Data (IMFD).

REFERENCES

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of databases*. Vol. 8. Addison-Wesley Reading.
[2] Serge Abiteboul, Paris C. Kanellakis, and Gösta Grahne. 1991. On the representation and querying of sets of possible worlds. *TCS* 78, 1 (1991), 158–187.
[3] Adam. 2013. Number of surjective functions. https://math.stackexchange.com/a/615136/378365
[4] Parag Agrawal, Anish Das Sarma, Jeffrey D. Ullman, and Jennifer Widom. 2010. Foundations of uncertain-data integration. *PVLDB* 3, 1 (2010), 1080–1090.
[5] Carme Álvarez and Birgit Jenner. 1993. A very hard log-space counting class. *TCS* 107, 1 (1993), 3–30.
[6] Periklis Andritsos, Ariel Fuxman, and Renee J Miller. 2006. Clean answers over dirty databases: A probabilistic approach. In *ICDE*. IEEE, 30–30.
[7] Lyublena Antova, Christoph Koch, and Dan Olteanu. 2009. $10^{(10^6)}$ worlds and beyond: efficient representation and processing of incomplete information. *VLDB J.* 18, 5 (2009), 1021–1040.
[8] Marcelo Arenas, Pablo Barceló, and Mikaël Monet. 2019. Counting Problems over Incomplete Databases. *arXiv preprint arXiv:1912.11064* (2019). https://arxiv.org/abs/1912.11064 Extended version of the current article.
[9] Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. 1999. Consistent query answers in inconsistent databases. In *PODS*. 68–79.
[10] Marcelo Arenas, Luis Alberto Croquevielle, Rajesh Jayaram, and Cristian Riveros. 2019. Efficient logspace classes for enumeration, counting, and uniform generation. In *PODS*. 59–73.
[11] Omar Benjelloun, Anish Das Sarma, Alon Y. Halevy, and Jennifer Widom. 2006. ULDBs: Databases with uncertainty and lineage. In *VLDB*. 953–964.
[12] Leopoldo E. Bertossi. 2011. *Database repairing and consistent query answering*. Morgan & Claypool Publishers.
[13] Leopoldo E. Bertossi. 2019. Database repairs and consistent query answering: Origins and further developments. In *PODS*. 48–58.
[14] Jin-Yi Cai, Pinyan Lu, and Mingji Xia. 2012. Holographic reduction, interpolation and hardness. *Computational Complexity* 21, 4 (2012), 573–604.
[15] Marco Calautti, Marco Console, and Andreas Pieris. 2019. Counting database repairs under primary Keys revisited. In *PODS*. 104–118.
[16] T Codd. 1975. Understanding relations (installment# 7). *FDT Bull. of ACM Sigmod* 7 (1975), 23–28.
[17] Marco Console, Paolo Guagliardo, and Leonid Libkin. 2016. Approximations and refinements of certain answers via many-valued logics. In *KR*. 349–358.
[18] Nilesh Dalvi, Christopher Re, and Dan Suciu. 2011. Queries and materialized views on probabilistic databases. *J. Comput. System Sci.* 77, 3 (2011), 473–490.
[19] Nilesh N. Dalvi and Dan Suciu. 2012. The dichotomy of probabilistic inference for unions of conjunctive queries. *J. ACM* 59, 6 (2012), 30.
[20] Martin Dyer, Alan Frieze, and Mark Jerrum. 2002. On counting independent sets in sparse graphs. *SIAM J. on Computing* 31, 5 (2002), 1527–1541.
[21] Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. 2008. Conditional functional dependencies for capturing data inconsistencies. *ACM Trans. Database Syst.* 33, 2 (2008), 6:1–6:48.
[22] Stephen A. Fenner, Lance Fortnow, and Stuart A. Kurtz. 1994. Gap-definable counting classes. *J. Comput. Syst. Sci.* 48, 1 (1994), 116–148.
[23] Amélie Gheerbrant and Cristina Sirangelo. 2019. Best answers over incomplete data : Complexity and first-order rewritings. In *IJCAI*. 1704–1710.
[24] Sanjay Gupta. 1991. The power of witness reduction. In *[1991] Proceedings of the Sixth Annual Structure in Complexity Theory Conference*. IEEE, 43–59.
[25] Tomasz Imieliński and Witold Lipski, Jr. 1984. Incomplete information in relational databases. *J. ACM* 31, 4 (1984), 761–791.
[26] François Jaeger, Dirk L Vertigan, and Dominic JA Welsh. 1990. On the computational complexity of the Jones and Tutte polynomials. In *Mathematical Proc. of the Cambridge Phil. Soc.*, Vol. 108. Cambridge University Press, 35–53.
[27] Mark R Jerrum, Leslie G Valiant, and Vijay V Vazirani. 1986. Random generation of combinatorial structures from a uniform distribution. *TCS* 43 (1986), 169–188.
[28] Ker-I Ko. 1982. Some observations on the probabilistic algorithms and NP-hard problems. *Inf. Process. Lett.* 14, 1 (1982), 39–43.
[29] Johannes Köbler, Uwe Schöning, and Jacobo Torán. 1989. On counting and approximation. *Acta Informatica* 26, 4 (1989), 363–379.
[30] Leonid Libkin. 2014. Incomplete data: what went wrong, and how to fix it. In *PODS*. 1–13.
[31] Leonid Libkin. 2018. Certain answers meet zero-one laws. In *PODS*. 195–207.
[32] Dany Maslowski and Jef Wijsen. 2013. A dichotomy in the complexity of counting database repairs. *JCSS* 79, 6 (2013), 958–983.
[33] Dany Maslowski and Jef Wijsen. 2014. Counting database repairs that satisfy conjunctive queries with self-joins.. In *ICDT*. 155–164.
[34] Mikaël Monet. 2019. A variant of #POSITIVE-2-DNF. https://cstheory.stackexchange.com/q/43888/38111
[35] Mitsunori Ogiwara and Lane A. Hemachandra. 1993. A complexity theory for feasible closure properties. *J. Comput. Syst. Sci.* 46, 3 (1993), 295–325.

[36] J Scott Provan and Michael O Ball. 1983. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM J. Comput.* 12, 4 (1983), 777–788.
[37] Raymond Reiter. 1978. *On closed world data bases.* Springer US, 55–76.
[38] Anish Das Sarma, Omar Benjelloun, Alon Y. Halevy, Shubha U. Nabar, and Jennifer Widom. 2009. Representing uncertain data: models, properties, and algorithms. *VLDB J.* 18, 5 (2009), 989–1019.
[39] Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. 2011. *Probabilistic databases.* Morgan & Claypool.
[40] Seinosuke Toda and Osamu Watanabe. 1992. Polynomial time 1-Turing reductions from #PH to #P. *Theor. Comput. Sci.* 100, 1 (1992), 205–221.
[41] Leslie G. Valiant. 1976. Relative complexity of checking and evaluating. *Inf. Process. Lett.* 5, 1 (1976), 20–23.
[42] Leslie G Valiant. 1979. The complexity of computing the permanent. *TCS* 8, 2 (1979), 189–201.
[43] Leslie G. Valiant. 1979. The complexity of enumeration and reliability problems. *SIAM J. Comput.* 8, 3 (1979), 410–421.
[44] Ron van der Meyden. 1998. Logical approaches to incomplete information: A survey. In *Logics for databases and information systems.* Springer, 307–356.
[45] Moshe Y Vardi. 1982. The complexity of relational query languages. In *STOC.* ACM, 137–146.