Conjunctive Queries on Probabilistic Graphs: Combined Complexity

Antoine Amarilli
LTCI, Télécom ParisTech, Université Paris-Saclay
antoine.amarilli@telecom-paristech.fr

Mikaël Monet
LTCI, Télécom ParisTech, Université Paris-Saclay
mikael.monet@telecom-paristech.fr

Pierre Senellart
DI, École normale supérieure, PSL Research University
& Inria Paris
pierre.senellart@telecom-paristech.fr

Query evaluation over probabilistic databases is known to be intractable in many cases, even in data complexity, i.e., when the query is fixed. Although some restrictions of the queries [19] and instances [4] have been proposed to lower the complexity, these known tractable cases usually do not apply to combined complexity, i.e., when the query is not fixed. This leaves open the question of which query and instance languages ensure the tractability of probabilistic query evaluation in combined complexity.

This paper proposes the first general study of the combined complexity of conjunctive query evaluation on probabilistic instances over binary signatures, which we can alternatively phrase as a probabilistic version of the graph homomorphism problem, or of a constraint satisfaction problem (CSP) variant. We study the complexity of this problem depending on whether instances and queries can use features such as edge labels, disconnectedness, branching, and edges in both directions. We show that the complexity landscape is surprisingly rich, using a variety of technical tools: automata-based compilation to d-DNNF lineages as in [4], β-acyclic lineages using [10], the $\underline{X}$-property for tractable CSP from [24], graded DAGs [27] and various coding techniques for hardness proofs.

1. Introduction

Uncertainty naturally arises in many data management applications, when integrating data that may be untrustworthy, erroneous, or outdated; or when generating or annotating data using information extraction or machine learning approaches. The framework of *probabilistic databases* [31] has been introduced to answer such needs: it provides a natural semantics for concise representations of probability distributions on data, and allows the user to evaluate queries directly on the representations. The simplest probabilistic framework is that of *tuple-independent databases* (TID), where each tuple in the relational database is annotated with a probability of actually being present, assuming independence across all tuples. Evaluating a Boolean query Q over a TID instance I means computing the probability that Q is true according to the distribution of I, or in other words, the total probability mass of the possible worlds of I that satisfy Q.

As is usual in database theory, the complexity of this probabilistic query evaluation problem (PQE) can be measured as a function of both I and Q, namely, *combined complexity* [33], or as a function of I when the query Q is fixed, called *data complexity*. Almost all works on PQE so far have focused on data complexity, where they have explored the general intractability of PQE in this sense. Indeed, while non-probabilistic query evaluation of fixed queries in first-order logic has polynomial-time data complexity (specifically, AC^0), the PQE problem is #P-hard[1] already for some fixed conjunctive queries [18]. Specifically, the celebrated PQE result by Dalvi and Suciu [19], has shown a dichotomy on unions of conjunctive queries: some are *safe queries*, enjoying PTIME data complexity (specifically, linear [14]), and all other queries are #P-hard. Earlier work by some of the present authors has shown also a dichotomy on instance families for fixed monadic second-order queries, with tractable data complexity for bounded-treewidth families [4], and intractability otherwise under some assumptions [6].

However, even when PQE is tractable in data complexity, the task may still be infeasible because of unrealistically large constants that depend on the query. For instance, our approach in [4] is nonelementary in the query, and the algorithm for safe queries in [19] is generally super-exponential in the query [31]. For this reason, we believe that it is also important to achieve a good understanding of the *combined* complexity of PQE, and to isolate cases where PQE is tractable in combined complexity; similarly to how, e.g., Yannakakis' algorithm can evaluate α-acyclic queries on non-probabilistic instances with tractable combined complexity [37]. This motivates the question studied in this paper: *for which classes of queries and instances does PQE enjoy tractable combined complexity?*

Related work. Surprisingly, the question of achieving combined tractability for PQE does not seem to have been studied before. To our knowledge, the only exception is in the setting of probabilistic XML [26], where deterministic tree automata queries were shown to enjoy tractable combined complexity [15]. In the context of relational databases, our

[1] #P is the class of counting problems that can be expressed as the number of accepting paths of a nondeterministic polynomial-time Turing machine.

recent work [3] shows the combined tractability of *provenance computation* for a specific Datalog fragment on bounded-treewidth instances, but observes that these results do not seem to give tractability of PQE, which is already intractable in much more restricted settings. These results, however (Propositions 36 and 38 of [2]), do not give a complete picture of the combined complexity of PQE; in particular, they do not even give any non-trivial setting where it is tractable.

Questions of combined tractability have also been studied in the setting of *constraint satisfaction problems* (CSP), following a well-known connection between CSP and the conjunctive query evaluation problem in database theory, or the study of the graph homomorphism problem (see, e.g., [23]). We can then see the restriction of PQE to conjunctive queries as a probabilistic, or weighted, variant of these problems, but we are not aware of any existing study of this variant. In the graph homomorphism setting, a related but different problem is that of *counting* graph homomorphisms [11]: but this amounts to counting the number of matches of a query in a database instance, which is different from counting the possible worlds of an instance where the query has some match, as we do. A more related problem is `#SUB` [17], which asks, given a query graph G and an instance graph H, for the *number of subgraphs* of H which are *isomorphic* to G. When all facts are labeled with $1/2$, our problem asks instead for the number of subgraphs of H to which G admits a *homomorphism*. A further difference is that we allow arbitrary probability annotations, amounting to a form of weighted counting; in particular, facts can be given probability 1.

Problem statement. Inspired by the connection to graph homomorphism and CSP, in this paper we investigate the probabilistic query evaluation problem for conjunctive queries on tuple-independent instances, over arity-two signatures. To our knowledge, our paper is the first to focus on the combined complexity of conjunctive query evaluation on probabilistic relational data. For simplicity of exposition, we will phrase our problem in terms of graphs: given a *query graph* and a probabilistic *instance graph*, where each edge is annotated by a probability, we must determine the probability that the query graph has a homomorphism to the instance graph, i.e., the total probability mass of the subgraphs which ensure this, assuming independence between edges. We always assume the query and instance graphs to be directed.

As we will see, the problem is generally intractable, so we will have to study restricted settings. We accordingly study this problem under assumptions on the query and input graphs. Inspired by our prior intractability results [6], one general assumption that we will make is to impose *tree-likeness* of the instance. In fact, we will generally restrict it to be a *polytree*, i.e., a directed graph whose underlying undirected graph is a tree. As we will see, however, even this restriction does not suffice to ensure tractability, so we study the impact of several other features:

- *Labels*, i.e., whether edges of the query and instance can be labeled by a finite alphabet, as would be the case on a relational signature with more than one binary predicate.

- *Disconnectedness*, i.e., allowing disconnected queries and instances.
- *Branching*, i.e., allowing graphs to branch out, instead of requiring them to be a path.
- *Two-wayness*, i.e., allowing edges with arbitrary orientation, instead of requiring all edges to have the same orientation (as in a one-way path, or downward tree).

We accordingly study our problem for *labeled graphs* and *unlabeled graphs*, and when query and instance graphs are in the following classes, that cover the possible combinations of the above characteristics: one-way and two-way paths, downward trees and polytrees, and disjoint unions thereof.

Results. This paper presents our combined complexity results for the probabilistic query evaluation problem in all these settings. After introducing the preliminaries and defining the problem in Section 2, we first study the impact of disconnectedness in instances and queries in Section 3. While we can easily show that disconnectedness does not matter for instances (Lemma 3.7), we show that disconnectedness of queries has an unexpected impact on complexity: in the labeled case, even the simplest disconnected queries on the simplest kinds of instances are intractable (Proposition 3.3): this result is shown via the hardness of counting edge covers in bipartite graphs. The picture for disconnected queries is more complex in the unlabeled case (see Table 1): indeed, the problem is still hard when allowing two-wayness in the query and instance (as it can be used to simulate labels, see Proposition 3.4), but disallowing two-wayness in the instance ensures tractability of all queries. This latter result (Proposition 3.6) is established by showing that all queries then essentially collapse to a one-way path: we do so by assigning a *level* to all vertices of the query using a notion of *graded DAGs* [27, 29].

We then focus on connected queries, and first study the labeled setting in Section 4; see Table 2 for a summary of results. We show that disallowing instance branching ensures the tractability of all connected queries (Proposition 4.11), and that disallowing branching in the query *and* two-wayness in the instance and query also does (Proposition 4.10). These two results are shown by computing a *Boolean lineage* of the query [31], and proving that we can tractably evaluate its probability because it is β-acyclic [10], thanks to the restricted instance structure. For the first result, this process further relies on a CSP tool to show the tractability of homomorphism testing in labeled two-way paths, a condition dubbed the $\underline{X}$-property [24, 22]. We show the intractability of all other cases (Propositions 4.1, 4.4, and 4.5), by coding #SAT-reduction, reusing in part a coding from [2].

We last study the unlabeled setting for connected queries in Section 5. We show that disallowing query branching and two-wayness suffices to obtain tractability, provided that the instance is a polytree (Proposition 5.4): this result is proven by building in PTIME a deterministic tree automaton to test the length of the longest path, and compiling a d-DNNF lineage as in [4]. This result immediately extends to branching queries, as they are equivalent to paths in this case (Proposition 5.5). We complete the picture by showing that, by contrast, allowing two-wayness in the query leads to

intractability on polytrees, by a variant of our coding technique (Proposition 5.6). We then conclude in Section 6.

Our results completely classify the complexity of probabilistic conjunctive query evaluation for all combinations of instance and query restrictions, in the labeled and unlabeled setting. Full proofs are given in appendix.

2. Preliminaries

We first provide some formal definitions of the concepts we use in this paper, and introduce the *probabilistic graph homomorphism* problem and the different classes of graphs that we consider.

Graphs and homomorphisms. Let σ be a finite non-empty set of labels. When $|\sigma| > 1$, we say that we are in the *labeled setting*; when $|\sigma| = 1$, in the *unlabeled setting*.

We consider *directed graphs* with edge labels from σ, i.e., triples $H = (V, E, \lambda)$ with V a non-empty finite set of vertices, $E \subseteq V^2$ a set of edges, and $\lambda : E \to \sigma$ a labeling function. We write $a \xrightarrow{R} b$ for an edge $e = (a, b)$ with label $\lambda(e) = R$. Note that we do not allow multi-edges: an edge e has a unique label $\lambda(e)$. When $|\sigma| = 1$, i.e., in the unlabeled setting, we simply write (V, E) for the graph and $a \to b$ for an edge. Unless otherwise specified, all graphs that we consider in this paper are directed.

A graph $H' = (V', E', \lambda')$ is a *subgraph* of the graph $H = (V, E, \lambda)$, written $H' \subseteq H$, when we have $V' = V$, $E' \subseteq E$, and when λ' is $\lambda_{|E'}$, i.e., the restriction of λ to E'. (Note that, in a slightly non-standard way, we impose that subgraphs have the same set of vertices than the original graph; this will simplify some notation.)

A *graph homomorphism* h from some graph $G = (V_G, E_G, \lambda_G)$ to some graph $H = (V_H, E_H, \lambda_H)$ is a function $h : V_G \to V_H$ such that, for all $(u, v) \in E_G$, we have $(h(u), h(v)) \in E_H$ and further $\lambda_H((h(u), h(v))) = \lambda_G((u, v))$. A *match* of G in H is the image in H of such a homomorphism h, i.e., the graph with vertices $h(u)$ for $u \in V_G$ and edges $(h(u), h(v)))$ for $(u, v) \in E_G$. Note that two different homomorphisms may define the same match. Also note that two distinct nodes of G could have the same image by h, so a match of G in H is not necessarily homomorphic to G. We write $G \rightsquigarrow H$ when there exists a homomorphism from G to H. We call two graphs G and G' *equivalent* if, for any graph H, we have $G \rightsquigarrow H$ iff $G' \rightsquigarrow H$. It is easily seen that G and G' are equivalent if and only if $G \rightsquigarrow G'$ and $G' \rightsquigarrow G$.

Probabilistic graphs. A *probability distribution on graphs* is a function Pr from a finite set $\mathcal{W}$ of graphs (called the *possible worlds* of Pr) to values in $[0; 1]$ represented as rational numbers, such that the probabilities of all possible worlds sum to 1, namely, $\sum_{H \in \mathcal{W}} \Pr(H) = 1$.

A *probabilistic graph* is intuitively a concise representation of a probability distribution. Formally, it is a pair (H, π) where H is a graph with edge labels from σ and where π is a probability function $\pi : E \to [0; 1]$ that maps every edge e of H to a probabil-

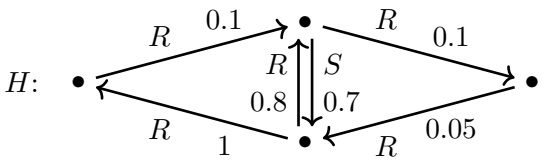


Figure 1: Example probabilistic graph H

ity $\pi(e)$, represented as a rational number. Note that each edge (u, v) in a probabilistic graph (H, π) is annotated both with a label $\lambda((u, v)) \in \sigma$, and a probability $\pi((u, v))$.

The *probability distribution* Pr defined by the probabilistic graph (H, π) is obtained intuitively by considering that edges are kept or deleted independently according to the indicated probability. Formally, the possible worlds $\mathcal{W}$ of Pr are the subgraphs of $H = (V, E, \lambda)$, and for $H' = (V, E', \lambda_{|E'}) \subseteq H$ we define $\Pr(H') := \prod_{e \in E'} \pi(e) \times \prod_{e \in E \setminus E'} (1 - \pi(e))$. Note that, when H has edges labeled with 0 or 1, some possible worlds are given probability 0 by π.

Example 2.1. *Figure 1 represents a probabilistic graph (H, π) on signature $\sigma = \{R, S\}$, where each edge is annotated with its label and probability value. There are 2^6 possible worlds, 2^5 of which have non-zero probability.*

The possible world where all R-edges are kept and all S-edges are removed has probability $0.1 \times 1 \times 0.8 \times 0.1 \times 0.05 \times (1 - 0.7)$.

Probabilistic graph homomorphism. The goal of this paper is to study the *probabilistic homomorphism problem* PHom, for the set of labels σ that we fixed: given a graph G on σ and a probabilistic graph (H, π) on σ, compute the probability that there exists a homomorphism from G to H under Pr, i.e., the sum of the probabilities of all subgraphs of H' to which G has a homomorphism:

$$\Pr(G \rightsquigarrow H) := \sum_{\substack{H' \subseteq H \\ G \rightsquigarrow H'}} \Pr(H').$$

Example 2.2. *Continuing the example, let us consider the* PHom *problem for the graph $G : \xrightarrow{R} \xrightarrow{S} \xleftarrow{S}$ and the example probabilistic graph (H, π) in Figure 1. The graph G intuitively corresponds to the relational calculus query $\exists xyzt\ R(x, y) \wedge S(y, z) \wedge S(t, z)$. Of course, we can compute $\Pr(G \rightsquigarrow H)$ by summing over the possible worlds of H, but this process is generally intractable. Here, by considering the possible matches of G in H, we can see that $\Pr(G \rightsquigarrow H) = 0.7 \times (1 - (1 - 0.1) \times (1 - 0.8))$.*

Following database terminology, we call G the *query graph* and (H, π) the *(probabilistic) instance graph*. Indeed, the PHom problem is easily seen to be equivalent to conjunctive query evaluation on probabilistic tuple-independent relational databases [19], over binary relational signatures.

Note that, in this paper, we measure the complexity of PHom as a function of *both* the query graph G and of the instance graph (H, π), i.e., in database terminology, we measure

Figure 2: Inclusions between classes of graphs

the *combined complexity* [33] of probabilistic query evaluation. The PHom problem is known to be #P-hard in general [18] (even for some fixed query graphs): by this, we mean that it is hard (under polynomial-time reductions) for the class #P of counting problems that can be expressed as the number of accepting paths of a nondeterministic polynomial-time Turing machine. To achieve tractable complexity for PHom, we will classify the complexity of PHom under various restrictions. We say that the complexity of some variant of the problem is *tractable* if the probability can be computed by a deterministic polynomial-time Turing machine: by a slight abuse of terminology, we then say that it is *in PTIME*. All PHom variants that we study will be shown either to be PTIME in this sense, or to be #P-hard.

We will study restrictions of PHom first by distinguishing the *labeled* and *unlabeled* settings. We write $\mathsf{PHom}_{\mathsf{L}}$ for the problem when the fixed label set σ is such that $|\sigma| > 1$, and $\mathsf{PHom}_{\not\mathsf{L}}$ when the fixed σ is such that $|\sigma| = 1$.

The second restriction concerns the input query graphs and instance graphs. We will model restrictions on these graphs by requiring them to be taken from specific graph classes, where by *graph class* we simply mean an infinite set of graphs. Inspired by the notation used in CSP, for two classes $\mathcal{G}$ and $\mathcal{H}$ of graphs in the labeled setting, we denote $\mathsf{PHom}_{\mathsf{L}}(\mathcal{G}, \mathcal{H})$ the problem that takes as input a graph G in class $\mathcal{G}$ and a probabilistic graph (H, π) with H in class $\mathcal{H}$, and computes the probability $\Pr(G \rightsquigarrow H)$. We denote the same problem in the unlabeled setting by $\mathsf{PHom}_{\not\mathsf{L}}(\mathcal{G}, \mathcal{H})$.

Graph classes. The graph classes which we study in this paper are defined as follows, on a graph G with edge labels from σ:

- G is a *one-way path* (1WP) if it is of the form $a_1 \xrightarrow{R_1} \cdots \xrightarrow{R_{m-1}} a_m$ for some m, with all $a_1, \ldots, a_m$ being pairwise distinct, and with $R_i \in \sigma$ for $1 \leqslant i < m$.
- G is a *two-way path* (2WP) if it is of the form $a_1 - \cdots - a_m$, with all $a_1, \ldots, a_m$ being pairwise distinct, and each $-$ being $\xrightarrow{R_i}$ or $\xleftarrow{R_i}$ (but not both) for some label $R_i \in \sigma$.
- G is a *downwards tree* (DWT) if it is a rooted unranked tree (each node can have an arbitrary number of children), with all edges going from parent to child in the tree.
- G is a *polytree* (PT) if its underlying undirected graph is an unranked tree, without restriction on edge directions.

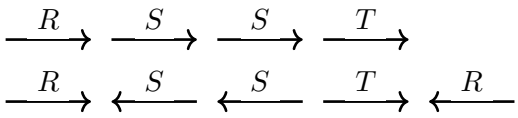


Figure 3: Example of labeled 1WP (top) and 2WP (bottom) for $\sigma = \{R, S, T\}$.

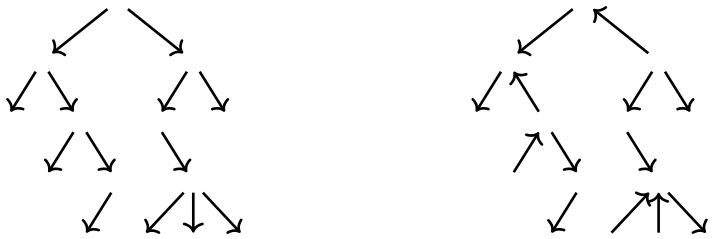


Figure 4: Examples of unlabeled DWT (left) and PT (right)

We also consider the class Connected of connected graphs, and write All the class of all graphs. The inclusion diagram between our graph classes is shown in Figure 2. See Figure 3 for an example of a labeled one-way path and two-way path, and Figure 4 for an unlabeled downwards tree and polytree.

We also introduce the classes $\bigsqcup$1WP (resp., $\bigsqcup$2WP, $\bigsqcup$DWT, $\bigsqcup$PT) of graphs that are *disjoint unions of* 1WP (resp., 2WP, DWT, PT), that is, of possibly disconnected graphs whose connected components are 1WP (resp., 2WP, DWT, PT).

Our graph classes were chosen to be representative of different features of graphs that will have an impact in the complexity of the PHom problem, namely, *labeling*, *two-wayness*, *branching*, and *disconnectedness*. Indeed, 2WP (resp., PT) adds two-wayness to 1WP (resp., DWT); DWT (resp., PT) adds branching to 1WP (resp., 2WP); and $\bigsqcup$1WP (resp., $\bigsqcup$2WP, $\bigsqcup$DWT, $\bigsqcup$PT) adds disconnectedness to 1WP (resp., 2WP, DWT, PT).

In the following sections, we investigate the complexity of probabilistic graph homomorphism for these various classes of conjunctive queries and instances.

3. Disconnected Case

We first consider the case where either the query or probabilistic instance graph is *disconnected*, i.e., not in the Connected class. When the query is disconnected, we show in this section that the probabilistic homomorphism problem is #P-hard in all but the most restricted of cases (in particular in the labeled setting), which justifies that we restrict to connected queries in the rest of the paper. On the other hand, we will show that disconnectedness in the probabilistic instance graph has essentially no impact on combined complexity.

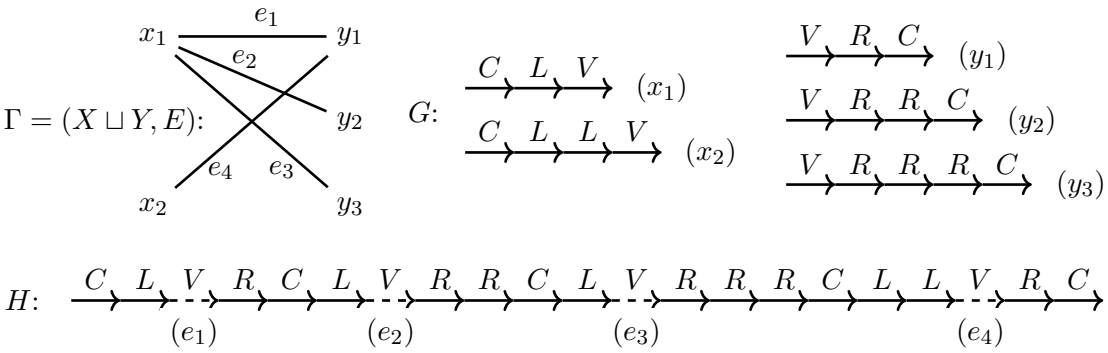


Figure 5: Illustration of the proof of Proposition 3.3, for the bipartite graph Γ. Dashed edges have probability $\frac{1}{2}$. We show (in parentheses) the edge of Γ coded by each V-labeled edge in the instance graph H, and the vertex of Γ coded by each 1WP component of the query graph G.

3.1. Labeled Disconnected Queries

We establish our main intractability result on disconnected queries by reduction from the `#Bipartite-Edge-Cover` problem on *undirected* graphs:

Definition 3.1. *An undirected graph is* bipartite *if its vertices can be partitioned into two classes such that no edge connects two vertices of the same class. An* edge cover *of an undirected graph is a subset of its edges such that every vertex is incident to at least one edge of the subset.* `#Bipartite-Edge-Cover` *is the problem, given a bipartite undirected graph, of counting its number of edge covers.*

This problem was shown in [25] to be intractable. The result can also be proven using Valiant's holographic reductions [32] and the results of Cai, Lu, and Xia [13]: see Appendix D.

Theorem 3.2. *[25, 13] The* `#Bipartite-Edge-Cover` *problem is #P-complete.*

We can then use this result to show intractability for the simplest forms of disconnected query graphs ($\bigsqcup$1WP) on the simplest forms of probabilistic instance graphs (1WP), in the *labeled* case:

Proposition 3.3. $\mathsf{PHom_L}(\bigsqcup \mathsf{1WP}, \mathsf{1WP})$ *is #P-hard.*

Proof. We reduce from `#Bipartite-Edge-Cover`. Let $\Gamma = (X \sqcup Y, E)$ be an input to `#Bipartite-Edge-Cover`, i.e., a bipartite undirected graph with parts X and Y; we write $X = (x_1, \ldots, x_{n_\mathrm{l}})$, $Y = (y_1, \ldots, y_{n_\mathrm{r}})$, $E = (e_1, \ldots, e_m)$, and for all $1 \leqslant i \leqslant m$ we write $e_i = (x_{l_i}, y_{r_i})$, with $1 \leqslant l_i \leqslant n_\mathrm{l}$ and $1 \leqslant r_i \leqslant n_\mathrm{r}$.

We first construct in PTIME the 1WP probabilistic graph (H, π): see Figure 5 for an illustration of the construction. Specifically, for $1 \leqslant j \leqslant m$, we construct the following 1WP:

$$H_{e_j} := (\xrightarrow{L})^{l_j} \xrightarrow{V} (\xrightarrow{R})^{r_j}.$$

The graph H is then defined as:

$$\xrightarrow{C} H_{e_1} \xrightarrow{C} H_{e_2} \xrightarrow{C} \cdots \xrightarrow{C} H_{e_m} \xrightarrow{C} .$$

We define π as follows: edges labeled by V have probability $\frac{1}{2}$ (intuitively coding whether an edge is part of the candidate cover), all others have probability 1.

We then construct the query graph $G \in \bigsqcup\mathsf{1WP}$, coding the edge covering constraints. For every $1 \leqslant i \leqslant n_{\mathrm{l}}$, the graph G contains the $\mathsf{1WP}$ component $\xrightarrow{C} (\xrightarrow{L})^i \xrightarrow{V}$, and for every $1 \leqslant i \leqslant n_{\mathrm{r}}$, the graph G contains the $\mathsf{1WP}$ component $\xrightarrow{V} (\xrightarrow{R})^i \xrightarrow{C}$.

It is clear that H is in $\mathsf{1WP}$, G is in $\bigsqcup\mathsf{1WP}$ and that both can be constructed in PTIME from Γ. We now show that $\Pr(G \rightsquigarrow H)$ is exactly the number of edge covers of Γ divided by 2^m, so that the computation of the latter reduces in PTIME to the computation of the former, concluding the proof.

To see why, we define a bijection between the subsets of edges of Γ, seen as valuations $\nu : E \to \{0, 1\}$, to the possible worlds H' of H of non-zero probability. We do so in the expected way: keep the one V-edge $\xrightarrow{V}$ of H_{e_i} iff $\nu(e_i) = 1$. We now show that there is a homomorphism from G to H' if and only if ν is an edge cover of Γ. As the number of H''s such that there is a homomorphism from G to H' is exactly $\Pr(G \rightsquigarrow H) \times 2^m$, this will allow us to conclude.

Indeed, if there is a homomorphism h from G to H', then, considering the $\mathsf{1WP}$ component in G that codes the constraint on x_i (resp., on y_i), its image must be of the form $\xrightarrow{C} (\xrightarrow{L})^i \xrightarrow{V}$ (resp., $\xrightarrow{V} (\xrightarrow{R})^i \xrightarrow{C}$), but then by construction of H the V-fact must correspond to an edge e such that x_i (resp., y_i) is adjacent to e, so that we have $\nu(e) = 1$ and so x_i (resp., y_i) is covered. As this is true for each $\mathsf{1WP}$ component, all the vertices are covered and ν is indeed an edge cover of Γ.

Conversely, suppose that ν is an edge cover of Γ, then for every vertex x_i (resp., y_i) we know that there exists $1 \leqslant j \leqslant m$ such that $\nu(e_j) = 1$ and $l_j = i$ (resp., $r_j = i$), and we can use the V-fact corresponding to e_j and the surrounding facts to build the homomorphism as above from each component of G to H'. □

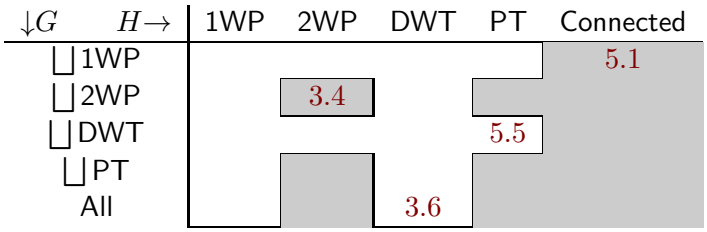
The proof of Proposition 3.3 crucially requires multiple labels in the signature. Indeed, it is easy to see that, in the unlabeled setting, a query graph in $\bigsqcup\mathsf{1WP}$ (or even in $\bigsqcup\mathsf{DWT}$) is equivalent to the longest path within the graph, and we will show further (Proposition 5.5) that $\mathsf{PHom}_{\not\mathsf{L}}(\mathsf{1WP}, \mathsf{1WP})$ (indeed, even $\mathsf{PHom}_{\not\mathsf{L}}(\bigsqcup\mathsf{DWT}, \mathsf{PT})$) is PTIME.

3.2. Unlabeled Disconnected Queries

In light of this intractability result, let us now consider the unlabeled setting. We show in Table 1 where the tractability frontier lies. First, introducing two-wayness in both query and instance graphs is enough to obtain an analogue of the intractability of Proposition 3.3:

Proposition 3.4. $\mathsf{PHom}_{\not\mathsf{L}}(\bigsqcup\mathsf{2WP}, \mathsf{2WP})$ *is #P-hard.*

Table 1: Tractability of $\mathsf{PHom}_{\not\mathsf{L}}$ for disconnected queries (Section 3.2). Results also hold when instances are unions of the indicated classes.



PTIME #P-hard Numbers given correspond to propositions for border cases, remaining cells can be filled using the inclusions from Figure 2.

Proof. We reduce, again, from the #P-hard problem `#Bipartite-Edge-Cover`. The idea of the reduction is similar to that used in the proof of Proposition 3.3, but we face the additional difficulty of not being allowed to use labels. Fortunately, we can use two-wayness to simulate them.

Let $\Gamma = (X \sqcup Y, E)$ be an input of `#Bipartite-Edge-Cover`. Consider the reduction from Γ used in the proof of Proposition 3.3 and the 1WP probabilistic graph (H, π) and the $\bigsqcup$1WP query graph G that were constructed. We construct from H and G the unlabeled probabilistic graph H' and unlabeled $\bigsqcup$2WP query graph G' as follows:

- replace each L- or R-labeled edge $a \xrightarrow{L} b$ or $a \xrightarrow{R} b$ in H and G by 3 edges $a \rightarrow\rightarrow\leftarrow b$;
- replace each C-labeled edge $a \xrightarrow{C} b$ of H and G by 3 edges $a \leftarrow\leftarrow\leftarrow b$;
- replace each V-labeled edge $a \xrightarrow{V} b$ of H and G by 6 edges $a \rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\leftarrow b$.

All edges of H' have probability 1, except the first edge of each sequence of 6 edges that replaced a V-labeled edge, which has probability $\frac{1}{2}$.

Consider a 1WP component of G that codes the constraint on a vertex from Y, e.g $\xrightarrow{V} (\xrightarrow{R})^i \xrightarrow{C}$, which was rewritten in G' into $\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\leftarrow (\rightarrow\rightarrow\leftarrow)^i \leftarrow\leftarrow\leftarrow$. A homomorphism from this component into a possible world J' of H' must actually map to a rewriting of a $\xrightarrow{V} (\xrightarrow{R})^i \xrightarrow{C}$ sequence in H': indeed, the key observation is that the first 5 $\rightarrow$ edges can only be matched to 5 consecutive $\rightarrow$ in J', which only exist as the first 5 edges of a sequence of 6 edges that replaced a V-labeled fact in H. There is no choice left to match the subsequent edges without failing. A similar observation holds for components coding the constraints on vertices from X ($\xrightarrow{C} (\xrightarrow{L})^i \xrightarrow{V}$). Hence, we can show correctness of the reduction using the same argument as before. □

Allowing two-wayness in both the query and the instance graphs thus allows us to simulate labels, so that $\mathsf{PHom}_{\not\mathsf{L}}$ is intractable. We will study in Section 5 what happens

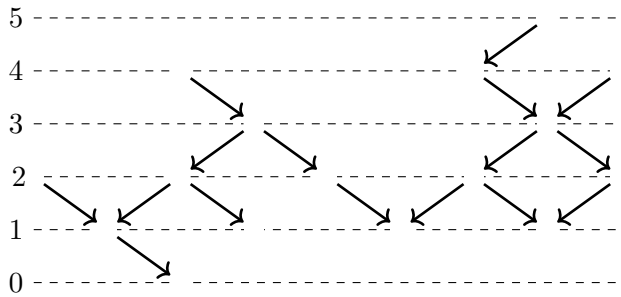


Figure 6: A DAG with a level mapping (dashed lines), see Definition 3.5.

for query graph classes without two-wayness (i.e., 1WP, DWT, and unions thereof); so let us now consider the case of instance graph classes where two-wayness is forbidden, i.e., is in $\bigsqcup$DWT. As we will show, PHom$_{\not\perp}$ of *arbitrary* query graphs on such $\bigsqcup$DWT instance graphs is tractable. To this end, we need to introduce *level mappings* of acyclic directed graphs (DAGs):

Definition 3.5. *A* level mapping *of a DAG G is a mapping μ from the vertices of G to $\mathbb{Z}$ such that for each directed edge $u \to v$ of G we have $\mu(v) = \mu(u) - 1$. We call G a* graded DAG *if it has a level mapping.*

An example of graded DAG together with a level mapping is given in Figure 6. It is easy to see (and shown in Proposition 1 of [27]) that a DAG G is graded iff there are no two vertices u, v and two directed paths χ, χ' in G from u to v such that χ and χ' have different lengths (in the terminology of [27], G does not have a *jumping edge*). Graded DAGs are related to the classical notion of graded ordered set [29], and the level mapping function has been called in the literature a *depth function* [27], a *grading function* [29], a *set of levels* [29], or a *rank function* [30].

To obtain such a level mapping, we can proceed by picking one vertex in each connected component of G, mapping each of these vertices to level 0, and then exploring G by a breadth-first traversal and assigning the level of each vertex according to the level of the vertex used to reach it, visiting all edges and defining the image of each vertex. It is clear that this process yields a level mapping of G unless it tries to assign two different levels to the same vertex v, which cannot happen if there is no jumping edge [27, Proposition 1].

We will now use the notion of graded DAG to show:

Proposition 3.6. PHom$_{\not\perp}$(All, $\bigsqcup$DWT) *is PTIME.*

Proof sketch. We only give the idea when the query graph is connected and the graph instance H is a DWT (see Appendix for full proof). As we pointed out already, if the query graph G is not a graded DAG, then it has a cycle or a pair of vertices joined by two directed paths of different lengths: then, from the structure of the DWT instance graph, this clearly implies that $\Pr(G \rightsquigarrow H) = 0$. So it suffices to study the case when G is a graded DAG.

As we explained earlier, we can then compute in PTIME a level mapping μ of G. It is clear that, as G is connected, the level mapping μ is uniquely defined up to an additive constant. Hence, we shift μ so that the smallest value of its image is 0, and we then call the *difference of levels* of G the largest value m in the image of μ. Note that m is *not* the maximal length from a root of G to a leaf of G (see, e.g., Figure 6). We then claim that, on any possible world H' of the DWT instance graph H, the query graph G is in fact equivalent to the 1WP query graph $\rightarrow^m$ of length m. This allows us to conclude using Proposition 5.5.

One direction is easy to observe, because μ directly gives a homomorphism from G to $\rightarrow^m$. For the converse, suppose that a homomorphism h from G to H' exists. Because G is connected and H' is in $\bigsqcup$DWT, the image of h is actually a DWT, call it T. Now it is easy to see that the image of a node that has level $m - i$ in G has depth i in T, so that T (and so H') contains the 1WP $\rightarrow^m$. □

3.3. Disconnected Instances

We conclude our study of the disconnected case with the case of disconnected *instance graphs*, which we show to be less interesting than the disconnected *query graphs* that we studied so far. Specifically, when the query is *connected*, PHom on arbitrary instances can reduce in PTIME to PHom of the same queries on a corresponding class of connected instances:

Lemma 3.7. *For any class of graphs $\mathcal{H}$, let $\mathcal{H}'$ be the class of connected components of graphs in $\mathcal{H}$. Then for any class of* connected *graphs $\mathcal{G}$, $\mathsf{PHom}_{\mathsf{L}}(\mathcal{G}, \mathcal{H})$ reduces in PTIME to $\mathsf{PHom}_{\mathsf{L}}(\mathcal{G}, \mathcal{H}')$, and $\mathsf{PHom}_{\not\mathsf{L}}(\mathcal{G}, \mathcal{H})$ reduces in PTIME to $\mathsf{PHom}_{\not\mathsf{L}}(\mathcal{G}, \mathcal{H}')$.*

Proof. Let $G \in \mathcal{G}$, $H \in \mathcal{H}$, and write $H = H'_1 \sqcup \ldots \sqcup H'_n$: we have $H'_i \in \mathcal{H}'$ for all $1 \leqslant i \leqslant n$. Let π be a probability distribution over H: the independence assumption ensures that the edges of any H'_i are pairwise independent from those of any H'_j for $i \neq j$. Now, as G is connected, any image of a homomorphism from G to H must actually be included in some H'_i. Thus, the computation of $\Pr(G \rightsquigarrow H)$ reduces to that of the $\Pr(G \rightsquigarrow H'_i)$ for $1 \leqslant i \leqslant n$, as follows:

$$\Pr(G \rightsquigarrow H) = 1 - \prod_{1 \leqslant i \leqslant n} (1 - \Pr(G \rightsquigarrow H'_i)).$$ □

We last discuss the case when both the query and instance graphs are disconnected. Let us consider the results of Table 1 for connected instance graphs. Clearly, any hardness results of a connected class carries over to the corresponding disconnected class. Conversely, we have shown in Proposition 3.6 that $\mathsf{PHom}_{\not\mathsf{L}}(\mathsf{All}, \bigsqcup\mathsf{DWT})$ is PTIME; this implies that all tractable cases in Table 1 also hold for unions of the indicated instance classes, except $\mathsf{PHom}_{\not\mathsf{L}}(\bigsqcup\mathsf{1WP}, \bigsqcup\mathsf{PT})$ and $\mathsf{PHom}_{\not\mathsf{L}}(\bigsqcup\mathsf{DWT}, \bigsqcup\mathsf{PT})$. But we have noted at the end of Section 3.1 that, in the unlabeled setting, $\bigsqcup$1WP or $\bigsqcup$DWT query graphs are equivalent to 1WP query graphs: thus, Lemma 3.7, together with tractability of $\mathsf{PHom}_{\not\mathsf{L}}(\mathsf{1WP}, \mathsf{PT})$, implies $\mathsf{PHom}_{\not\mathsf{L}}(\bigsqcup\mathsf{1WP}, \bigsqcup\mathsf{PT})$ and $\mathsf{PHom}_{\not\mathsf{L}}(\bigsqcup\mathsf{DWT}, \bigsqcup\mathsf{PT})$ are both

Table 2: Tractability of $\mathsf{PHom}_{\mathsf{L}}$ in the connected case (Section 4)

$\downarrow G$ $H\rightarrow$	1WP	2WP	DWT	PT	Connected
1WP			4.10	4.1	
2WP			4.5		
DWT			4.4		
PT					
Connected		4.11			

PTIME #P-hard Numbers given correspond to propositions for border cases, remaining cells can be filled using the inclusions from Figure 2.

in PTIME. Hence, the results of Table 1 also hold when instances are unions of the indicated classes.

We have thus completed our study of $\mathsf{PHom}_{\mathsf{L}}$ and $\mathsf{PHom}_{\not\mathsf{L}}$ for disconnected instances and/or disconnected queries, We accordingly focus on connected queries and instances in the next two sections.

4. Labeled Connected Queries

In this section, we focus on the *labeled* setting, i.e., the $\mathsf{PHom}_{\mathsf{L}}$ problem, for classes of connected queries and instances. Table 2 shows the entire classification of the labeled setting for the classes that we consider.

Intuitively, we show intractability for polytree instance graphs, and for downward trees instance graphs when the query graphs allow either two-wayness or branching. Conversely, we show tractability of one-way path query graphs on downward trees, and of arbitrary connected queries on two-way path instances. We first present the hardness results, and then the tractability results.

4.1. Hardness Results

We recall that, if we allow *arbitrary* connected unlabeled probabilistic instance graphs (or even just 4-partite graphs), then computing the probability that there exists a path of length 2 is already #P-hard: this is shown in [31], and we will state this result in our context as Proposition 5.1 in the next section. Hence, if we want to obtain PTIME complexity for PHom, we need to restrict the class of instances. We can start by restricting the instances to be polytrees, but as we show, this does not suffice to ensure tractability:

Proposition 4.1. $\mathsf{PHom}_{\mathsf{L}}(\mathsf{1WP}, \mathsf{PT})$ *is #P-hard.*

To show this result, we will reduce from the problem of computing the probability of a Boolean formula, which we now define:

Definition 4.2. *Given a set of variables $\mathcal{X}$ and a* probability assignment π *mapping each variable X in $\mathcal{X}$ to a rational probability $\pi(X) \in [0,1]$, we define the* probability $\pi(\nu)$ *of a valuation $\nu : \mathcal{X} \to \{0,1\}$ as*

$$\pi(\nu) := \left(\prod_{X \in \mathcal{X},\ \nu(X)=1} \pi(X)\right)\left(\prod_{X \in \mathcal{X},\ \nu(X)=0} (1-\pi(X))\right).$$

The Boolean probability computation problem *is defined as follows: given a Boolean formula φ on variables $\mathcal{X}$ and a probability assignment π on $\mathcal{X}$, compute the total probability of the valuations that satisfy φ, i.e., $\Pr(\varphi, \pi) = \sum_{\nu \text{ satisfies } \varphi} \pi(\nu)$.*

This problem is known to be #P-hard, even under severe restrictions on the formula φ. We will use the `#PP2DNF` formulation of the above problem, which is #P-hard [28, 31]:

Definition 4.3. *A* positive DNF *is a Boolean formula φ of the form*

$$\varphi = \bigvee_{1 \leqslant i \leqslant m} \left(\bigwedge_{1 \leqslant j \leqslant n_i} X_{i,j}\right),$$

i.e., it is a disjunction of (conjunctive) clauses *that are conjunctions of variables of $\mathcal{X}$. We assume that each variable of $\mathcal{X}$ occurs in φ, as we can eliminate the others without loss of generality.*

A positive partitioned 2-DNF *(PP2DNF) is intuitively a positive DNF φ on a partitioned set of variables where each clause contains one variable from each partition. Formally, the variables of φ are $\mathcal{X} \sqcup \mathcal{Y}$, where we write $\mathcal{X} = \{X_1, \ldots, X_{n_1}\}$ and $\mathcal{Y} = \{Y_1, \ldots, Y_{n_2}\}$, and φ is of the form $\bigvee_{j=1\ldots m}(X_{x_j} \wedge Y_{y_j})$ with $1 \leqslant x_j \leqslant n_1$ and $1 \leqslant y_j \leqslant n_2$ for $1 \leqslant j \leqslant m$.*

The `#PP2DNF` *problem is the Boolean probability computation problem when we impose that π maps every variable to $1/2$, and that φ is a PP2DNF.*

We show Proposition 4.1 by reducing from `#PP2DNF`:

Proof sketch. The full proof is in appendix; see Figure 7 for an illustration. From the PP2DNF formula φ, we construct a PT probabilistic instance where each branch starting at the root describes a variable of the formula. The first edge is probabilistic and represents the choice of valuation. The edges are oriented upwards or downwards depending on whether the variable belongs to $\mathcal{X}$ or to $\mathcal{Y}$. We add a special gadget at different depths of the branch to code the index of each of the clauses where the variable occurs.

We code satisfaction of the formula by a query that tests for a path of a specific length that starts and ends with the gadget. The query has a match exactly on possible worlds where we have set two variables to true such that the sum of the depths of the gadgets corresponds to the query length: this happens iff the two variables occur in the same clause. □

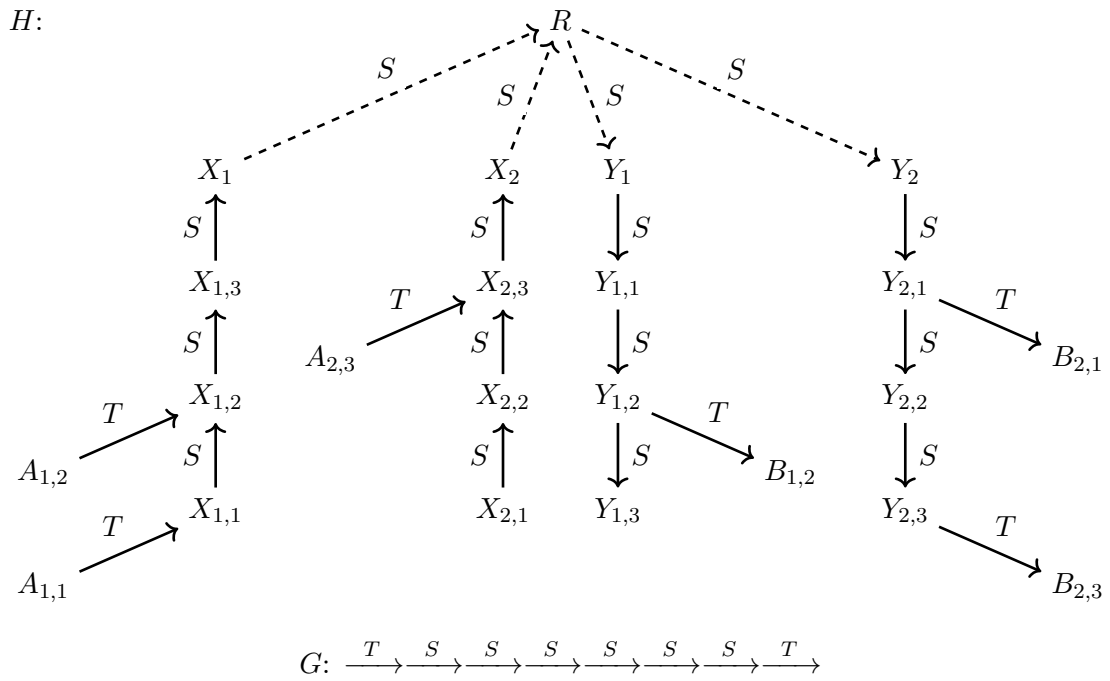


Figure 7: Illustration of the proof of Proposition 4.3 for the PP2DNF formula $X_1Y_2 \vee X_1Y_1 \vee X_2Y_2$. Dashed edges have probability $\frac{1}{2}$, all others have probability 1.

Hence, restricting instances to polytrees is not sufficient to ensure tractability, even for 1WP query graphs. We must thus restrict the instance further, by disallowing one of the two remaining features, namely branching and two-wayness. The first option of disallowing branching, i.e., requiring the instance to be a 2WP, is studied in Section 4.2 below, where we show that the problem is tractable for arbitrary query graphs.

The second option is to forbid two-wayness on the instance, i.e., restrict it to be a DWT. In this case, we first show that intractability holds even when we also forbid two-wayness in the query graph, i.e., we also restrict it to be a DWT. The result follows from our earlier work on the combined complexity of query evaluation [3, 2]:

Proposition 4.4 [2]. $\mathsf{PHom_L}(\mathsf{DWT}, \mathsf{DWT})$ *is #P-hard.*

If we forbid branching in the query graph instead of two-wayness, requiring it to be a 2WP, then intractability still holds, which also follows from our earlier results:

Proposition 4.5 [2]. $\mathsf{PHom_L}(\mathsf{2WP}, \mathsf{DWT})$ *is #P-hard.*

Thus, on DWT instances, the only remaining case is when the query is a one-way path. We will now show in the section below that this case is tractable, in addition to the case of arbitrary queries on 2WP instances that we left open above.

4.2. Tractability Results

The general proof technique to obtain PTIME combined complexity in this section is inspired by the probabilistic database literature [31]: compute the *lineage* of G on H as a Boolean formula in positive disjunctive normal form (DNF), then compute its probability. Let us first define *lineages*:

Definition 4.6. *Let G be a query graph and (H, π) be a probabilistic graph with edge set E. For any valuation $\nu : E \to \{0, 1\}$, we denote by $\nu(H)$ the possible world of H where each edge $e \in E$ is kept iff $\nu(e) = 1$. Letting φ be a Boolean function whose variables are the edges of E, we say that φ* captures the lineage of G on H *if, for any valuation $\nu : E \to \{0, 1\}$, the function φ evaluates to 1 under ν iff we have $G \rightsquigarrow \nu(H)$.*

Lineage representations allow us to reduce the PHom problem to the Boolean probability computation problem on the lineage function (recall Definition 4.2). Formally, for any query graph G and probabilistic graph (H, π), given a Boolean function φ that captures the lineage of G on H, we compute the answer to PHom on G and (H, π) as the probability $\Pr(\varphi, \pi)$ of φ under π: it is immediate by definition that these two quantities are equal.

Of course, computing a lineage representation does not generally suffice to show tractability, because, as we explained earlier, the Boolean probability computation problem is generally intractable. However, computing a Boolean lineage allows us to leverage the known tractable classes of Boolean formulas. Specifically, we will show how to use the class of *β-acyclic positive DNF formulas*, which are known to be tractable [10]. We define this notion, by first recalling the notion of a *β-acyclic hypergraph*, and then defining a *β-acyclic positive DNF*:

Definition 4.7. *A* hypergraph *$\mathcal{H} = (V, E)$ is a finite set V of vertices and a set E of non-empty subsets of V, called* hyperedges*. For $v \in V$, we write $\mathcal{H} \setminus v$ for the hypergraph $(V \setminus \{v\}, E')$ where E' is $\{e \setminus \{v\} \mid e \in E\} \setminus \{\emptyset\}$.*

A vertex $v \in V$ of $\mathcal{H}$ is called a β-leaf *[9] if the set of hyperedges that contain it, i.e., $\{e \in E \mid v \in e\}$, is totally ordered by inclusion. In other words, we can write $\{e \in E \mid v \in e\}$ as $(e_1, \ldots, e_k)$ in a way that ensures that $e_i \subseteq e_{i+1}$ for all $1 \leqslant i < k$.*

A β-elimination order for a hypergraph $\mathcal{H} = (V, E)$ is defined inductively as follows:

- *if $E = \emptyset$, then the empty tuple is a β-elimination order for $\mathcal{H}$;*
- *otherwise, a tuple $(v_1, \ldots, v_n)$ of vertices of $\mathcal{H}$ is a β-elimination order for $\mathcal{H}$ if v_1 is a β-leaf in $\mathcal{H}$ and $(v_2, \ldots, v_n)$ is a β-elimination order for $\mathcal{H} \setminus v_1$.*

The hypergraph $\mathcal{H}$ is β-acyclic *if there is a β-elimination order for $\mathcal{H}$.*

We can see a positive DNF (recall Definition 4.3) as a hypergraph of clauses on the variables, and introduce the notion of *β-acyclic positive DNFs* accordingly:

Definition 4.8. *The* hypergraph *$\mathcal{H}(\varphi)$ of a positive DNF on variables $\mathcal{X}$ has $\mathcal{X}$ as vertex set and has one hyperedge per clause, i.e., we have $\mathcal{H}(\varphi) := (\mathcal{X}, E)$ with $E := \{\{X_{i,j} \mid 1 \leqslant j \leqslant n_i\} \mid 1 \leqslant i \leqslant m\}$. We say that the positive DNF φ is* β-acyclic *if $\mathcal{H}(\varphi)$ is β-acyclic.*

It follows directly from results by Brault-Baron, Capelli, and Mengel about the β-acyclic $\#\mathsf{CSP}_\mathrm{d}$ problem [10] that we can tractably compute the probability of β-acyclic positive DNFs:

Theorem 4.9. *The Boolean probability computation problem is in PTIME when restricted to β-acyclic positive DNF formulas.*

Proof sketch. The $\#\mathsf{CSP}_\mathrm{d}$ problem studied in [10] is about computing a partition function over the hypergraph, under weighted constraints on hyperedges: it generalizes the problem of counting the number of valuations of β-acyclic formulae in conjunctive normal form (CNF) by [10, Lemma 3]. We show how the result extends to β-acyclic positive DNF, using de Morgan's law, and to probability computation for weighted variables, using additional constraints on singleton variable sets. □

We will then use the tractability of β-acyclic formulas to show PTIME combined complexity results for our PHom_L problem. The first result that we show is tractability for labeled 1WP query graphs on DWT probabilistic instance graphs:[2]

Proposition 4.10. $\mathsf{PHom}_\mathsf{L}(\mathsf{1WP}, \mathsf{DWT})$ *is PTIME.*

Proof sketch. Intuitively, the proof proceeds in three steps. The first step is to enumerate all candidate minimal matches of the query graph in the instance graph, i.e., subgraphs of the instance graph to which the query graph could have a homomorphism, and which are minimal for inclusion. As the query graph is a path, we know that the minimal matches are downward paths in the DWT instance: hence, as each vertex of the DWT instance can be the lowest vertex of at most one match, there are polynomially many matches to consider.

The second step is to decide which ones of these matches are actually a match of the query, by considering the labels: as both the query graph and the match are a 1WP, this is straightforward. These first two steps produce a positive DNF that captures the lineage of the query graph on the instance in the standard sense.

The third step is to notice that this lineage expression is β-acyclic: this is because its variables can be eliminated by considering the nodes of the instance DWT in a bottom-up fashion. □

Interestingly, we were not able to prove this result using tree automata-based dynamic programming approach (like we will do later for Proposition 5.4).

The second result that we show is tractability when restricting the instance to be a 2WP, and allowing arbitrary *connected* queries (remember from Proposition 3.3 that the problem is hard even on 1WP instances if we allow *disconnected* queries):

Proposition 4.11. $\mathsf{PHom}_\mathsf{L}(\mathsf{Connected}, \mathsf{2WP})$ *is PTIME.*

To show this result, we follow the same scheme as in the proof of Proposition 4.10 above: (i) enumerate all candidate matches; (ii) check whether they are indeed matches;

[2] The connection to β-acyclicity in this context is due to Florent Capelli.

and (iii) argue that the resulting lineage is β-acyclic. For the first step, there are polynomially many candidate matches to consider, because matches are necessarily connected subgraphs of the instance graph H, that are uniquely defined by their endpoints: this is where we use connectedness of the query. For the third step, the resulting lineage is β-acyclic for the same reason as in Proposition 4.10, as we can eliminate variables following the order of the path H: all connected subpaths containing an endpoint of the path are ordered by inclusion. What changes, however, is the second step: from the quadratically many possible matches, to compute the lineage expression, we must decide which ones are actually matches.

Deciding this for each subpath amounts to testing, given the connected query graph G and a candidate match H', whether $G \rightsquigarrow H'$, in the non-probabilistic sense. This graph homomorphism problem is generally intractable, but here the minimal match H' is a 2WP (as it is a subpath of H), so it turns out to enjoy combined tractability. The corresponding result was first shown by Gutjahr [24] for *unlabeled* graphs, when the instance graph is a path, or for more general instances satisfying a condition called the $\underline{X}$-property; this was generalized to labeled graphs by Gottlob, Koch, and Schulz in [22]. We recall here the definition of this property:

Definition 4.12 (Definition 3.2 of [22]). *Let $H = (V, E, \lambda)$ be a directed graph with labels on σ, let $R \in \sigma$, and let $<$ be a total order on V. Then R is said to have the $\underline{X}$-*property *w.r.t. $<$ if for all $n_0, n_1, n_2, n_3 \in V$ such that $n_0 < n_1$ and $n_2 < n_3$, if we have $n_0 \xrightarrow{R} n_3$ and $n_1 \xrightarrow{R} n_2$ then we also have $n_0 \xrightarrow{R} n_2$. H is said to have the $\underline{X}$-*property *w.r.t. $<$ if it is the case of each label R.*

Theorem 4.13. *(Theorem 3.5 of [22], extending Theorem 3.1 of [24]) Given a labeled query graph G, and given a labeled directed graph H with the $\underline{X}$-property w.r.t. some order $<$, we can determine in time $O(|H| \times |G|)$ whether $G \rightsquigarrow H$.*

We can use this result to check, for all connected subpaths of the 2WP instance graph, whether the query graph has a homomorphism to the subpath. This leads to the following sketch for the proof of Proposition 4.11 (the full proof is in Appendix):

Proof sketch. We proceed following the three-step process outlined above. We first enumerate the possible query matches in the instance, i.e., the quadratic number of connected subpaths. Second, we test for each subpath $a_i - \cdots - a_{i+k}$ whether it satisfies the query. We can do so tractably because the subpath clearly has the $\underline{X}$-property w.r.t. the order $a_i < \cdots < a_{i+k}$: using the notation of Definition 4.12, there are in fact no n_0, n_1, n_2, n_3 that satisfy the conditions. Third, having computed the resulting DNF, we compute its probability using β-acyclicity, eliminating variables in the order of the path as we explained above. □

5. Unlabeled Connected Queries

We now turn to the unlabeled setting, whose classification is presented in Table 3. We start with an intractability result which follows directly from the well-known intractability of query evaluation in probabilistic databases [31]:

Table 3: Tractability of $\mathsf{PHom}_{\not\mathsf{L}}$ in the connected case (Section 5)

$\downarrow G$ $H\rightarrow$	1WP	2WP	DWT	PT	Connected
1WP					5.1
2WP				5.6	
DWT				5.5	
PT					
Connected		4.11	3.6		

PTIME #P-hard Numbers given correspond to propositions for border cases, remaining cells can be filled using the inclusions from Figure 2.

Proposition 5.1 [31]. *The* $\mathsf{PHom}_{\not\mathsf{L}}(\mathsf{1WP}, \mathsf{Connected})$ *problem is #P-hard.*

Proof. Example 3.3 of [31] states that the conjunctive query $\exists x \exists y \exists z\, U(x,y) \wedge U(y,z)$ is #P-hard on TID instances. In other words, $\mathsf{PHom}_{\not\mathsf{L}}(\{\rightarrow\rightarrow\}, \mathsf{All})$ is #P-hard, which implies the #P-hardness of $\mathsf{PHom}_{\not\mathsf{L}}(\mathsf{1WP}, \mathsf{All})$. We conclude using Lemma 3.7, which provides a PTIME (Turing) reduction[3] from the $\mathsf{PHom}_{\not\mathsf{L}}(\mathsf{1WP}, \mathsf{Connected})$ problem to the $\mathsf{PHom}_{\not\mathsf{L}}(\mathsf{1WP}, \mathsf{All})$ problem. □

Note that this $\mathsf{PHom}_{\not\mathsf{L}}(\mathsf{1WP}, \mathsf{Connected})$ problem can be phrased in a very simple way: given an unlabeled connected probabilistic graph (H, π) and a length m as input (namely, that of the 1WP graph query), compute the probability that H contains a directed path of length m.

This result suggests that, to obtain tractability, we need to restrict the instance graphs. In fact, such tractability results were already obtained in the previous sections. In Section 3, we proved (Proposition 3.6) that $\mathsf{PHom}_{\not\mathsf{L}}(\mathsf{All}, \mathsf{DWT})$ has PTIME combined complexity. Similarly, in the previous section, we proved that $\mathsf{PHom}_{\mathsf{L}}(\mathsf{Connected}, \mathsf{2WP})$ is PTIME (Proposition 4.11), which means $\mathsf{PHom}_{\not\mathsf{L}}(\mathsf{Connected}, \mathsf{2WP})$ is also PTIME. This completes the analysis of the unlabeled case for 1WP, 2WP and DWT instances (see Table 3), so the only remaining case is that of PT instances.

We start our study of $\mathsf{PHom}_{\not\mathsf{L}}$ for PT instances with the simplest queries, namely, 1WP, for which we will show tractability. We will proceed by translating the 1WP query to a *bottom-up deterministic tree automata* [16]:

Definition 5.2. *Given an alphabet* Γ*, a* bottom-up deterministic tree automaton *on full binary (every node has either* 0 *or* 2 *children) rooted trees whose nodes are labeled by* Γ *is a tuple* $A = (Q, F, \iota, \Delta)$*, where:*

(i) Q *is a finite set of* states*;*

(i) $F \in Q$ *is a subset of* accepting states*;*

[3] Note that it is usual to define #P-hardness under Turing reductions rather than under Karp reductions, as #P is a counting complexity class.

(i) $\iota : \Gamma \to Q$ *is an* initialization function *determining the state of a leaf from its label;*

(i) $\Delta : \Gamma \times Q^2 \to Q$ *is a* transition function *determining the state of an internal node from its label and the states of its two children.*

Given a Γ*-tree* $\langle T, \lambda \rangle$ *(where* $\lambda : T \to \Gamma$ *is the* labeling function*), we define the* run *of* A *on* $\langle T, \lambda \rangle$ *as the function* $\varphi : T \to Q$ *such that (1)* $\varphi(l) = \iota(\lambda(l))$ *for every leaf* l *of* T*; and (2)* $\varphi(n) = \Delta(\lambda(n), \varphi(n_1), \varphi(n_2))$ *for every internal node* n *of* T *with children* n_1 *and* n_2*. The automaton* A accepts $\langle T, \lambda \rangle$ *if its run on* T *maps the root of* T *to a state of* F*.*

We will evaluate 1WP queries by translating them to a tree automaton and running it on an uncertain tree. This will use again the notion of *lineage* (recall Definition 4.6), which was extended in [4] to tree automata running on trees with uncertain Boolean labels: the *lineage* of an automaton on such a tree describes the set of annotations of the tree that makes the automaton accept. In this context, the lineage of *deterministic* tree automata was shown in [6] to be compilable to a *deterministic decomposable negation normal form* circuit [20]:

Definition 5.3. *A* deterministic decomposable negation normal form *(d-DNNF) is a Boolean circuit* C *with the following properties:*

(i) negations are only applied to input gates;

(ii) the inputs of any AND-gate depend on disjoint sets of input gates;

(iii) the inputs of any OR-gate are mutually exclusive, i.e., for any two input gates $g_1 \neq g_2$ *of* g*, there is no valuation of the inputs of* C *under which* g_1 *and* g_2 *both evaluate to true.*

We can then straightforwardly extend the Boolean probability computation problem (Definition 4.2) to take circuits as inputs, and the properties of d-DNNF circuits are designed to ensure that the Boolean probability computation problem restricted to d-DNNF has PTIME complexity [20]. Combining these tools, we can show that $\mathsf{PHom}_{\not\mathsf{L}}$ on one-way path queries and polytree instances is tractable:

Proposition 5.4. $\mathsf{PHom}_{\not\mathsf{L}}(\mathsf{1WP}, \mathsf{PT})$ *is PTIME.*

Proof sketch. The idea of the proof is to construct in polynomial time in the query graph G a bottom-up deterministic automaton A_G, which runs on binary trees T representing possible worlds of the polytree instance H, and accepts such a tree T iff the corresponding possible world satisfies G. We can then construct a d-DNNF representation of the lineage of G on H by [6, Theorem 6.11], which allows us to efficiently compute $\Pr(G \rightsquigarrow H)$: the complexity of this process is in $O(|A_G| \cdot |H|)$, hence polynomial in $|H| \cdot |G|$. (An alternative way to see this is to use the results of [15].)

Intuitively, the design of the bottom-up automaton ensures that, when it reaches a node n after having processed the subtree T_n rooted at n, its state reflects three linear-sized quantities about T_n:

1. the length of the longest path leading out of n;
2. the length of the longest path leading to n;
3. the length of the longest path overall in T_n (not necessarily via n).

The final states are those where the third quantity is greater than the length of G. The transitions compute each triple from the child triples by considering how the longest leading paths are extended, and how longer overall paths can be formed by joining an incoming and outgoing path. □

Hence, PT instances enjoy tractability for the simplest query graphs. We now study whether this result can be extended to more general queries. We first notice that this result immediately extends to branching (i.e., to DWT queries), and even to unions of DWT queries. Indeed, in the unlabeled setting, as we already observed at the end of Section 3.1, such queries are equivalent to 1WP queries:

Proposition 5.5. $\mathsf{PHom}_{\not\mathsf{L}}(\mathsf{DWT}, \mathsf{PT})$ *and* $\mathsf{PHom}_{\not\mathsf{L}}(\bigsqcup\mathsf{DWT}, \mathsf{PT})$ *are PTIME.*

Proof. We first show the result for a DWT query graph G. Let m be its height, i.e., the length of the longest directed path it contains, and let G' be the 1WP of length m, which can be computed in PTIME from G. It is easy to observe that G and G' are equivalent. Indeed, we can find G' as a subgraph of G by taking any directed path of maximal length, and conversely there is a homomorphism from G to G': map the root of G to the first vertex of G' and each element of G' at distance i from the root to the i-th element of G'. Hence, $\mathsf{PHom}_{\not\mathsf{L}}$ on G and an input probabilistic PT instance reduces to $\mathsf{PHom}_{\not\mathsf{L}}$ on G' and the same instance, so that the result follows from Proposition 5.4.

The same argument extends to $\bigsqcup$DWT by considering the greatest height of a connected component of G. □

Thus, we have successfully extended from 1WP to $\bigsqcup$DWT queries while preserving tractability on PT instances. However, as we now show, tractability is not preserved if we extend queries to allow two-wayness. Indeed:

Proposition 5.6. $\mathsf{PHom}_{\not\mathsf{L}}(\mathsf{2WP}, \mathsf{PT})$ *is #P-hard.*

Proof. We adapt the proof of Proposition 4.1, but we face the additional difficulty of not being allowed to use labels. Fortunately, we can use the two-wayness in the query graph to simulate labels.

We reduce from #PP2DNF (recall Definition 4.3): the input consists of two disjoint sets $\mathcal{X} = \{X_1, \ldots, X_{n_1}\}$, $\mathcal{Y} = \{Y_1, \ldots, Y_{n_2}\}$ of Boolean variables, and a PP2DNF formula φ. We construct a 2WP query graph G' and PT instance H' with the same construction as the one that yielded H and G in that proof, except that we perform the following replacements (see Figure 8):

- replace every edge $a \xrightarrow{S} b$ of H and G by 3 edges $a \rightarrow\rightarrow\leftarrow b$;

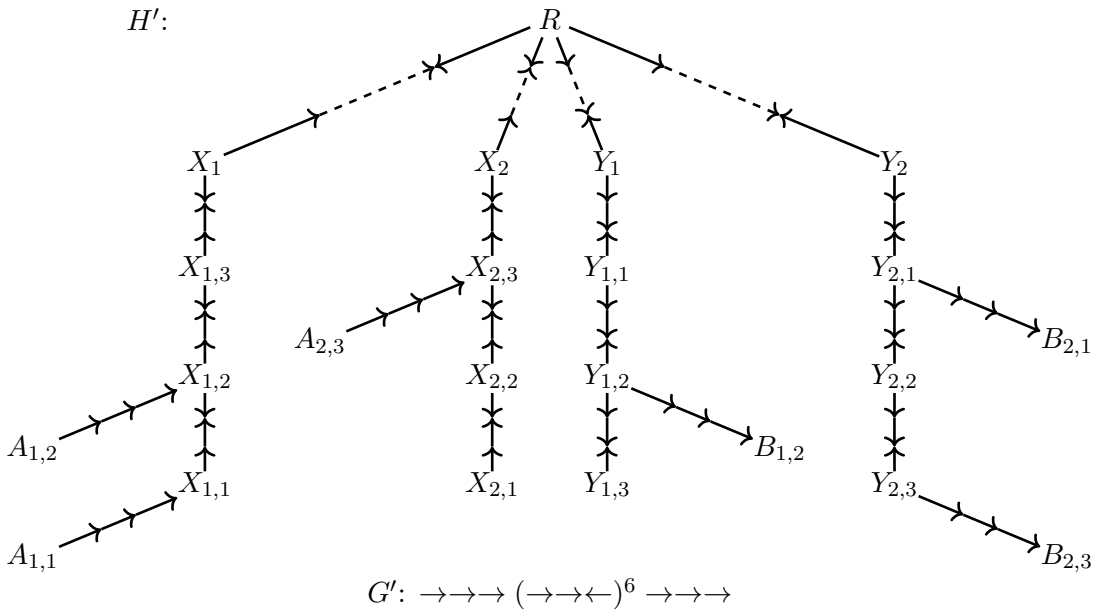


Figure 8: Illustration of the proof of Proposition 5.6 for the PP2DNF formula $X_1Y_2 \vee X_1Y_1 \vee X_2Y_2$. Dashed edges have probability $\frac{1}{2}$, all others have probability 1.

- replace every edge $a \xrightarrow{T} b$ of H and G by 3 edges $a \rightarrow\rightarrow\rightarrow b$.

In particular, the query graph is then defined as follows:

$$G' := \rightarrow\rightarrow\rightarrow\ (\rightarrow\rightarrow\leftarrow)^{m+3}\ \rightarrow\rightarrow\rightarrow\ .$$

All the edges of H' have probability 1, except the middle edge of the edges that replaced the S-labeled edges used to code the valuation of the variables (e.g., for X_i, the middle edge of the 3 edges $X_i \rightarrow\rightarrow\leftarrow R$), which have probability $\frac{1}{2}$.

One can check that any image of G' must again go from the vertex $A_{x_j,j}$ to the vertex $B_{y_j,j}$ for some $1 \leqslant j \leqslant m$. The key insight is that the first $\rightarrow^5$ of G must be matched to a $\rightarrow^5$-path in H', which only exist as the concatenation of a $\rightarrow^3$ obtained by rewriting a T-edge for some variable X_j, and of the first $\rightarrow^2$ of the (undirected) path from $X_{x_j,j}$ to R. Then there is no choice left to match the next edges without failing.

From this we deduce that, from any possible world H'_{W} of the modified instance H', considering the corresponding possible world H_{W} of the unmodified instance H following the natural bijection, the modified query graph G' has a homomorphism to H'_{W} iff the unmodified query graph G has a homomorphism to H_{W}. We thus conclude that the probabilistic homomorphism problem on G' and H' has the same answer as the one on G and H, which finishes the proof. □

6. Conclusion

We have introduced the probabilistic homomorphism problem, also known in the database community as probabilistic evaluation of conjunctive queries on TID instances, and studied its combined complexity for various restricted classes of query and instance graphs. Our classes illustrate the impact on PHom of various features: acyclicity, two-wayness, branching, connectedness, and labeling. As we show, the landscape is already quite enigmatic, even for those seemingly restricted classes! In particular, we have identified four incomparable maximal tractable cases, reflecting various tradeoffs between the expressiveness that we can allow in the queries and in the instances:

- arbitrary queries on unlabeled downward trees (Proposition 3.6);
- one-way path queries on labeled downward trees (Proposition 4.10);
- connected queries on two-way labeled path instances (Proposition 4.11);
- downward tree queries on unlabeled polytrees (Proposition 5.5);

These results all extend to disconnected instances, as shown in Section 3.3. The (somewhat sinuous) tractability border is described in Tables 1, 2, and 3.

It is debatable whether the tractable classes we have identified yield interesting tractable cases for practical applications. The settings of Propositions 3.6, 5.5, and 4.11 may look restrictive, as both labels and branching are important features of real-world instances, though some situations may involve unlabeled tree-like instances, or labeled words. The setting of Proposition 4.10 may be richer, and is reminiscent of probabilistic XML [26]: the instance is a labeled (downward) tree, while the query is a path evaluated on that tree.

Future work. The query and instance features studied in this paper could be completed by other dimensions: e.g., studying an *unweighted* case inspired by counting CSP where all probabilities are 1/2 (as our hardness proofs seem to heavily rely on some edges being certain); imposing *symmetry* in the sense of [7]; or alternatively restricting the *degree* of graphs (though all our hardness proofs on polytrees and lower classes can seemingly be modified to work on bounded-degree). Another option would be to modify some of the existing dimensions: first, polytrees could be generalized to *bounded-treewidth instances*, as we believe that the relevant tractability result (Proposition 5.5) adapts to this setting; second, non-branching instances could be generalized to *bounded-pathwidth instances*, or maybe general instances with the $\underline{X}$-property (recall Definition 4.12).

Of course, another natural direction would be to lift the arity-two restriction, although it is not immediate to generalize the definition of our classes to work in higher arity signatures. We could also extend the query language: in particular, allow *unions* of conjunctive queries as in [19]; allow a *descendant* axis in the spirit of XML query languages [8]; or more generally allow fixpoint constructs as done in [3] in the non-probabilistic case. An interesting question is whether an extended query language could capture the tractability results obtained in the context of probabilistic XML by [15] (remembering, however,

that such results crucially depend on having an order relation on node children [1]). Another possibility would be to search for extensions of the β-acyclicity approach, and investigate which restrictions on the queries and instances ensure that the lineages are β-acyclic.

Last, the connection to CSP would seem to warrant further investigation. In particular, we do not know whether one could show a general dichotomy result on the combined complexity of query evaluation on TID instances, to provide a probabilistic analogue to the Feder–Vardi conjecture [21].

Acknowledgements. We are grateful to Florent Capelli for pointing out to us the connection to β-acyclic instances and suggesting the idea used in the proof of Proposition 4.10, and to Tyson Williams for pointing out a connection to holographic algorithms, in a very informative CSTheory post [34]. We also thank anonymous referees for their valuable feedback. This work was partly funded by the Télécom ParisTech Research Chair on Big Data and Market Insights.

References

[1] A. Amarilli. The possibility problem for probabilistic XML. In *AMW*, 2014.

[2] A. Amarilli, P. Bourhis, M. Monet, and P. Senellart. Combined tractability of query evaluation via tree automata and cycluits (extended version). *CoRR*, abs/1612.04203, 2016. `https://arxiv.org/abs/1612.04203`. Extended version of [3].

[3] A. Amarilli, P. Bourhis, M. Monet, and P. Senellart. Combined tractability of query evaluation via tree automata and cycluits. In *ICDT*, 2017.

[4] A. Amarilli, P. Bourhis, and P. Senellart. Provenance circuits for trees and treelike instances. In *ICALP*, volume 9135 of *LNCS*, 2015.

[5] A. Amarilli, P. Bourhis, and P. Senellart. Provenance circuits for trees and treelike instances (extended version). *CoRR*, abs/1511.08723, 2015. `https://arxiv.org/abs/1511.08723`. Extended version of [4].

[6] A. Amarilli, P. Bourhis, and P. Senellart. Tractable lineages on treelike instances: limits and extensions. In *PODS*, 2016.

[7] P. Beame, G. Van den Broeck, E. Gribkoff, and D. Suciu. Symmetric weighted first-order model counting. In *PODS*, 2015.

[8] M. Benedikt and C. Koch. XPath leashed. *CSUR*, 2009.

[9] J. Brault-Baron. Hypergraph acyclicity revisited. *ArXiv e-prints*, abs/1403.7076, 2014.

[10] J. Brault-Baron, F. Capelli, and S. Mengel. Understanding model counting for β-acyclic CNF-formulas. In *STACS*, 2015.

[11] A. A. Bulatov. The complexity of the counting constraint satisfaction problem. *J. ACM*, 60(5), 2013.

[12] J. Cai, P. Lu, and M. Xia. Holographic reduction, interpolation and hardness. *Computational Complexity*, 21(4), 2008. Preprint of [13].

[13] J. Cai, P. Lu, and M. Xia. Holographic reduction, interpolation and hardness. *Computational Complexity*, 21(4), 2012.

[14] I. I. Ceylan, A. Darwiche, and G. Van den Broeck. Open-world probabilistic databases. In *KR*, 2016.

[15] S. Cohen, B. Kimelfeld, and Y. Sagiv. Running tree automata on probabilistic XML. In *PODS*, 2009.

[16] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata: Techniques and applications, 2007. Available from `http://www.grappa.univ-lille3.fr/tata`.

[17] R. Curticapean and D. Marx. Complexity of counting subgraphs: Only the boundedness of the vertex-cover number counts. In *FOCS*, 2014.

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

[19] N. Dalvi and D. Suciu. The dichotomy of probabilistic inference for unions of conjunctive queries. *JACM*, 59(6), 2012.

[20] A. Darwiche. On the tractable counting of theory models and its application to truth maintenance and belief revision. *J. Applied Non-Classical Logics*, 11(1-2), 2001.

[21] T. Feder and M. Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through Datalog and group theory. *SIAM Journal on Computing*, 1998.

[22] G. Gottlob, C. Koch, and K. U. Schulz. Conjunctive queries over trees. *JACM*, 53(2), 2006.

[23] M. Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *JACM*, 2007.

[24] W. Gutjahr, E. Welzl, and G. Woeginger. Polynomial graph-colorings. *Discrete Applied Mathematics*, 35(1), 1992.

[25] S. Khanna, S. Roy, and V. Tannen. Queries with difference on probabilistic databases. *PVLDB*, 2011.

[26] B. Kimelfeld and P. Senellart. Probabilistic XML: models and complexity. In Z. Ma and L. Yan, editors, *Advances in Probabilistic Databases for Uncertain Information Management*. Springer, 2013.

[27] S. Odagiri and H. Goto. On the greatest number of paths and maximal paths for a class of directed acyclic graphs. *IEICE Trans. Fundamentals*, E97-A(6), 2014.

[28] J. S. Provan and M. O. Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM J. Computing*, 12(4), 1983.

[29] B. Schröder. *Ordered Sets*. Birkhäuser, second edition, 2016.

[30] R. Stanley. *Enumerative Combinatorics*, volume 1. Cambridge University Press, 1997.

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

[32] L. G. Valiant. Holographic algorithms. *SIAM J. Comput.*, 37(5), 2008.

[33] M. Y. Vardi. On the complexity of bounded-variable queries. In *PODS*, 1995.

[34] T. Williams. Complexity of counting the number of edge covers of a graph. `http://cstheory.stackexchange.com/q/21309`, 2014.

[35] T. Williams. *Advances in the Computational Complexity of Holant Problems*. PhD thesis, PhD thesis, University of Wisconsin-Madison, 2015.

[36] M. Xia, P. Zhang, and W. Zhao. Computational complexity of counting problems on 3-regular planar graphs. *Theoretical Computer Science*, 384(1):111–125, 2007.

[37] M. Yannakakis. Algorithms for acyclic database schemes. In *VLDB*, 1981.

A. Proofs for Section 3 (Disconnected Case)

Proposition 3.6. $\mathsf{PHom}_{\not\mathsf{L}}(\mathsf{All}, \bigsqcup\mathsf{DWT})$ *is PTIME.*

Proof. Let G be an arbitrary unlabeled graph and (H, π) a probabilistic graph with $H \in \bigsqcup\mathsf{DWT}$. We observe that if G contains a directed cycle, then it cannot have a homomorphism to a subgraph of H (which is necessarily acyclic), so $\Pr(G \rightsquigarrow H) = 0$. Hence, it suffices to study the case where the query graph G is a DAG.

Likewise, if there are two vertices u, v of G and directed paths χ, χ' in G from u to v such that χ and χ' have different lengths, then again G cannot have a homomorphism to a subgraph of H: indeed, any subgraph of H is a directed forest and there is at most one directed path between each pair of nodes. So we can assume without loss of generality that there is no such pattern in G, and G is therefore graded.

Letting μ be a level mapping of G, we call the *difference of levels* of μ the difference between the largest and smallest value of its image; the *difference of levels* of G itself is

the minimum difference of levels of a level mapping of G. As the level mappings of G only differ in the constant value that they add to all vertices of each connected component, the difference of levels can clearly be computed in PTIME by shifting each connected component so that its minimal level is zero, and computing the difference; we call the result of the shifting the *minimal level mapping* of G.

Letting m be the difference of levels of G, we now make the following claim: *in any subgraph H' of H, there is a homomorphism from G to H' if and only if H' has a directed path of length m.*

This claim implies the result. Indeed, we can first check in PTIME if G has no cycles and has no pairs of paths of different lengths between two endpoints, and return 0 if the conditions are violated. We can then compute in PTIME the difference of levels m of G using the observations above. Now, on any subgraph of H, the query G is equivalent to the 1WP graph $\rightarrow^m$, so our result follows from Proposition 5.5.

All that remains is to prove the claim. We first note that it suffices to show the claim under the assumption that G is connected. Indeed, if the claim is true for all connected G, then the claim is implied for arbitrary G by considering each of its connected components, applying the claim, and observing that G has a suitable homomorphism to H' iff each one of its connected components does, i.e., iff H' has a directed path whose length is the maximal difference of levels of a connected component of G, and this is precisely the difference of levels m of G. Hence, we now prove the claim for connected G.

We start with the backwards direction of the claim. It is easily seen that there is a homomorphism h' from G to the 1WP graph $\rightarrow^m$. Indeed, we define h' according to the minimal level mapping μ of G: we set h' to map all the vertices whose level is i to the i-th vertex of $\rightarrow^m$. From the existence of h', we know that, whenever there is a homomorphism h from $\rightarrow^m$ to H', then $h \circ h'$ is a homomorphism from G to H', which shows the backwards implication.

For the forward direction of the claim, suppose that there exists a homomorphism h from G to H', and let m be the difference of levels of G. Because G is connected and H' is in $\bigsqcup$DWT, the image of h is actually a DWT, call it T. Now it is easy to see that the image of a node that has level $m - i$ in G has depth i in T, so that T (and so H') contains the 1WP $\rightarrow^m$. This finishes the proof of the converse and thus the proof of Proposition 3.6. ☐

B. Proofs for Section 4 (Labeled Connected Queries)

Proposition 4.1. $\mathsf{PHom_L}(\mathsf{1WP}, \mathsf{PT})$ *is #P-hard.*

Proof. We reduce from the #P-hard problem `#PP2DNF`. From the formula φ, we construct the following $\{S, T\}$-labeled probabilistic graph H (an example of this construction is presented in Figure 7):

- The vertices of H are $\{R\} \sqcup \{X_1, \dots X_{n_1}\} \sqcup \{Y_1, \dots, Y_{n_2}\} \sqcup \{X_{i,j} \mid 1 \leqslant i \leqslant n_1,\ 1 \leqslant j \leqslant m\} \sqcup \{Y_{i,j} \mid 1 \leqslant i \leqslant n_2,\ 1 \leqslant j \leqslant m\} \sqcup \{A_{x_j,j} \mid 1 \leqslant j \leqslant m\} \sqcup \{B_{y_j,j} \mid 1 \leqslant j \leqslant m\}$.

- The edges of H, all of which have probability 1 except when specified, are:
  - $X_i \xrightarrow{S} R$ for all $1 \leqslant i \leqslant n_1$ and $R \xrightarrow{S} Y_i$ for all $1 \leqslant i \leqslant n_2$, all having probability $\frac{1}{2}$ and intuitively coding the valuation of each variable;
  - For all $1 \leqslant i \leqslant n_1$, the edge $X_{i,m} \xrightarrow{S} X_i$ and the edges $X_{i,j} \xrightarrow{S} X_{i,j+1}$ for all $1 \leqslant j \leqslant m-1$;
  - For all $1 \leqslant i \leqslant n_2$, the edge $Y_i \xrightarrow{S} Y_{i,1}$ and the edges $Y_{i,j} \xrightarrow{S} Y_{i,j+1}$ for all $1 \leqslant j \leqslant m-1$;
  - For all $1 \leqslant j \leqslant m$, the edges $A_{x_j,j} \xrightarrow{T} X_{x_j,j}$ and $Y_{y_j,j} \xrightarrow{T} B_{y_j,j}$, intuitively indicating that variables X_{x_j} and Y_{y_j} belong to clause j.

The $\{S,T\}$-labeled graph G is then $\xrightarrow{T} (\xrightarrow{S})^{m+3} \xrightarrow{T}$. It is clear that G is a 1WP query graph, H is a polytree and that both can be constructed in PTIME from φ. We now show that $\Pr(G \rightsquigarrow H)$ is exactly the number of satisfying assignments of φ divided by 2^n, so that the computation of one reduces in PTIME to the computation of the other, concluding the proof. To see why, we define a bijection between the valuations ν of $\{X_1, \ldots, X_{n_1}\} \sqcup \{Y_1, \ldots, Y_{n_2}\}$ to the possible worlds H' of H that have non-zero probability, in the expected way: keep the edge $X_i \xrightarrow{S} R$ (resp., $R \xrightarrow{S} Y_i$) iff X_i (resp., Y_i) is assigned to true in the valuation. We then show that there is a homomorphism from G to H' if and only if φ evaluates to true under ν.

Indeed, if there is a homomorphism from G to H', then by considering the only possible matches of the T-edges, one can check easily that the image of the match in H' must be of the following form for some $1 \leqslant j \leqslant m$: $A_{x_j,j} \xrightarrow{T} X_{x_j,j} \xrightarrow{S} X_{x_j,j+1} \xrightarrow{S} \cdots \xrightarrow{S} X_{x_j,m} \xrightarrow{S} X_{x_j} \xrightarrow{S} R \xrightarrow{S} Y_{y_{j'}} \xrightarrow{S} Y_{y_{j'},1} \xrightarrow{S} Y_{y_{j'},2} \xrightarrow{S} \cdots \xrightarrow{S} Y_{y_{j'},j'} \xrightarrow{T} B_{y_{j'},j'}$; further, from the length of the S-path we must have $(m-j)+4+(j'-1) = m+3$, so that we must have $j = j'$. Then, by construction, X_{x_j} and Y_{y_j} belong to clause j, so the valuation satisfies φ. Conversely, suppose that the valuation satisfies φ, then for some $1 \leqslant j \leqslant m$ we know that X_{x_j} and Y_{y_j} are assigned to true by the valuation, and so we can build the homomorphism as above from G to H'. □

Proposition 4.4 [2]. $\mathsf{PHom_L}(\mathsf{DWT}, \mathsf{DWT})$ *is #P-hard.*

Proof. The proof is almost the same as that of Proposition 36 of [2], straightforwardly adapted to our setting of probabilistic graphs (in particular replacing the unary relation R by a binary relation), by observing that the probabilistic instance defined in this proof is actually a DWT (beyond having treewidth 1), and that the query actually corresponds to a DWT graph (beyond being α-acyclic). □

Proposition 4.5 [2]. $\mathsf{PHom_L}(\mathsf{2WP}, \mathsf{DWT})$ *is #P-hard.*

Proof. The proof is almost the same as that of Proposition 38 of [2], again adapted to our setting of probabilistic graphs, with one small modification: we do not materialize edges $b \xrightarrow{S_-} a$ in the instance graph for each edge $a \xrightarrow{S} b$ in the instance, and instead

modify the query to replace all edges $x \xrightarrow{S_-} y$ by edges $x \xleftarrow{S} y$. This ensures that the query is a 2WP and the instance is a DWT, and hardness is shown similarly to the original proof. □

Theorem 4.9. *The Boolean probability computation problem is in PTIME when restricted to β-acyclic positive DNF formulas.*

Proof. We reduce our Boolean probability computation problem to the problem of β-acyclic $\#\mathtt{CSP}_\mathrm{d}$ of [10], which they show to be in PTIME (Theorem 26 of [10]). We will explain how probability computation in the sense of Definition 4.2 can be encoded in their setting, by a variant of their own encoding (in Lemma 3 of [10]): we give a full proof for completeness.

First, we recall their definition of $\#\mathtt{CSP}_\mathrm{d}$ (Definitions 1 and 2 in [10]) in the case of a Boolean domain. We denote by $\mathbb{Q}_+$ the nonnegative rational numbers. Denote by $\{0,1\}^{\mathcal{X}}$ the set of functions from $\mathcal{X}$ to $\{0,1\}$, i.e., the Boolean valuations of $\mathcal{X}$. For $\nu \in \{0,1\}^{\mathcal{X}}$ and $\mathcal{Y} \subseteq \mathcal{X}$, we denote by $\nu_{|\mathcal{Y}}$ the restriction of ν to $\mathcal{Y}$. A *weighted constraint (with default value)* on variables $\mathcal{X}$ is a pair $c = (f, \mu)$ that consists of a function $f : S \to \mathbb{Q}_+$ for some subset S of $\{0,1\}^{\mathcal{X}}$, called the *support* of c, and a *default value* $\mu \in \mathbb{Q}_+$; we write $\mathrm{var}(c) := \mathcal{X}$. The constraint c induces a total function on $\{0,1\}^{\mathcal{X}}$, also denoted c, that maps $\nu \in \{0,1\}^{\mathcal{X}}$ to $f(\nu)$ if $\nu \in S$, and to μ otherwise. The *size* of c is $|c| = |S| \times |\mathcal{X}|$. Intuitively, a constraint with default value assigns a weight in $\mathbb{Q}_+$ to all valuations of $\mathcal{X}$, but the default value mechanism allows us to avoid writing explicitly the complete table of this mapping.

An instance of the $\#\mathtt{CSP}_\mathrm{d}$ problem then consists of a finite set I of weighted constraints. The size of I is $|I| := \sum_{c \in I} |c|$, and we write $\mathrm{var}(I) := \bigcup_{c \in I} \mathrm{var}(c)$. The output of the problem is the *partition function*

$$w(I) = \sum_{\nu \in \{0,1\}^{\mathrm{var}(I)}} \prod_{c \in I} c(\nu_{|\mathrm{var}(c)}).$$

The *hypergraph* $\mathcal{H}(I)$ of the $\#\mathtt{CSP}_\mathrm{d}$ instance I (defined in Section 2.2 of [10]) is the hypergraph $(\mathrm{var}(I), E_I)$ where $E_I = \{\mathrm{var}(c) \mid c \in I\}$. We say that I is *β-acyclic* if $\mathcal{H}(I)$ is a β-acyclic hypergraph (recall Definition 4.7), and we call *β-acyclic* $\#\mathtt{CSP}_\mathrm{d}$ the problem $\#\mathtt{CSP}_\mathrm{d}$ restricted to β-acyclic instances. By Theorem 26 of [10], the problem β-acyclic $\#\mathtt{CSP}_\mathrm{d}$ is in PTIME.

We now explain how to reduce the probability computation problem to the β-acyclic $\#\mathtt{CSP}_\mathrm{d}$ problem. Let $\varphi = \bigvee_{1 \leqslant i \leqslant m} \left(\bigwedge_{1 \leqslant j \leqslant n_i} X_{i,j} \right)$ be a Boolean β-acyclic DNF on variables $\mathcal{X}$, with probabilities $\pi(X) \in [0,1]$ for each $X \in \mathcal{X}$. We construct in linear time from φ and π the variable set $\mathcal{X}' := \{X' \mid X \in \mathcal{X}\}$, the CNF formula $\varphi' := \bigwedge_{1 \leqslant i \leqslant m} \left(\bigvee_{1 \leqslant j \leqslant n_i} X'_{i,j} \right)$, and the probability valuation π' on $\mathcal{X}'$ defined by $\pi'(X'_{i,j}) = 1 - \pi(X_{i,j})$. By De Morgan's duality law, φ' is equivalent to the negation of φ, so that we have $\Pr(\varphi, \pi) = 1 - \Pr(\varphi', \pi')$; hence, the probability computation problem for φ and π reduces in PTIME to the same problem for φ' and π'.

We then construct in linear time a β-acyclic $\#\mathtt{CSP}_\mathrm{d}$ instance I such that $\Pr(\varphi', \pi') = w(I)$, which concludes the proof. For each variable $X' \in \mathcal{X}'$, we define a weighted constraint $c_{X'}$ on variables $\{X'\}$ by $c_{X'}(X' \mapsto 1) = \pi'(X')$ and $c_{X'}(X' \mapsto 0) = 1 - \pi'(X')$, which codes the probability of the variables. Now, for each clause $1 \leqslant i \leqslant m$, just like in Lemma 3 of [10], we define a weighted constraint $c_i = (f_i, 1)$ with default value 1 whose variables are $\{X'_{i,j} \mid 1 \leqslant j \leqslant n_i\}$, i.e., those that occur in the clause: $f_i(\nu)$ is 0 for the (unique) valuation that sets all variables of the clause to 0, intuitively coding the constraint of the clause. From the fact that φ was β-acyclic, it is clear that I is also β-acyclic. Now, the result $w(I)$ of the partition function sums over all valuations of the variables of I, namely the variables $\mathcal{X}'$ of φ'. Whenever a valuation does not satisfy some clause $1 \leqslant i \leqslant m$, the weighted constraint c_i will give it weight 0, hence ensuring that the product evaluates to 0, so we can restrict the sum to valuations that satisfy φ': such valuations are given weight 1 by all weighted constraints c_i. Now, it is easy to see that the weight of valuations ν that satisfy φ is their probability $\pi'(\nu)$, as each $c_{X'}$ gives them weight $\pi'(X')$ or $1 - (\pi'(X'))$ depending on whether $\nu(X')$ is 1 or 0. Hence, we have reduced the probability computation problem for β-acyclic DNF formulas to β-acyclic $\#\mathtt{CSP}_\mathrm{d}$ in PTIME, which concludes the proof. □

Proposition 4.10. $\mathsf{PHom_L}(\mathsf{1WP}, \mathsf{DWT})$ *is PTIME.*

Proof. Let $G := u_1 \xrightarrow{R_1} \cdots \xrightarrow{R_{m-1}} u_m$ be the 1WP query (where all R_i are not necessarily distinct), and H be the downwards tree instance. The idea is to construct the lineage of G on H as a β-acyclic DNF φ, so that we can conclude with Theorem 4.9. It is clear that any match of G can only be a downwards path of H, hence we construct φ as follows: for every downwards path $a_1 \xrightarrow{R'_1} \cdots \xrightarrow{R'_{m-1}} a_m$ of length m of H (their number is linear in $|H|$ because each path is uniquely defined by the choice of a_m) check if the path is a match of G (i.e, check that $R_i = R'_i$ for $1 \leqslant i \leqslant m-1$), and if it is the case then create a new clause of φ whose variables are all the facts $a_i \xrightarrow{R_i} a_{i+1}$ for $1 \leqslant i \leqslant m-1$.

The formula φ thus obtained is then a DNF representation of the lineage of H on G, and has been built in time $O(|H| \cdot |G|)$, i.e., in PTIME. We now justify that φ is β-acyclic by giving a β-elimination order for φ: while H still has edges, repeatedly pick a leaf b of H and, letting a be the parent of b, eliminate the variable $a \xrightarrow{R} b$ from φ. Such a variable will always be a β-leaf, as any set of downwards paths of a downwards tree all ending at a leaf is necessarily ordered by inclusion. From the above, the fact that φ is β-acyclic suffices to conclude the proof. □

Proposition 4.11. $\mathsf{PHom_L}(\mathsf{Connected}, \mathsf{2WP})$ *is PTIME.*

Proof. First of all, notice that, as the query graph G is connected, the image of a homomorphism from the query G to the 2WP instance H is necessarily a connected component of H. Moreover, each connected component of H is also a 2WP and there are $O(|H|^2)$ of them. We then proceed as follows. For every connected subpath $C = a_1 - \cdots - a_n$ (with each $-$ being either $\xrightarrow{R}$ or $\xleftarrow{R}$ for some binary relation R in H) of H, we check if there is a homomorphism from G to C. This can be done in PTIME by Theorem 4.13,

because C trivially has the $\underline{X}$-property w.r.t. the total order $a_1 < a_2 < \cdots < a_n$: the only possibility for (n_0, n_3) and (n_1, n_2) to be edges of C when n_0 comes before n_1 and n_2 comes before n_3 is if $n_0 = n_2$ and $n_1 = n_3$, in which case it cannot hold that $n_0 \xrightarrow{R} n_3$ and $n_1 \xrightarrow{R} n_2$ at the same time, because we disallow multi-edges. If there is such a homomorphism, then we create a new clause of φ whose variables are all the facts that belong to C.

From this, we obtain in PTIME a positive DNF φ that captures the lineage of G on H. We now justify that φ is β-acyclic by giving a β-elimination order for φ, by an argument similar to the proof of Proposition 4.10: repeatedly eliminate a variable $a - b$ from φ and this fact from H, where b is an endpoint of H. Indeed such a variable will always be a β-leaf, as any set of connected component of H including $a - b$ is necessarily ordered by inclusion. Hence, φ is β-acyclic, which allows us to conclude. □

C. Proofs for Section 5 (Unlabeled Connected Queries)

Proposition 5.4. $\mathsf{PHom}_{\not\mathsf{L}}(\mathsf{1WP}, \mathsf{PT})$ *is PTIME.*

Proof. Let G, (H, π) be the 1WP query graph and the probabilistic PT instance, and m be the length of G. Then $\Pr(G \rightsquigarrow H)$ is the probability that H contains a directed path of length at least m.

Because we will use automata that run on full binary trees, we will have to represent possible worlds of H as full binary trees. The first step is to transform H in linear time into a full binary polytree H' by applying a variant of the left-child-right-sibling encoding: in so doing, in addition to unlabeled edges of both orientations that exist in the polytree, we will also introduce some edges called *ε-edges* that are labeled by ε and whose orientation does not matter (so we see them as undirected edges and write them $a - b$); intuitively, the ε-edge $a - b$ means that a and b are in fact the same. For a node $a \in H$ and a child b of a, we say that b is an *up-child* of a if we have $b \to a$ and a *down-child* of a if we have $a \to b$. We do this transformation by processing H bottom-up as follows:

- If n is a leaf node of H, then create a node n' in H'.
- If n is an internal node of H with up-children $u_1, \ldots, u_k$ and down-children $d_1, \ldots, d_l$ then, letting $u'_1, \ldots, u'_k$ and $d'_1, \ldots, d'_l$ be the corresponding nodes in H': create a node n' in H' and nodes $n'_1, \ldots, n'_{k+l-2}$ with the following ε-edges: $n' - n'_1 - \ldots - n'_{k+l-2}$, all having probability 1. Create an edge $u'_1 \to n'$ whose probability is that of $u_1 \to n$. For $2 \leqslant i \leqslant k$ create an edge $u'_i \to n'_{i-1}$ annotated with the same probability as $u_i \to n$. For $1 \leqslant i \leqslant l - 1$ create an edge $n'_{k-1+i} \to d'_i$ annotated with the same probability as $n \to d'_i$, and finally create an edge $n'_{k-2+l} \to d'_l$ annotated with the same probability as $n \to d'_l$. Last, if any node has exactly one children (specifically, n', in case $k + l = 1$), then create a node n'' in H' and connect it with an ε-edge to the node.

One can check that H' is indeed a full binary polytree (with some edges being labeled by ε and being undirected) and that $\Pr(G \rightsquigarrow H)$ equals the probability that H' contains a path of the form $(\rightarrow -^*)^m$, that is, m occurrences of a directed edge $\rightarrow$ followed by some sequence of ε-edges $-$.

The second step is to transform in linear time H' into a probabilistic tree T to which we can apply the construction of [5]. Specifically, T must be an ordered full binary rooted tree whose edges do not have a label or an orientation, but whose nodes n carry a label in some finite alphabet Γ (written $\lambda(n)$, where λ is the labeling function) and with a probability value written $\pi(n)$. Writing $\overline{\Gamma} := \Gamma \times \{0,1\}$ as in [5], the semantics of T is that it stands for a probability distribution on $\bar{\Gamma}$-trees, i.e., trees T' labeled with $\Gamma \times \{0,1\}$, which have same skeleton as T: for each node n of T, the corresponding node n' in a possible world T' has label $(\lambda(n), 1)$ with probability $\pi(n)$ and label $(\lambda(n), 0)$ otherwise. We do this transformation by first adding a new root vertex to H' with an ε-edge with probability 1 to the original root (this clearly does not change the probability that H' has a path of the prescribed form), and then simply create T from H' by assigning the label and probability of each node that is not the new root as the direction of its parent edge (in $\Gamma := \{\uparrow, \downarrow, -\}$) and its probability (so the root of T' has label $-$ and probability 1).

Our last step is to construct a bDTA A_G running on $\overline{\Gamma}$-trees such that for every possible world W of H', letting T_W be its representation as a $\bar{\Gamma}$-tree, A_G accepts T_W if and only if W contains a path of the form $(\rightarrow -^*)^m$. The states of A_G are of the form $\langle \uparrow\colon\ i, \downarrow\colon\ j, \mathrm{Max} : k \rangle$ for $0 \leqslant i, j \leqslant k \leqslant m$, which ensures that A_G is of size polynomial in $|G|$ (and we will construct it in PTIME from G). The idea is that when a node n of T_W will be in such a state, it will mean that:

- Letting W_n be the subinstance of W which is represented by the subtree of T_W rooted at n, and letting r_n be the root of W_n, the longest directed upwards path in W_n finishing at r_n has length i (the path is the longest of the form $(\uparrow -^*)^*$ that ends at r_n).
- The longest directed downwards path in W_n beginning at r_n has length j (the path is the longest of the form $(\downarrow -^*)^*$ that begins at r_n).
- The longest directed path in W_n has length k (the path is of the form $(\rightarrow -^*)^k$ and is the longest in W_n).

We now describe the initialization function ι of A_G:

- $\iota((s,0)) := \langle \uparrow\colon\ 0, \downarrow\colon\ 0, \mathrm{Max} : 0 \rangle$ for any $s \in \Gamma$.
- $\iota((-,1)) := \langle \uparrow\colon\ 0, \downarrow\colon\ 0, \mathrm{Max} : 0 \rangle$.
- $\iota((\uparrow,1)) := \langle \uparrow\colon\ 1, \downarrow\colon\ 0, \mathrm{Max} : 1 \rangle$.
- $\iota((\downarrow,1)) := \langle \uparrow\colon\ 0, \downarrow\colon\ 1, \mathrm{Max} : 1 \rangle$.
- $\Delta((\uparrow,1), \langle \uparrow\colon\ i, \downarrow\colon\ j, \mathrm{Max} : k \rangle, \langle \uparrow\colon\ i', \downarrow\colon\ j', \mathrm{Max} : k' \rangle) := \langle \uparrow\colon\ i'', \downarrow\colon\ 0, \mathrm{Max} : k'' \rangle$ where $i'' := \min(m, \max(i+1, i'+1))$ and $k'' := \min(m, \max(i'', i+j', i'+j, k, k'))$.

- $\Delta((\downarrow,1),\langle\uparrow\colon\ i,\downarrow\colon\ j,\mathrm{Max}:k\rangle,\langle\uparrow\colon\ i',\downarrow\colon\ j',\mathrm{Max}:k'\rangle) := \langle\uparrow\colon\ 0,\downarrow\colon\ j'',\mathrm{Max}:k''\rangle$ where $j'' := \min(m,\max(j+1,j'+1))$ and $k'' := \min(m,\max(j'',i+j',i'+j,k,k'))$.
- $\Delta((-,1),\langle\uparrow\colon\ i,\downarrow\colon\ j,\mathrm{Max}:k\rangle,\langle\uparrow\colon\ i',\downarrow\colon\ j',\mathrm{Max}:k'\rangle) := \langle\uparrow\colon\ i'',\downarrow\colon\ j'',\mathrm{Max}:k''\rangle$ where $i'' := \max(i,i')$ and $j'' := \max(j,j')$ and $k'' := \min(m,\max(k,k',i+j',i'+j))$.
- $\Delta((s,0),\langle\uparrow\colon\ i,\downarrow\colon\ j,\mathrm{Max}:k\rangle,\langle\uparrow\colon\ i',\downarrow\colon\ j',\mathrm{Max}:k'\rangle) := \langle\uparrow\colon\ 0,\downarrow\colon\ 0,\mathrm{Max}:k''\rangle$ where $k'' := \min(m,\max(k,k',i+j',i'+j))$ for every $s\in\{-,\uparrow,\downarrow\}$.

The final state of A_G are all the states $\langle\uparrow\colon\ i,\downarrow\colon\ j,\mathrm{Max}:k\rangle$ such that $k=m$. One can check by a straightforward induction that the semantics of each state is respected, so that indeed the automaton tests the query G.

We conclude thanks to Proposition 3.1 of [5] by computing in linear time in $|A_G|$ and $|H'|$ a representation of the lineage on H' of the query that checks whether the input contains a directed path of the form $(\rightarrow\ {}^{-*})^m$, and observe by Theorem 6.11 of [6] that it is a d-DNNF. We then compute the probability of this d-DNNF [20], yielding $\Pr(G\rightsquigarrow H)$ in PTIME: this concludes the proof. □

D. Proof of Hardness of Counting Edge Covers

In this appendix, following a connection pointed out in [34], we give a proof of the following strengthening of Theorem 3.2, which is independent from the proof of [25].

Theorem D.1. *The* `#Bipartite-Edge-Cover` *problem is #P-complete. Hardness holds even for 2–3 regular bipartite undirected graphs that are planar.*

Proof. Membership in #P is straightforward: the machine guesses a subset of edges and accepts in PTIME iff the subset is a matching. Hence, we focus on hardness.

Recall that 2–3 *regular bipartite undirected graphs* are bipartite undirected graphs $\Gamma=(U\sqcup V,E)$ where the degree of each vertex in U is 2 and that of each vertex in V is 3. We will show how the result derives from the holographic reduction results of [13].

For $t\in U\sqcup V$, we denote by $E(t)$ the set of edges to which t is adjacent. For a valuation of the edges $\nu: E\to\{0,1\}$ and a vertex t, we write $\nu(E(t))$ the multiset $\{\!\{\nu(e)\mid e\in E(t)\}\!\}$. Given a multiset of bits B, the *Hamming weight* of B is the number of 1 bits in B. For each $x_0,\ldots,x_n\in\{0,1\}$, let $[x_0,\ldots,x_n]$ denote the function that takes a multiset of n bits as input and outputs x_i if the Hamming weight of those n bits is i.

For every $x_0,x_1,x_2,y_0,y_1,y_2,y_3\in\{0,1\}$, the problem $\#[x_0,x_1,x_2]|[y_0,y_1,y_2,y_3]$ is the following [13]: given a 2–3 regular bipartite undirected graph $\Gamma=(U\sqcup V,E)$, compute the quantity

$$\sum_{\nu:E\to\{0,1\}}\prod_{u\in U}[x_0,x_1,x_2](\nu(E(u)))\times\prod_{v\in V}[y_0,y_1,y_2,y_3](\nu(E(v))).$$

Then, when we restrict our attention to 2–3 regular bipartite undirected graphs, our problem `#Bipartite-Edge-Cover` can be seen to be the same as $\#[0,1,1]|[0,1,1,1]$. Indeed, seeing a valuation ν of the edges as a set of edges, the value under the sum for a valuation ν will be 1 if and only if, for every vertex, there exists an adjacent edge such that $\nu(e) = 1$, which exactly means that ν is an edge cover of Γ.

Now, let us consider the problem $\#[1,1,0]|[1,1,1,0]$. As observed at the end of Section 8 of [12], it is the *reversal* of the problem $\#[0,1,1]|[0,1,1,1]$. Indeed, the problem $\#[1,1,0]|[1,1,1,0]$ amounts to counting the number of subsets S of edges such that, for every vertex v, there exists at least one edge adjacent to v that is not in S, i.e., that is in $E \setminus S$. But this means that $\#[1,1,0]|[1,1,1,0]$ counts the number of sets S such that $E \setminus S$ is an edge cover of Γ. As there is a trivial bijection between the sets S that are edge covers and the sets S' such that $E \setminus S'$ is an edge cover, `#Bipartite-Edge-Cover` is PTIME-equivalent to $\#[1,1,0]|[1,1,1,0]$ on 2–3 regular bipartite undirected graphs.

Now, the problem $\#[1,1,0]|[1,1,1,0]$ is shown in [12, 13] to be #P-hard. However, one subtle problem is that the notion of graph used in these works is different from our own notion, because they generally allow the graph to contain multiple occurrences of the same edge, and they allow *self-loops* (edges from a vertex to itself): this is what we would call a *multigraph*. By contrast, the graphs that we used throughout the paper are *simple graphs*, where edges cannot be repeated, and we disallow self-loops. Now, even though this is implicit in [12, 13], the line of works on Holant problems typically consider multigraphs instead of graphs (e.g., see footnote 1 page 217 of [35]), so the hardness result in [12, 13] would in principle apply to input *multigraphs*. Thus, it does not directly allow us to conclude the proof of our result, as we stated that the problem `#Bipartite-Edge-Cover` is #P-hard on *simple* graphs. For this reason, to conclude our proof, we will follow in detail the reasoning used by [12] to show that $\#[1,1,0]|[1,1,1,0]$ is #P-hard on 2–3 regular bipartite planar (multi)graphs, and check that their argument in fact establishes hardness even when the input graphs are required to be simple.

The first step of the hardness proof for $\#[1,1,0]|[1,1,1,0]$ in [12] is [12, Lemma 4.1]. For our purposes, this lemma shows that the problem $\#[x_0,x_1,x_2]|[1,1,1,0]$ is #P-hard for some $x_0, x_1, x_2 \in \mathbb{R}$ (which uses some generalized definition of the notation $\#[x_0,x_1,x_2]|[y_0,y_1,y_2,y_3]$ that allows values outside of $\{0,1\}$), and that this hardness holds even for inputs that are 2–3 regular bipartite planar (multi)graphs. We want to show that this is true also when the input graphs are required to be simple. This hardness result in [12, Lemma 4.1] is shown by reducing from the problem of counting vertex covers on 3-regular planar (multi)graphs, which is #P-hard by [36]: as they explain, this implies that the problem $\#[0,1,1]|[1,0,0,1]$ is #P-hard on 2–3 regular bipartite planar (multi)graphs. Now, we note that [36, Theorem 9] actually shows hardness of the problem #3RBP-VC of counting vertex covers on 3-regular planar (multi)graphs that are *bipartite*: this means in particular that these input graphs cannot contain self-loops (but they can contain multi-edges). We can use this to show #P-hardness of the problem $\#[0,1,1]|[1,0,0,1]$ on 2–3 regular bipartite *simple* graphs. Indeed, as in [12, Lemma 4.1], we reduce from #3RBP-VC by taking an input graph G to #3RBP-VC and creating a new graph G' by subdividing each edge (adding a vertex of degree 2 in the middle of the edge). Now, the resulting graph G' is 2–3 regular bipartite, it is still planar, it con-

tains no multi-edges because G does not contain any self-loops (any multi-edges in G get translated to different edges in G' as each of them is subdivided with a different middle vertex), and the answer to #3RBP-VC on G is exactly the answer to $\#[0,1,1]|[1,0,0,1]$ on G'. Indeed, as explained in [12, Lemma 4.1], counting the vertex covers on G amounts to counting the number of subsets S of the vertices of degree 3 of G' such that every vertex of degree 2 is adjacent to a vertex of that subset. Equivalently, this is counting subsets S' of the edges of G' such that each vertex of degree 2 is adjacent to an edge of S' (hence the constraint $[0,1,1]$) and such that for every vertex of degree 3 we either keep all incident edges in S' (i.e., we keep the vertex in S) or keep none (i.e., we do not keep the vertex in S), hence the constraint $[1,0,0,1]$. Thus, this reduction shows that the problem $\#[0,1,1]|[1,0,0,1]$ on 2–3 regular bipartite planar graphs is #P-hard, even when the input graphs are required to be simple. Let us denote this problem by $(\star)$.

The rest of the proof of [12, Lemma 4.1] uses a holographic reduction to reduce problem $(\star)$ to the problem $\#[x_0,x_1,x_2]|[1,1,1,0]$, for some $x_0,x_1,x_2 \in \mathbb{R}$. More precisely, the hardness is shown by reducing either from problem $(\star)$, or from a different problem of counting matchings, depending on the signature that we are interested in. Specifically, the reduction from $(\star)$ is used when reducing to a signature $\#[x_0,x_1,x_2]|[y_0,y_1,y_2,y_3]$ when we can find values $\alpha_1,\alpha_2,\beta_1,\beta_2$ such that $\alpha_1\beta_2-\alpha_2\beta_1 \neq 0$ and we have $y_i = \alpha_1^{3-i}\alpha_2^i + \beta_1^{3-i}\beta_2^i$ for all i. For the signature that we are interested in, namely $[y_0,y_1,y_2,y_3]=[1,1,1,0]$, we can take $\alpha_1 := 0, \alpha_2 := -1, \beta_1 := 1, \beta_2 := 1$ and verify that the conditions are satisfied, so indeed the reduction is from problem $(\star)$. The reduction from problem $(\star)$ in [12, Lemma 4.1] is a holographic reduction, which does not modify the input graphs (see for instance [35, Section 3.2] for an introduction to holographic reductions). Thus, it follows from the proof of [12, Lemma 4.1] that there is a choice of $x_0,x_1,x_2 \in \mathbb{R}$ such that the problem $\#[x_0,x_1,x_2]|[1,1,1,0]$ is #P-hard on 2–3 bipartite planar graphs that are simple.

Then, the authors of [12] use what is called the *interpolation technique*, in order to show that any problem of the form $\#[x_0,x_1,x_2]|[1,1,1,0]$ is polynomial-time reducible (under Turing reductions) to the problem $\#[1,1,0]|[1,1,1,0]$, which is the problem that we are interested in. This is done in the appendix of [12], on page 15, by replacing the degree-2 nodes of the input graph by some graph gadgets. On page 15 we see that they use graph gadget number 1, and by looking at that gadget (Figure 3), we see that performing replacements with this gadget can never introduce parallel edges or self-loops. Thus, as the reduction is from simple graphs (as we argued in the preceding paragraphs), the graphs that are images of this reduction are also simple graphs. Thus, the proof of [12] actually shows that the problem $\#[1,1,0]|[1,1,1,0]$ is #P-hard on 2–3 regular bipartite planar graphs that are simple. This is the result needed to conclude the proof. □