The Complexity of Weighted Counting for Acyclic Conjunctive Queries

Arnaud Durand

IMJ UMR 7586 - Logique

Université Paris Diderot

F-75205 Paris, France

durand@logique.jussieu.fr

Stefan Mengel[*]

Institute of Mathematics

University of Paderborn

D-33098 Paderborn, Germany

smengel@mail.uni-paderborn.de

November 27, 2012

This paper is a study of weighted counting of the solutions of acyclic conjunctive queries (ACQ). The unweighted quantifier free version of this problem is known to be tractable (for combined complexity), but it is also known that introducing even a single quantified variable makes it #**P**-hard. We first show that weighted counting for quantifier-free ACQ is still tractable and that even minimalistic extensions of the problem lead to hard cases. We then introduce a new parameter for quantified queries that permits to isolate a large island of tractability. We show that, up to a standard assumption from parameterized complexity, this parameter fully characterizes tractable subclasses for counting weighted solutions of ACQ queries. Thus we completely determines the tractability frontier for weighted counting for ACQ.

1 Introduction

Conjunctive query answering (CQ for short) is a fundamental problem from database theory. It consists of evaluating a $\{\exists, \wedge\}$-formula of first-order logic over a finite structure. It is equivalent to the so-called Select-Project-Join queries in SQL and has several equivalent definitions, in particular, in terms of constraint satisfaction or homomorphism problems. The problem is known to be **NP**-complete [CM77], but due to its naturalness and its practical and theoretical importance, the search for tractable fragments has attracted a lot of attention.

One source of tractability is putting structural conditions onto the graph or hypergraph associated to the formula. It turns out, indeed, that a number of such structurally restricted classes of conjunctive queries admit efficient algorithms. When the arity of queries is bounded i.e. for problems over a fixed language (such as graphs or ternary relations), it has been proved that being of bounded treewidth precisely characterizes tractability [GSS01, Gro07]. For unbounded arity, the situation is more complex since no ultimate feasible class of queries is known. In this context, acyclic conjunctive queries (ACQ, for short) form a large and useful fragment which is well-known to be tractable (see [Yan81, GLS01]) and is also the base case of numerous other classes, obtained by hypergraph decomposition methods, for which evaluation is tractable (see [GLS02, CJG08, Mik08].

Counting solutions to database queries is a basic operation of standard database systems, in particular in the context of aggregate queries. The problem has been extensively studied in constraint satisfaction in the non uniform setting i.e. when the structure is fixed and only the formula is considered as input. As a generalization of #3SAT, it is easy to see that counting solutions to unquantified (i.e. projection free) conjunctive queries is #**P**-complete. Recently, Pichler and Skritek [PS11] showed that the restriction to quantifier free acyclic queries gives tractable instances for counting like it does for decision. This result also extends to other classes for which decision is known to be tractable such as bounded hypertree width but, until now, for unbounded arity, there is no criterion completely characterizing when a class of counting query is tractable. For bounded arity however, the situation is, again, better understood. Dalmau and Jonsson proved in [DJ04] that counting the solutions for a class of queries is polynomial time if and only if the query is of bounded treewidth. A consequence of a latter work by Mengel [Men11] is that on the bounded arity case, weighted counting on bounded treewidth queries is easy, too. So in a sense if we do not allow quantification, not much changes if we go from decision to counting problems.

Clearly, existential quantification — or equivalently the possibility to project out part of the result before counting — may be a source of hardness. This is confirmed for the general case for which the picture changes completely when existential quantification of variables is allowed. While the complexity of the decision problem remains unchanged, it turns out that counting solutions of general conjunctive queries becomes much harder since it is complete for $\# \cdot \mathbf{NP}$ [BCC$^+$05]. More surprisingly, also the acyclic case is no longer tractable. Pichler and Skritek [PS11] showed that introducing one single existential quantifier allows constructing #**P**-hard instances for very simple acyclic formulas. An immediate consequence of their result is that counting solutions to unions of a polynomial number of acyclic conjunctive queries is #**P**-hard, too, while the decision problem is, again, tractable. This shows that counting and decision differ fundamentally for acyclic conjunctive queries and that in order to find islands of tractability for counting we need new concepts that are tailored specifically to counting problems.

The main contribution of this paper is to give a complete picture of the (combined) complexity of weighted counting problems related to acyclic conjunctive queries. We consider weighted counting problems associated to instances of quantifier free ACQ and present algorithms that compute arithmetic circuits which, in turn, can be evaluated to

obtain the result of counting problems. Thus we separate structural manipulations of the CQ instance from the counting process itself. This separation is implicit in other papers (see e.g. [FMR08]) and is made explicit here. In this context, we first prove (Theorem 3, generalizing [PS11]) that computing the sum of the weights of solutions[1] of a (weighted) quantifier free ACQ can be done in polynomial time. Then, considering extensions of the problem above, we show that counting the number of solutions of a conjunction or a disjunction of *two* quantifier free ACQ is #**P**-complete even for Boolean domain and for fixed arity (see Proposition 5). This last result and the fact that one single quantification is enough to define hard cases (see [PS11]) shows that even "minimalistic" extensions of quantifier free ACQ lead to intractable counting problems and that one cannot hope to get any meaningful tractable class by these means.

However, our second set of results counterbalances this impression. We show that a large subclass of ACQ with existentially quantified variables leads to efficiently solvable counting problems. To this aim, we introduce a (hypergraph) parameter, called *quantified star size*, to measure the degree of dispersion of free variables in acyclic conjunctive queries (this generalizes the notion of connex-acyclicity of [BDG07]). We show (see Theorem 14) that if this parameter is bounded by some constant $k \in \mathbb{N}$, the resulting (weighted) counting problem is solvable in time $n^{O(k)}$ where n is the size of the instance. Furthermore, we show (see Theorem 16) that the quantified star size of a formula is efficiently computable making the counting result applicable. We also show that the runtime bound $n^{O(k)}$ can probably not be improved substantially in the sense that the parameterized problem is not fixed parameter tractable under the standard hypothesis $\mathbf{FPT} \neq \#\mathbf{W}[\mathbf{1}]$ from parameterized complexity [FG04, FG06]. Under the same hypothesis, we finally show (Theorem 22) that quantified star size is optimal in the sense that any other structural restriction on acyclic conjunctive queries that leads to polynomial time counting of the solutions must imply bounded quantified star size.

In this paper, we limit our study to acyclic conjunctive queries. There are several reasons to do so. One of them is that acyclicity provides a natural and clear setting to present our approach on charting the tractability frontier for conjunctive queries. Another reason is that acyclicity is also the base case of numerous other tractable fragments and although some additional work is required to extend our methods, it is really the fragment that needs to be investigated and solved first.

A nice feature of our results, is their consequences on another direction of research that concerns the complexity of families of polynomials. Indeed, our study of counting problems is mostly formulated in the slightly more general setting of arithmetic circuit complexity. So a consequence of our results is that they also provide information about the structural complexity of families of polynomials computed by arithmetic circuits and in particular on tractability and intractability in this context. As an example, it is an immediate corollary of our results that the polynomial families we consider which are built over bounded quantified star size queries characterize $\mathbf{VP}$, a class that formalizes efficient computation in the Valiant model [Val79] and is well known for its apparent lack

---

[1]Provided the evaluation space itself admits efficient algorithms for multiplication and addition of weights (such as the field of rationals $\mathbb{Q}$)

of natural non-circuit characterizations. Analog completeness results can also be derived from our hardness results too. We refer the reader to Section 6 for an exposition of the motivations and the results obtained in this setting.

Related works   Counting complexity has deserved a lot of attention in the context of non uniform CSP i.e. when only the formula is part of the input and the language (i.e. the structure) is fixed. A dichotomy result has been given for $\sharp CSP$ in [Bul08] using tools of universal algebra. This dichotomy has been reproved by simpler means and proved to be effective in [DR10] (see also [DGJ09, CCL11] and [BDG$^+$10] for similar result on partition functions). Another related approach that has recently received a lot of attention is based on holographic algorithms and has lead to major advances in the understanding of counting complexity. Fixing the input structure results in a problem of a very different nature and to our knowledge the tools and technics developed in the area does not apply to our setting.

Structure of the paper   Necessary preliminaries about arithmetic circuits, conjunctive queries and acyclicity are given in Section 2. In Section 3 we show that the weighted counting of solutions to acyclic conjunctive queries is easy extending the result of [PS11] but also some results in [Men11]. In Section 4 we show that unions of solutions of acyclic conjunctive queries are hard to count. Afterwards we turn to quantified star size to give a parameterization of counting solutions to quantified acyclic queries in Section 5. In passing, we prove that determining the quantified star size of a query (i.e. of the hypergraph associated to the query) can be done in polynomial time. Here we also show hardness in the sense of parameterized complexity and show that quantified star size likely is the most general restriction that leads to polynomial time counting. Finally, in section 6 we apply our results to arithmetic circuit complexity.

2   Preliminaries

Arithmetic circuit complexity   An *arithmetic circuit* over a field $\mathbb{F}$ is a labeled directed acyclic graph (DAG) consisting of vertices or gates with indegree or fanin 0 or 2. The gates with fanin 0 are called input gates and are labeled with constants from $\mathbb{F}$ or variables $X_1, X_2, \ldots, X_n$. The gates with fanin 2 are called computation gates and are labeled with $\times$ or $+$. The polynomial computed by an arithmetic circuit is defined in the obvious way: An input gate computes the value of its label, a computation gate computes the product or the sum of its childrens' values, respectively. We assume that a circuit has only one sink which we call output gate. We say that the polynomial computed by the circuit is the polynomial computed by the output gate. The *size* of an arithmetic circuit is the number of gates. The *depth* of a circuit is the length of the longest path from an input gate to the output gate in the circuit. A circuit is called *multiplicatively disjoint* if, for each $\times$-gate, its two input subcircuits are disjoint.

Conjunctive query: decision, counting and weighted counting. We assume the reader to be familiar with the basics of (first order) logic (see [Lib04]). If ϕ is a first order formula, $\mathsf{var}(\phi)$ denotes the set of its variables, by $\mathsf{free}(\phi) \subseteq \mathsf{var}(\phi)$ the set of its free variables and $\mathsf{atom}(\phi)$ the set of its atomic formulas. Let $\mathbf{x} = x_1, ..., x_k$, we denote $\phi(\mathbf{x})$ the formula with free variables $\mathbf{x}$. The *Boolean query problem* $\Phi = (\mathcal{S}, \phi)$ associated to a formula $\phi(\mathbf{x})$ and a structure $\mathcal{S}$, asks whether the set

$$\phi(\mathcal{S}) = \{\mathbf{a} : (\mathcal{S}, \mathbf{a}) \models \phi(\mathbf{x})\}$$

called the *query result* is empty or not. The (general) query problem consists of computing the set $\phi(\mathcal{S})$, while the corresponding counting problem is computing the size of $\phi(\mathcal{S})$, denoted by $|\phi(\mathcal{S})|$. When ϕ is a $\{\exists, \wedge\}$-first order formula the boolean query problem is known as the *Conjunctive Query Problem*, CQ for short. It is well known that the the Boolean CQ problem is equivalent to the propositional satisfiability problem and thus is **NP**-complete. We denote by #CQ the associated counting problem: given a query instance $\Phi = (\mathcal{S}, \phi)$, return the value of $|\phi(\mathcal{S})|$.

Let now $\mathbb{F}$ be a field and $\mathcal{S}$ be a finite structure of domain D. A $\mathbb{F}$-weight function for $\mathcal{S}$ is a mapping $w : D \to \mathbb{F}$. If $\mathbf{a}$ is a tuple of elements of D of length k, the weight of $\mathbf{a}$ is

$$w(\mathbf{a}) = \prod_{i=1}^{k} w(a_i).$$

The *weighted* counting problem for CQ, denoted $\#_{\mathbb{F}}$CQ, is the following problem: given $\Phi = (\mathcal{S}, \phi)$ and an $\mathbb{F}$-weighted function w, return the sum of the weights of all solutions i.e. the value of

$$\sum_{\mathbf{a} \in \phi(\mathcal{S})} w(\mathbf{a}).$$

When w is the constant function 1, this value is clearly equal to $|\phi(\mathcal{S})|$.

Query size and Model of computation. The size $\|\Phi\|$ of a query input is the sum of the size of the formula $|\phi|$ and of the size $|\mathcal{S}|$ of $\mathcal{S}$. We consider the size $|\mathcal{S}|$ of $\mathcal{S}$ with domain D to be the number of elements in D plus the number of tuples in relations of $\mathcal{S}$.

All algorithms below are expressed in terms of operations on tuples (without consideration on their size). Hence, we choose the RAM model (with addition as basic operation) under uniform cost measure as underlying model of computation. However, the choice of a model will have some importance only in the few cases where precise polynomial time bounds are given.

Acyclic Conjunctive Queries. A (finite) hypergraph $\mathcal{H}$ is a pair (V, E) where V is a finite set and $E \subseteq \mathcal{P}(V)$. We associate a hypergraph $\mathcal{H} = (V, E)$ to a formula ϕ (the *canonical* structure describing ϕ) by setting $V := \mathsf{var}(\phi)$ and $E := \{\mathsf{var}(a) \mid a \in \mathsf{atom}(\phi)\}$.
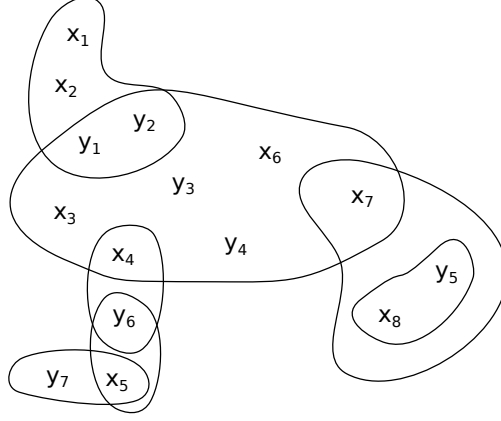
Figure 1: The hypergraph associated to the formula from Example 1.

Example 1. *The formula*

$$\varphi(y_1, \ldots y_7) := \exists x_1 \ldots \exists x_8 \quad R(x_1, x_2, y_1, y_2) \wedge S(y_2, y_1, x_7, y_4, x_6, y_3, x_4, x_3) \wedge$$
$$T(x_4, y_6) \wedge T(y_6, x_5) \wedge T(x_5, y_7) \wedge P(x_7, y_5, x_8) \wedge T(x_8, y_5)$$

has the associated hypergraph illustrated in Figure 1.

Definition 1. *A join tree (or tree structure) of a hypergraph $\mathcal{H} = (V, E)$ is a pair $(\mathcal{T}, \lambda)$ where $\mathcal{T} = (V_T, T)$ is a tree and λ is a function from V_T to E such that:*

- *for each $e \in E$, there is a $t \in V_T$ such that $\lambda(t) = e$,*

- *For each $v \in V$, the set $\{t \in V_T : v \in \lambda(t)\}$ is a connected subtree of T.*

A hypergraph is acyclic if it has a join tree [Fag83]. When there is no ambiguity, we often identify vertices of a join tree and their labellings. We also sometimes only specify the tree T explicitly, without talking of λ which is always implicitly understood to exist. A formula ϕ is acyclic if its associated hypergraph is acyclic. Considering acyclic $\{\exists, \wedge\}$-first order formulas yields the Boolean acyclic conjunctive query problem denoted ACQ. We denote by #ACQ (resp. $\#_{\mathbb{F}}$ACQ) the associated counting (resp. $\mathbb{F}$-weighted counting) problem. If ϕ is such that $\mathsf{free}(\phi) = \mathsf{var}(\phi)$) then ϕ is said to be *quantifier-free*.

Example 2. *The formula $\varphi(y_1, \ldots, y_7)$ from example 1 is acyclic. It's join tree is depicted in Figure 2. It holds: $\mathsf{free}(\varphi) = \{y_1, \ldots y_7\}$ and $\mathsf{var}(\varphi) = \{x_1, \ldots, x_8, y_1, \ldots y_7\}$.*

Any $\mathbf{a} \in \phi(\mathcal{S})$ will be alternatively seen as an assignment $\mathbf{a} : \mathsf{var}(\phi) \to D$ or as a tuple of dimension $|\mathsf{var}(\phi)|$. Two arbitrary assignments $\mathbf{a}$ and $\mathbf{a}'$ are *compatible* (symbol: $\mathbf{a} \sim \mathbf{a}'$) if they agree on their common variables. We will make use of the following classical join operators.
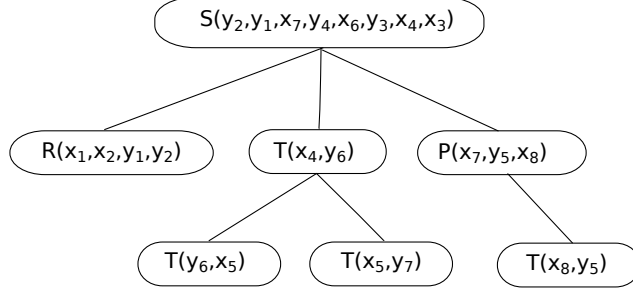
Figure 2: A join tree for the formula from Example 1.

Definition 2. *Let* $\phi(\mathbf{x}, \mathbf{y})$, $\psi(\mathbf{y}, \mathbf{z})$ *be two conjunctive queries with* $\mathbf{x} \cap \mathbf{z} = \emptyset$ *and let* $\mathcal{R}, \mathcal{S}$ *be two finite structures. Let us define:*

   - The natural join: $\phi(\mathcal{R}) \bowtie \psi(\mathcal{S}) = \{(\mathbf{a}, \mathbf{b}, \mathbf{c}) : (\mathbf{a}, \mathbf{b}) \in \phi(\mathcal{R})$ *and* $(\mathbf{b}, \mathbf{c}) \in \psi(\mathcal{S})\}$

   - The (left) semi-join: $\phi(\mathcal{R}) \ltimes \psi(\mathcal{S}) = \{(\mathbf{a}, \mathbf{b}) : (\mathbf{a}, \mathbf{b}) \in \phi(\mathcal{R})$ *and there exists* $(\mathbf{b}, \mathbf{c}) \in \psi(\mathcal{S})\}$.

When $\mathcal{R} = \mathcal{S}$, $\phi(\mathcal{R}) \bowtie \psi(\mathcal{S})$ is simply $[\phi \wedge \psi](\mathcal{R})$. The natural join between two relations can be computed in time linear in the size of the relations (here in time linear in $|\phi(\mathcal{R})|$ and $|\psi(\mathcal{S})|$).

Remark 1 (CQ and CSP). *A well-known equivalent formulation of the quantifier-free conjunctive query problem can be stated in terms of* CSP. *In this later problem, given two structures* $\mathcal{S}$ *and* $\mathcal{T}$, *one asks whether there exists an homomorphism from* $\mathcal{S}$ *to* $\mathcal{T}$. *In the recent past, counting and weighted counting for* CSP *have been mainly stated in the non uniform version of the problem i.e. when the template* $\mathcal{T}$ *is fixed and only* $\mathcal{S}$ *is given as input (see e.g. [DR10, BDG$^+$10, DGJ09]). In this paper, we focus on the uniform version of the problem.*

Polynomials defined by conjunctive queries   We briefly introduce a polynomial Q that generalize #CQ. A more thorough discussion of Q can be found in [Men11].

Let $\Phi = (\phi, \mathcal{S})$ be a query with domain D. We assign to Φ the following polynomial $Q(\Phi)$ in the variables $\{X_d \mid d \in D\}$.

$$Q(\Phi) := \sum_{a \in \phi(\mathcal{S})} \prod_{x \in \mathsf{var}(\phi)} X_{a(x)} = \sum_{a \in \phi(\mathcal{S})} \prod_{d \in D} X_d^{\mu_d(a)},$$

where $\mu_d(a) = |\{x \in \mathsf{var}(\phi) \mid a(x) = d\}|$ computes number of variables mapped to d by a. Note that the number of variables in $Q(\Phi)$ is $|D|$, the size of the domain, and that $Q(\Phi)$ is homogeneous of degree $|\mathsf{var}(\Phi)|$.

Observe that Q is essentially the weighted counting problem $\#_{\mathbb{F}}\text{CQ}$ by setting $X_d := w(d)$ for all $d \in D$. Thus if we can efficiently compute circuits that in turn compute $Q(\Phi)$, we can efficiently solve $\#_{\mathbb{F}}\text{ACQ}$ on the instance Φ.

Parameterized counting complexity   This section is a very short introduction to parameterized counting complexity (for more details see [FG04, FG06]).

A parameterized counting problem is a function $F : \Sigma^* \times \mathbb{N} \to \mathbb{N}$, for an alphabet Σ. Let $(x, k) \in \Sigma^* \times \mathbb{N}$, then we call x the input of F and k the parameter. A parameterized counting problem F is fixed parameter tractable, or $F \in \textbf{FPT}$, if there is an algorithm computing $F(x, k)$ in time $f(k) \cdot |x|^c$ for some computable function $f : \mathbb{N} \to \mathbb{N}$ and some constant $c \in \mathbb{N}$.

Let $F : \Sigma^* \times \mathbb{N} \to \mathbb{N}$ and $G : \Pi^* \times \mathbb{N} \to \mathbb{N}$ be two parameterized counting problems. A parameterized parsimonious reduction from F to G is an algorithm that computes for every instance (x, k) of F an instance (y, l) of G in time $f(k) \cdot |x|^c$ such that $l \leq g(k)$ and $F(x, k) = G(y, l)$ for computable functions $f, g : \mathbb{N} \to \mathbb{N}$ and a constant $c \in \mathbb{N}$. A parameterized T-reduction from F to G is an algorithm with an oracle for G that solves any instance (x, k) of F in time $f(k) \cdot |x|^c$ in such a way that for all oracle queries the instances (y, l) satisfy $l \leq g(k)$ for computable functions f, g and a constant $c \in \mathbb{N}$.

Let p-#Clique be the problem of counting k-cliques in a graph where k is the parameter and the graph is the input. A parameterized problem F is in #$\textbf{W}[\textbf{1}]$ if there is a parameterized parsimonious reduction from F to p-#Clique[2]. F is #$\textbf{W}[\textbf{1}]$-hard, if there is a parameterized T-reduction from p-#Clique to F. As usual, F is #$\textbf{W}[\textbf{1}]$-complete if it is in #$\textbf{W}[\textbf{1}]$ and hard for it, too.

A standard assumption from parameterized complexity is that there are problems in #$\textbf{W}[\textbf{1}]$ (in particular the complete problems) that are not fixed parameter tractable. Thus, from showing that a problem F is #$\textbf{W}[\textbf{1}]$-hard it follows that F can be assumed to be not fixed parameter tractable.

Except for these definitions we will not use parameterized parsimonious reductions and we will not use the complete power of parameterized T-reductions either. Instead, all parameterized reductions in the remainder of the paper will be T-reductions with exactly one oracle call.

3   Constructing circuits for acyclic conjunctive queries

Theorem 3. *Given an acyclic quantifier free conjunctive query Φ, we can in time polynomial in $\|\Phi\|$ compute a multiplicatively disjoint arithmetic circuit C that computes $Q(\Phi)$.*

Proof. The first step follows that of [Men11]. Since arity of queries is not bounded, a new approach is necessary to show that only polynomially many gates are necessary to compute $Q(\Phi)$. Also the algorithmic nature of the construction is stressed more to give the upper bound on the complexity of constructing the circuit C.

So let $\Phi = (\mathcal{S}, \phi)$ be an acyclic conjunctive query. Let $(\mathcal{T}, \lambda)$ the join tree associated with ϕ. By definition, the tree $\mathcal{T}$ has m vertices $t_1, ..., t_m$ associated to the atoms $\lambda(t_1), ..., \lambda(t_m)$

---

[2]Let us remark that Thurley [Thu06] gives good arguments for defining #$\textbf{W}[\textbf{1}]$ not with parsimonious reductions. He instead defines #$\textbf{W}[\textbf{1}]$ with parameterized T-reductions with only one oracle call. We keep the definition of [FG04, FG06], because we will show no #$\textbf{W}[\textbf{1}]$ upper bounds and thus can avoid these subtleties. We remark though that finding the right reduction notions for counting problems is notoriously tricky to get right (see e.g. [KPZ99, DHK05]).

in ϕ. Observe that $\mathcal{T}$ can be constructed from ϕ in polynomial time; indeed it can even be computed in logarithmic space, see [GLS01]. Thus we do not consider the construction of $\mathcal{T}$ but take it as given. For $t \in V_T$, we call ϕ_t the conjunction of constraints corresponding to the subtree $\mathcal{T}_t$ with t as root. The set $\mathsf{var}(\phi_t) = \bigcup_{t' \in \mathcal{T}_t} \mathsf{var}(\lambda(t'))$ is denoted by e_t. For convenience we also denote the atomic formula $\lambda(t)$ by λ_t.

Let $\mathbf{a}$ be an assignment of some variables of ϕ and $c \subseteq \mathsf{var}(\phi_t)$. We show by induction on the depth of $\mathcal{T}$ that the following polynomial in the variables $\{X_d \mid d \in D\}$ can be computed by an arithmetic circuit of polynomial size

$$f_{t,\mathbf{a},c} = \sum_{\substack{\alpha \in \phi_t(\mathcal{S}) \\ \alpha \sim \mathbf{a}}} \prod_{x \in c} X_{\alpha(x)}.$$

Remark that if r is the root of $\mathcal{T}$ then, $f_{r,\emptyset,\mathsf{var}(\phi)} = Q(\Phi)$. Observe that in contrast to [Men11] we have an exponential number of polynomials $f_{t,\mathbf{a},c}$, so we cannot afford to compute them all in a bottom up fashion. Instead we will construct the circuit top down starting from r and make sure that in each step only polynomially many $f_{t,\mathbf{a},c}$ are needed. This will directly give the runtime bound for the construction.

So suppose first that t is a leaf. Then, ϕ_t is some atomic constraint λ_t, hence $\phi_t(\mathcal{S})$ is of size at most linear in $|\mathcal{S}| \leq \|\Phi\|$, so the sum only involves a number of terms linear in $\|\Phi\|$.

Suppose now $t \in V_T$ is not a leaf and let $t_1, ..., t_k$ in V_T be the children of t in $\mathcal{T}$. Let $c_0, c_1, ..., c_k$ be a partition of c into disjoint sets such that each $c_i \subseteq e_i \cap c$, for $i = 1, ..., k$ and $c_0 \subseteq c \backslash \bigcup_{i=1}^{k} e_{t_i}$ (the need for choosing a partition is that each variable x appearing in several $\lambda(t_i)$ sets must be taken into account at most once in order not to overcount the exponent of $X_{a(x)}$).

$$\begin{aligned} f_{t,\mathbf{a},c} &= \sum_{\substack{\alpha \in \phi_t(\mathcal{S}) \\ \alpha \sim \mathbf{a}}} \prod_{x \in c} X_{\alpha(x)} \\ &= \sum_{\substack{\alpha \in \phi_t(\mathcal{S}) \\ \alpha \sim \mathbf{a}}} \prod_{x \in c_1} X_{\alpha(x)} \cdots \prod_{x \in c_k} X_{\alpha(x)} \prod_{x \in c_0} X_{\alpha(x)} \end{aligned}$$

Let $A_t = ((\lambda_t(\mathcal{S}) \ltimes \phi_{t_1}(\mathcal{S})) \ltimes \phi_{t_2}(\mathcal{S})) \ltimes \ldots \ltimes \phi_{t_k}(\mathcal{S})$. Note that $A_t \subseteq \lambda_t(\mathcal{S})$.

Claim 1. *The set A_t is computable in time $|\mathcal{S}| \times |\phi_t|$.*

Proof of the claim. By induction on the tree depth. The proof is an adaptation of Yannakakis algorithm to evaluate acyclic conjunctive queries (see [Yan81]). Note that A_t is a subrelation of $\phi_t(\mathcal{S})$ with ϕ_t being a constraint of the input formula. So if t is a leaf, the result is obvious. If t is not a leaf. Let $t_1, ..., t_k$ be its children. Remark that

$$A_t = (\lambda_t(\mathcal{S}) \ltimes A_{t_1}) \ltimes A_{t_2}) \ltimes \ldots \ltimes A_{t_k},$$

since each A_{t_i} is the projection of $\phi_{t_i}(\mathcal{S})$ onto $\mathsf{var}(\phi_{t_i})$. The set A_t is computed as follows: Semi-joins are computed step by step respecting the order given by parentheses. To compute each join sort the two sets of assignments using the lexicographic ordering

induced by the variables they have in common. Then, run once through the two sorted relations to select the right tuples. Suppose now that each A_{t_i} is computable in time $|\mathcal{S}| \cdot |\phi_{t_i}|$. Then, A_t is computable in time $|\mathcal{S}| \cdot (|\lambda_t| + |\phi_{t_1}| + \ldots + |\phi_{t_k}|) \leq |\mathcal{S}| \cdot |\phi_t|$. $\square$

Each solution $\alpha \in \phi_t(\mathcal{S})$ can be uniquely expressed as the natural join of a tuple $\beta \in A_t$ and a sequence of $\alpha_i \in \phi_{t_i}(\mathcal{S})$, $i = 1, ..., k$, compatible with β (more formally by natural join of singleton relations containing these tuples), i.e. given $\alpha \in \phi_t(\mathcal{S})$, there exist $\beta \in A_t$ and $\alpha_i \in \phi_{t_i}(\mathcal{S})$, $i = 1, ..., k$, such that

$$\{\alpha\} = \{\beta\} \bowtie \{\alpha_1\} \bowtie \ldots \bowtie \{\alpha_k\}.$$

Conversely, given $\beta \in A_t$ and a sequence of $\alpha_i \in \phi_{t_i}(\mathcal{S})$, $i = 1, ..., k$, compatible with β, the natural join of these tuples is an $\alpha \in \phi_t(\mathcal{S})$. This follows from the connectedness condition in the join tree, i.e. from the fact that given distinct $i, j \leq k$, $\mathsf{var}(\phi_{t_i}) \cap \mathsf{var}(\phi_{t_j}) \subseteq \mathsf{var}(\lambda_t)$. Indeed, if α_i and α_j assign values to of a common variable, they must agree on it, because they both agree with β. This implies that the following equalities hold.

$$
\begin{aligned}
f_{t,\mathbf{a},c} &= \sum_{\substack{\alpha \in \phi_t(\mathcal{S}) \\ \alpha \sim \mathbf{a}}} \prod_{x \in c_1} X_{\alpha(x)} \cdots \prod_{x \in c_k} X_{\alpha(x)} \prod_{x \in c_0} X_{\alpha(x)} \\
&= \sum_{\substack{\beta \in A_t \\ \beta \sim \mathbf{a}}} \sum_{\substack{\alpha_1 \in \phi_{t_1}(\mathcal{S}) \\ \alpha_1 \sim \beta}} \cdots \sum_{\substack{\alpha_k \in \phi_{t_k}(\mathcal{S}) \\ \alpha_k \sim \beta}} \prod_{x \in c_1} X_{\alpha_1(x)} \cdots \prod_{x \in c_k} X_{\alpha_k(x)} \prod_{x \in c_0} X_{\beta(x)} \\
&= \sum_{\substack{\beta \in A_t \\ \beta \sim \mathbf{a}}} f_{t_1,\beta,c_1} \cdots f_{t_k,\beta,c_k} \cdot \prod_{x \in c_0} X_{\beta(x)}
\end{aligned}
$$

Note that the sum is now over A_t and not over $\phi_t(\mathcal{S})$ anymore. We claim that the construction described above can be done in polynomial time. Indeed, for each t we only have to compute the $f_{t,\mathbf{a},c}$ for one fixed set c but for potentially all $\mathbf{a} \in A_{t'}$ where t' is the father of vertex t in $\mathcal{T}$. Thus for t we only have to compute $|A_{t'}| \leq |\mathcal{S}| \leq \|\Phi\|$ polynomials $f_{t,\mathbf{a},c}$. Furthermore for t we only have to access the polynomials for $t_1, ..., t_k$, more precisely all of $f_{t_1,\beta,c_1}, \ldots, f_{t_k,\beta,c_k}$ with $\beta \in A_t$. So the computation of one $f_{t,\mathbf{a},c}$ involves only $O(|A_t| \times (k + |c_0|)) = O(|\mathcal{S}| \times |\phi|)$ arithmetic operations. Computing the $f_{t,\mathbf{a},c}$ for all $\mathbf{a} \in A_{t'}$ but fixed t can then be done with $O(|\phi| \times |\mathcal{S}|^2)$ operations. Summing up over all t we get a total upper bound of $O(|\phi|^2 \times |\mathcal{S}|^2)$, so the circuit C for $Q(\Phi)$ is of polynomial size. Now remark that each set A_t, for $t \in V_T$ can be constructed in time $O(|\phi| \times |\mathcal{S}|)$ by Claim 1. For a fixed $\mathbf{a}$, filtering all elements β of A_t compatible with $\mathbf{a}$ can be done in linear time after sorting A_t. Hence, the index set of each sum is efficiently computable and the construction of the circuit can be done in polynomial time.

In a final step we apply the construction of Malod and Portier [MP08] to make the circuit multiplicatively disjoint. $\square$

We get the following corollary on weighted counting problems (which generalizes a recent result of Pichler and Skritek [PS11]).

Corollary 4. *Let $\mathbb{F}$ be a field such that iterated addition and multiplication are computable in polynomial time in $\mathbb{F}$. Then $\#_{\mathbb{F}}$ACQ can be solved in polynomial time for quantifier free queries.*

Proof. Given an instance Φ, Theorem 3 yields a circuit C that computes $Q(\Phi)$. Setting $X_d = w(d)$ for all $d \in D$, we can evaluate $Q(\Phi)$ to give the answer to the weighted counting problem. Observe that efficient evaluation is possible, because the degree of $Q(\Phi)$ is bounded and thus we can use standard depth reduction techniques to avoid a blowup of the size of representations of field elements. $\qquad\square$

4 Union and intersection of acyclic queries

In this section, we show that considering conjunction and disjunction of two acyclic conjunctive queries leads to intractable counting problems.

Proposition 5. *Computing the size of the union and the intersection of query results to two quantifier free $\#$ACQ-instances are both $\#\mathbf{P}$-complete. This result remains true for $\#$ACQ on boolean domain and arity at most 3.*

Remark 2. *In [GSS01], it is proved that the (bi-)colored grid homomorphism problem is $\mathbf{NP}$-complete. This result implies part of Proposition 5, i.e. that counting the assignments of the conjunction of two ACQ-instances is $\#\mathbf{P}$-complete (the fact that this hardness result is still true on Boolean domain does not follow, however).*

For the proof we use the following lemma:

Lemma 6. *Counting solutions to quantifier free conjunctive queries whose primal graph is a grid is $\#\mathbf{P}$-complete even for domains of size 4.*

Proof. Counting solutions to general quantifier free conjunctive queries is in $\#\mathbf{P}$, so we only need to show hardness. We show hardness by reducing a restricted version of $\#$circuitSAT to $\#$CQ with the desired grid structure. From the $\#\mathbf{P}$-completeness of our $\#$circuitSAT version we get $\#\mathbf{P}$-hardness for counting solutions of conjunctive queries with grid structure.

We now define this version of $\#$circuitSAT that we call $\#(\wedge\text{-}\neg\text{-grid})$-circuitSAT: An instance of $\#(\wedge\text{-}\neg\text{-grid})$-circuitSAT is a boolean circuit which only contains $\wedge$- and $\neg$-gates and in which all gates are vertices of a 2-dimensional grid. Furthermore, the edges of the circuit are non-intersecting paths along the edges of the grid.

Proposition 7. *$\#(\wedge\text{-}\neg\text{-grid})$-circuitSAT is $\#\mathbf{P}$-complete under parsimonious reductions.*

Proof. We make a parsimonious reduction from $\#$circuitSAT. Let C be a $\#$circuitSAT instance, i.e. a boolean circuit. In a first step we substitute all $\vee$-gates $x \vee y$ by $\neg(\neg x \wedge \neg y)$. We then make sure that every gate has at most degree 3 and that all input gates and the output gate have at most degree 2 by adding double negations. Call the resulting circuit C'.

11

We now embed C' into a grid. To do so we take a three step approach that starts with a coarse grid that is then refined. Let n be the size of C'. We first distribute the vertices that represent gates into a $n \times n$-grid G_1 such that each vertex of depth i has the coordinates (i, j) for some j. Furthermore each edge of the circuit is a sequence of straight lines where each straight line goes from a vertex in one row to another vertex in the next row. Also in each vertex of G_1 at most two lines start and end. For vertices on which no gate of C' lies, we assume that at most one edge starts and ends. It is clear that such an embedding can be constructed easily.

In a second step we make sure that the edges of the circuit follow the edges of a grid without congestion. We do this for each row of the coarse grid G_1 individually. We construct a new grid G_2 by adding $2n - 1$ new rows before each row in G_1 and one new column before each column. Observe that each vertex (i, j) in G_1 has the coordinates $(2ni, 2j)$ in G_2. Each vertex v of G_1 in row i has a most 2 outgoing straight lines l_1, l_2 representing edges of the circuit C' which both end in a vertex of row $i + 1$. Let l_1 end in $(i + 1, j)$ and l_2 end in $(i + 1, j')$ with $j < j'$, then we call l_1 be the low output and l_2 the high output. If there is only one output, we define it to be high. We also make the equivalent definition for high and low inputs.

Now we substitute the lines representing edges of C' by paths in G_2. Let l be a line that starts in G_1 in (i, j') and ends in $(i + 1, j)$. We construct a path P_l from $(2ni, 2j)$ to $(2n(i + 1), 2j')$:

- If l is a low output and a low input the path is the piecewise linear curve through the vertices $(2ni, 2j)(2ni, 2j - 1)(2ni + 2j, 2j - 1)(2ni + 2j, 2j')(2n(i + 1), 2j')$.

- If l is a high output and a low input the path is through $(2ni, 2j)(2ni + 2j + 1, 2j)(2ni + 2j + 1, 2j')(2n(i + 1), 2j')$.

- If l is a low output and a high input the path is through $(2ni, 2j)(2ni, 2j - 1)(2ni + 2j, 2j - 1)(2ni + 2j, 2j' + 1)(2n(i + 1), 2j' + 1)(2n(i + 1), 2j')$.

- If l is a high output and a high input the path is through $(2ni, 2j)(2ni + 2j + 1, 2j)(2ni + 2j + 1, 2j' + 1)(2n(i + 1), 2j' + 1)(2n(i + 1), 2j')$.

The result is an embedding of C' into a grid such that the gates are on vertices of G_2 and the edges of C' are paths in the grid. Observe that the paths were constructed in such a way that two paths between gates never share edges, so they only intersect in single vertices.

In the final step of the reduction we get rid of these intersections on non-gate vertices by adding additional gates. Each crossing in G_2 is substituted by the gadget illustrated in Figure 3. To do so we make the grid finer again by a constant factor. The result is a circuit C'' that is embedded into a grid. Furthermore C'' has the same satisfying assignments as C'.

$\square$

Remark 3. *We could also have given a proof of Proposition 7 with results on embedding general planar graphs into grids in the way we need it (see e.g. [Val81]). We have chosen to present an ad-hoc proof instead to keep the results of this paper self-contained.*
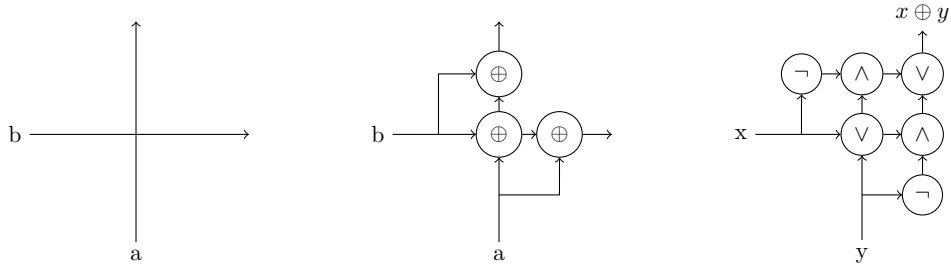
Figure 3: The crossing paths in the left are substituted by a gadget without crossings in the middle that uses $\oplus$-gates which compute xor of its inputs. It is easily checked that the outputs compute $(a \oplus b) \oplus a$ and $(a \oplus b) \oplus b$ which simplify to b and a respectively. On the right we show how the $\oplus$-gates can be simulated over the basis $\wedge, \vee, \neg$ without losing planarity. Degree 4 gates, splitting of edges and $\vee$-gates can be avoided by introducing some more $\neg$-gates and using De Morgan's law.

We now reduce $\#(\wedge\text{-}\neg\text{-grid})$-circuitSAT to $\#CQ$ instances of grid structure. So let (C, G) be an instance of $\#(\wedge\text{-}\neg\text{-grid})$-circuitSAT, i.e. a circuit C that is embedded into a grid G. Let G be of size $n \times n$. W.l.o.g. we may assume that no gates are on neighboring vertices in G and that the output gate is not a $\wedge$-gate. For each $\wedge$-gate af C we arbitrarily fix one input as the first input while the other one is the second one. We construct a binary conjunctive query Φ whose primal graph is G. The domain is $\{0, 1, 2, 3\}$ where 0 and 1 represent the usual boolean values while 2 and 3 are used in a gadget construction for $\wedge$-gates. For each edge $e = uv$ in G we add a constraint ϕ_e in the variables u and v in the following way:

- If e is not an edge of C, ϕ_e has the satisfying assignments $\{ab \mid a, b \in \{0, 1, 2, 3\}\}$.

- If e is an edge of C directed from u to v and v is not a gate and u is not a $\wedge$-gate, ϕ_e has the satisfying assignments $\{00, 11\}$.

- If e is an edge of C directed from u to v and v is a $\neg$-gate, ϕ_e has the satisfying assignments $\{01, 10\}$.

- If e is an edge of C directed from u to v and v is a $\wedge$-gate and the path to v over u is from the first input of v, ϕ_e has the satisfying assignment $\{00, 02, 11, 13\}$.

- If e is an edge of C directed from u to v and v is a $\wedge$-gate and the path to v over u is from the second input of v, ϕ_e has the satisfying assignment $\{00, 01, 12, 13\}$.

- If e is an edge of C directed from u to v and v is not a gate and u is a $\wedge$-gate, ϕ_e has the satisfying assignments $\{00, 10, 20, 31\}$.

Observe that the construction near the $\wedge$-gates is possible, because no two gates are neighbors. So the constraints are all well defined. Now each vertex that is not part of C

gets a unary constraint that has only the single satisfying assignment 1. Also the output gate of C gets such a unary constraint.

We claim that if we fix an assignment a to the variables representing the inputs of C, there is an satisfying extension to the other variables if and only if a satisfies C. Furthermore, this extension is unique. It is clear that the constraints along the paths and on the $\neg$-gates propagate the correct values along the grid. In a satisfying assignment, the variable representing an $\wedge$-gate has to take the value representing the values of its inputs in binary. The gates after the $\wedge$-gates then calculate the conjunction value for these inputs. $\qquad\square$

Proof of Proposition 5. Again, we only need to show hardness. By the inclusion-exclusion principle counting for unions and intersections is equally hard, so it suffices to show hardness for intersections. The reduction is straightforward with Lemma 6. Let Φ be a conjunctive query whose primal graph is a grid. We separate the constraints into two new formulas: Φ_1 gets all the constraints that lie on rows of the grid, Φ_2 gets those on the columns. Clearly we have $\Phi = \Phi_1 \wedge \Phi_2$ and the Φ_i are acyclic. Thus the first part of the lemma follows.

To show that the result is true for queries on boolean domain, we sketch a different encoding of $\wedge$-$\neg$-grid-#circuitSAT into conjunctive queries. Roughly speaking, the structure of the encoding is basically the same but non boolean elements are mapped to sequences of boolean variables (that represent their binary encodings). To do so we need ternary relations. For completeness, details are given below. Again let G be the $n \times n$ grid and suppose no gates are on neighboring vertices in G and that the output gate is not a $\wedge$-gate. For each $\wedge$-gate v, we introduce a second vertex/variable v_1. We construct a ternary CQ instance Φ as follows. For each edge $e = uv$ in G we add an constraint ϕ_e in the following way:

- If e is not edge of C , ϕ_e has the satisfying assignments $\{00, 01, 10, 11\}$.

- If e is an edge of C directed from u to v and v is not a gate and u is not a $\wedge$-gate, ϕ_e has the satisfying assignments $\{00, 11\}$.

- If e is an edge of C directed from u to v and v is a $\neg$-gate, ϕ_e is the constraint with the following satisfying assignments $\{01, 10\}$.

- If e is an edge of C directed from u to v and v is a $\wedge$-gate and the path to v over u is from the first input of v, ϕ_e is the ternary constraint on variables u, v, v_1 with the following satisfying assignment set $\{000, 010, 101, 111\}$.

- If e is an edge of C directed from u to v and v is a $\wedge$-gate and the path to v over u is from the second input of v, ϕ_e is the ternary constraint on variables u, v, v_1 which has the satisfying assignment $\{000, 001, 110, 111\}$.

- If e is an edge of C directed from u to v and v is not a gate and u is a $\wedge$-gate, ϕ_e is the constraint on variables u_1, u, v which has the satisfying assignments $\{000, 010, 100, 111\}$.

14

The constraint is then split into two conjunctive query instances Φ_1 and Φ_2 as above grouping horizontal and vertical constraints separately. Note that, connection at gates v between hyperedges is now on two vertices v and v_1. But the resulting hypergraphs for Φ_1 and Φ_2 are well acyclic. $\qquad\square$

The reductions of this section are all parsimonious, so we directly get the following corollary [3]:

Corollary 8. *Deciding if the intersection of the solutions of two quantifier free acyclic conjunctive queries is nonempty is* **NP**-*hard.*

Note that in contrast, it is well-known that deciding the disjunction of acyclic conjunctive queries can be done in time linear in $\|\Phi\|$.

5 Quantified star size

5.1 Definitions and statement of the results

It is proved in [PS11] that introducing one single existential quantifier in acyclic conjunctive queries leads to #**P**-complete problems. So bounding the number of quantified variables does not yield tractable instances. In this section, we will show that not the number of quantified variables is crucial but how they are distributed in the associated hypergraph. A basic observation on the hard instance in [PS11] is that the formula has a star (in the graph theoretical sense) in its associated graph whose center is the quantified variable. Abstracting this observation, we introduce a parameter called *quantified star size* that leads to tractable #ACQ instances.

Before we formulate the main results of this section, we make several definitions.

Let $\mathcal{H} = (V, E)$ be a hypergraph and $S \subseteq V$. The *induced subhypergraph* $\mathcal{H}[S]$ of $\mathcal{H}$ is the hypergraph $\mathcal{H}[S] = (S, \{e \cap S \mid e \in E, e \cap S \neq \emptyset\})$. The induced subhypergraph of an edge set $E' \subseteq E$ is $\mathcal{H}[E'] = (\bigcup_{e \in E'} e, E')$. Let $x, y \in V$, a *path* between x and y is a subset of edges $e_1, ..., e_k \in E$ such that $x \in e_1$, $y \in e_k$, and for all $i \leq k - 1$, $e_i \cap e_{i+1} \neq \emptyset$. Alternatively, a path can be seen as the sequence of vertices $(x, a_1, ..., a_k, y)$ such that $a_i \in e_i \cap e_{i+1}$. These definitions apply to graphs as well.

Observation 9. *If $\mathcal{H}$ is an acyclic hypergraph and $C \subseteq V(\mathcal{H})$, then $\mathcal{H}[C]$ is acyclic. If $\mathcal{T} = (V_T, T)$ is a join tree of $\mathcal{H}$ then $\mathcal{T}[C]$, the tree obtained from T by replacing for all vertices $t \in V(T)$ the labeling $\lambda(t)$ by $\lambda(t) \cap C$, is a join tree of $\mathcal{H}[C]$.*

Proof. Immediate. $\mathcal{T}[C]$ is a subforest of $\mathcal{T}$. The connectedness condition of the set $\{t \in V_{T[C]} : v \in \lambda(t)\}$, for all $v \in C$ is obviously true. $\qquad\square$

Definition 10 (*S*-component [BDG07]). *Let $\mathcal{H} = (V, E)$ be a hypergraph and $S \subseteq V$. Let $E_{\not\subseteq S}$ be the set of hyperedges $\{e \in E : e \not\subseteq S\}$. The S-component of $e \in E_{\not\subseteq S}$ is the hypergraph $\mathcal{H}[E']$ where E' is the set of all edges $e' \in E_{\not\subseteq S}$ such that there is a path from $e - S$ to $e' - S$ in $\mathcal{H}[V - S]$.*

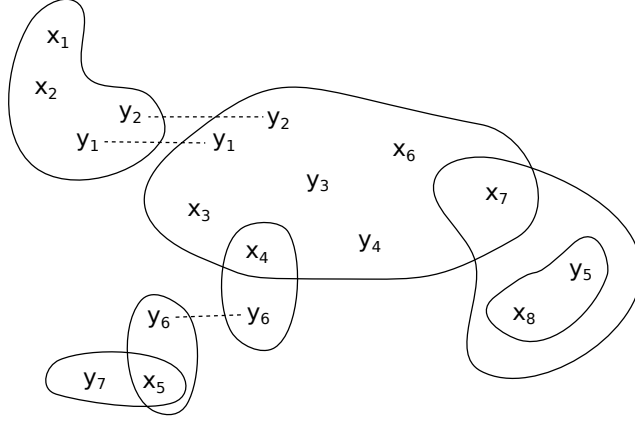[3]We state this corollary for completeness. Although we found no references, it is certainly already known

Figure 4: S-components

A subhypergraph $\mathcal{H}'$ of $\mathcal{H}$ is an S-component if there is an edge $e \in E_{\nsubseteq S}$ such that $\mathcal{H}'$ is the S-component of e.

Example 3. *The formula $\varphi(y_1, \ldots, y_7)$ of Example 1 has 3 S-components depicted in Figure 4.*

It is clear that hyperedges of $E_{\nsubseteq S}$ are partitioned into pairwise disjoint S-components.

Definition 11 (S-k-star, S-star size). *Let $\mathcal{H} = (V, E)$ be a hypergraph, $S \subseteq V$ and $k \in \mathbb{N}$. The subhypergraph $\mathcal{H}' = (V', E')$ of $\mathcal{H}$ is a S-k-star if:*

- *$\mathcal{H}'$ is an S-component of $\mathcal{H}$.*

- *there exist $y_1, ..., y_k \in V' \cap S$ such that there is no edge $e \in E$ that contains more than one of the y_i.*

We say that $y_1, \ldots, y_k$ form the S-k-star.
The S-star size of $\mathcal{H}$ is the maximum k such that there is a S-k-star in $\mathcal{H}$.

In other words, the S-star size of a hypergraph $\mathcal{H}$ is the maximal star size of its S-components.

Example 4. *Let us consider hypergraph $\mathcal{H}$ of Example 1 (and Figure 1) and $S = \{y_1, \ldots, y_7\}$. The hypergraph $\mathcal{H}$ has S-star size 3. The maximal starsize is obtained for the S-component $\mathcal{H}' = (V', E')$ with $V' = \{x_3, x_4, x_6, x_7, x_8, y_1, y_2, y_3, y_4, y_5\}$ and $E' = \{\{y_1, y_2, y_3, y_4, x_3, x_4, x_6, x_7\}, \{x_5, y_7, x_8\}, \{x_4, y_6\}\}$. This maximal S-star is formed by y_1, y_5, y_6.*

Observation 12. *Let $\mathcal{H} = (V, E)$ be a hypergraph, $S \subseteq V$ and $\mathcal{H}'$ be an S-component of $\mathcal{H}$. Then, if $\mathcal{H}$ is acyclic, $\mathcal{H}'$ is acyclic.*

16

Proof. Let $\mathcal{T}$ be a join tree of $\mathcal{H}$. An S-component $\mathcal{H}' = (V', E')$ is a subhypergraph induced by the edge set E'. By definition E' is connected in G and it follows that $\{t \mid \lambda(t) \in E'\}$ induces a subtree $\mathcal{T}'$ of $\mathcal{T}$. The connectedness condition holds in $\mathcal{T}$ and thus it holds in $\mathcal{T}'$, too. It follows that $\mathcal{T}'$ is a join tree of $\mathcal{H}'$ and $\mathcal{H}'$ is acyclic. $\qquad\square$

Definition 13. *The* quantified star size *of an acyclic conjunctive formula $\phi(\mathbf{x})$ is the S-star size of the hypergraph $\mathcal{H}$ associated to $\phi(\mathbf{x})$, where S is the set of free variables in $\phi(\mathbf{x})$.*

Example 5. *The formula from Example 7 has quantified starsize* 3.

Example 6. *The formula $\phi(x, y) \equiv \exists t \exists z R(x, y, t) \wedge S(x, z, t)$ has quantified star size* 1 *because the free variables x and y appear together in one atom.*

Path formulas (of arbitrary length), for example $\phi(x, y, z) \equiv \exists t_1 \exists t_2 R(x, t_1) \wedge R(t_1, z) \wedge R(z, t_2) \wedge R(t_2, y)$, are of quantified star size 2.

Star formulas, such as $\phi(x, y, z, t) \equiv \exists u R(u, x) \wedge R(u, y) \wedge R(u, z) \wedge R(u, t)$ have quantified star size equal to the degree of the center of the star (here 4).

Example 7. *The hard formula of [PS11] is of quantified star size n, the size of the structure domain.*

We now formulate the main results of this section. The first result is that bounding the quantified star size yields tractable counting problems.

Theorem 14. *There is an algorithm that given an acyclic conjunctive query Φ computes an arithmetic circuit C that computes $Q(\Phi)$. The runtime of the algorithm is $\|\Phi\|^{O(k)}$ where k is the quantified star size of Φ.*

Corollary 15. *There is an algorithm for the problem #ACQ that runs in time $\|\Phi\|^{O(k)}$ where k is the quantified star size of the input query Φ.*

The second result below implies that computing the quantified star size is easy and thus classes of #ACQ-instances of bounded quantified star size are efficiently decidable.

Theorem 16. *There is a polynomial time algorithm that, given a hypergraph $\mathcal{H} = (V, E)$ and $S \subseteq V$, computes the S-star size of $\mathcal{H}$.*

We prove Theorem 14 and Theorem 16 in the following two subsections.

5.2 Computation of S-star size

In this section we show that S-star size can be computed in polynomial time.

Let $\mathcal{H} = (V, E)$ be a hypergraph and $S \subseteq V$. We say that $E^* \subseteq E$ covers S if $S \subseteq \bigcup_{e \in E^*} e$. If $S = V$ we say that E^* is an edge cover of $\mathcal{H}$. An independent set I in $\mathcal{H}$ is a set $I \subseteq V$ such that there are no distinct vertices $x, y \in I$ that lie in a common edge $e \in E$.

Lemma 17. *For acyclic hypergraphs the size of a maximum independent set and a minimum edge cover coincide. Moreover, there is an algorithm that given an acyclic hypergraph $\mathcal{H}$ computes a maximum independent set I and a minimum edge cover E^* of $\mathcal{H}$.*

The first sentence in Lemma 17 can be seen as an adaptation of Kőnig's theorem for bipartite graphs (see e.g. [Bol98]) to acyclic hypergraphs. The proof uses a minimally modified version of an algorithm that Guo and Niedermeier [GN06] describe to compute minimum (unweighted) edge covers of acyclic hypergraphs. We show here that their techniques cannot only be used to compute minimum edge covers but also maximum independent sets of acyclic hypergraphs.

Proof. Clearly the size of any independent set is not greater than that of any edge cover, simply because no edge can cover two vertices in an independent set. So if we present an algorithm that computes an independent set I and an edge cover E^* of a given acyclic hypergraph $\mathcal{H} = (V, E)$ such that $|I| = |E^*|$ we are done.

So let us now describe an algorithm that computes I and E^*: Let $\mathcal{T} = (V_T, E_T)$ be a join tree of $\mathcal{H}$ with root r. We start with initially empty sets I and E^* and iteratively delete leaves of $\mathcal{T}$ in a bottom-up manner from the leaves to the root. It is easily seen that for each leaf $t \in V_T$, either $\lambda(t) \subseteq \lambda(t')$ where t' is the parent of t or there exists $y \in \lambda(t)$ such that $y \notin \lambda(t')$. In this case, we will say that y is unique for t. If $t = r$ is a leaf of $\mathcal{T}$, i.e. r is the only vertex in $\mathcal{T}$, we say by convention that if $\lambda(t)$ contains any vertices, they are all unique for t.

We do the following until V_T is empty. First, choose a leaf t of $\mathcal{T}$. If there is no vertex unique for t, we simply delete t from V_T. If there are vertices that are unique for t, choose one vertex y among them and add it to I. Furthermore, add $\lambda(t)$ to E^*, delete all vertices in $\lambda(t)$ from $\mathcal{H}$ and delete t from V_T. When V_T is empty, I and E^* are the result of the algorithm.

For a vertex $t \in V_T$ we denote by $\mathcal{T}_t$ the subtree of $\mathcal{T}$ with the root t. Let furthermore V_t be defined as the vertices in V that appear only in $\{\lambda(t^*) \mid t^* \in V(\mathcal{T}_t)\} \subseteq E$ and in no other edge in E.

Claim 2. *Whenever the algorithm deletes $t \in V_T$, the edge set E^* covers the vertices V_t.*

Proof. Assume that the claim is false, then there is a first vertex $t \in V_T$ met during the execution of the algorithm for which after t is deleted some vertex $y \in V_t$ is not covered by E^*. For all children t^* of t the vertices in V_{t^*} are covered by E^*, so y must lie in $\lambda(t)$. But then y is unique for t before t is deleted. Thus $\lambda(t)$ is added to E^* and y is covered by $\lambda(t) \in E^*$ after t is deleted which is a contradiction. $\qquad\square$

From Claim 2 it follows directly that E^* is an edge cover at the end of the algorithm.

Claim 3. *At each point in time during the algorithm I is an independent set in $\mathcal{H}$.*

Proof. Assume the claim is wrong. Then, there is a first vertex y that is added to I such that y is adjacent to x already in I. The vertex x was added to I, so there was

$t \in V_T$ such that x was unique for t when t was considered by the algorithm. Thus x is in V_t and consequently not in $\lambda(t')$ for any vertex $t' \in V_T \setminus V(\mathcal{T}_t)$. Hence, if x and y are adjacent, there must be a vertex $t^* \in V(\mathcal{T}_t)$ such that $\{x, y\} \subseteq \lambda(t^*)$. But y is added to I after x and thus it must appear in $\lambda(t')$ for a vertex $t' \in V_T \setminus V(\mathcal{T}_t)$. Then because of the connectedness condition and the fact that $\mathcal{T}$ is a tree, y must also be in $\lambda(t)$ and thus is deleted from $\mathcal{H}$ when t is deleted. But then y cannot be added later which is a contradiction. $\square$

With Claim 2 and Claim 3 we have that at the end of the algorithm E^* is an edge cover of G and I is an independent set in G. It is easy to see, that $|E^*| = |I|$. This completes the proof. $\square$

Corollary 18. *Let $\mathcal{H} = (V, E)$ be an acyclic hypergraph and $S \subseteq V$. Then the following statements are true:*

a) *The S-star size of $\mathcal{H}$ can be computed in polynomial time.*

b) *Let $\mathcal{H}' = (V', E')$ be an S-component of $\mathcal{H}$ and let k be the S-star size of $\mathcal{H}'$. There is a polynomial time algorithm that computes an edge set $E^* \subseteq E'$ that covers $S \cap V'$ and $|E^*| = k$.*

Proof. a) Let $\mathcal{H}_1, \ldots, \mathcal{H}_m$ be the S-components of $\mathcal{H}$. By Observation 12, each $\mathcal{H}_i = (V_i, E_i)$, $i \in \{1, \ldots, m\}$, is acyclic and then by Observation 9, $\mathcal{H}_i[S]$ is acyclic too. By Lemma 17, for each $i \in \{1, \ldots, m\}$, one can determine the size of a maximum independent set I_i of $\mathcal{H}_i[S]$. But we claim that for each i the star size s_i of the S-component $\mathcal{H}_i$ and the size of the maximum independent set in $\mathcal{H}_i[S]$ coincide. Indeed, consider two vertices $x, y \in S \cap V_i$ such that there is an edge $e \in E \setminus E_i$ such that $\{x, y\} \in e$. Note that x and y are each included in at least one edge of E_i. Remark also that $(V_i, E_i \cup E[V_i \cap S]) = \mathcal{H}[V_i]$ where $E[V_i \cap S] := \{e \cap (V_i \cap S) \mid e \in E, e \cap (V_i \cap S) \neq \emptyset\}$. $\mathcal{H}[V_i]$ is acyclic by Observation 9. Let $\mathcal{T}'_i$ be a join tree of $\mathcal{H}[V_i]$. The vertices $\{t \in V(\mathcal{T}'_i) \mid \lambda(t) \in E_i\}$ are connected in $\mathcal{H}[V_i]$, so they induce a subtree $\mathcal{T}_i$ of $\mathcal{T}'_i$. But then, if $e \in E \setminus E_i$, by the connectedness condition the vertex t with $\lambda(t) = e$ must be connected to $\mathcal{T}_i$ by two different paths that enter $\mathcal{T}_i$ via two different edges (since both x and y are in distinct edges of E_i). This contradicts the fact that $\mathcal{T}'_i$ is a tree. Thus s_i is indeed the size of a maximum independent set made of S-vertices in $\mathcal{H}[V_i]$ which is the S-star size of $\mathcal{H}_i$. The S-star size of $\mathcal{H}$ is then the maximal value among $s_1, ..., s_m$ and the result follows.

b) We compute an edge cover $\tilde{E}$ of size k for $\mathcal{H}'[S]$ with Lemma 17. Then for each edge $\tilde{e} \in \tilde{E}$ one can easily find an edge $e \in E'$ with $\tilde{e} \subseteq e$. $\square$

5.3 Efficiently computing the Q-polynomial

We now have the necessary ingredients to prove Theorem 14.

Proof of Theorem 14. Let $\Phi = (\mathcal{S}, \phi)$ be an input query of quantified star size k. We will construct a quantifier free formula φ and a new structure $\mathcal{S}'$ in time $\|\Phi\|^{O(k)}$ such that $\phi(\mathcal{S}) = \varphi(\mathcal{S}')$.

Let $\mathcal{H}$ be the hypergraph of ϕ and S the set of free variables of ϕ. Let $\mathcal{H}' = (V', E')$ be an S-component of $\mathcal{H}$ and let ϕ' be the subformula of ϕ whose atomic formulas are the hyperedges of E'. The formula ϕ can then be written as a conjunction $\phi' \wedge \psi$, where the formula ψ contains all the atoms of ϕ not in ϕ'. By the definition of S-components we have $(\mathsf{var}(\phi') \backslash \mathsf{free}(\phi')) \cap \mathsf{var}(\psi) = \emptyset$. In other words, the quantified variables in ϕ' only appear in atoms of ϕ' and common variables of ϕ' and ψ are necessarily in S, i.e. they are free.

Let now $E^* \subseteq E'$ be a cover of $V' \cap S$ of size $s \leq k$ computed with Corollary 18. Let $\phi_1, \ldots, \phi_s$ be the atomic formulas associated to edges in E^*. We will compute $\phi'(\mathcal{S})$ and construct in parallel a new atomic constraint φ_1 and a new relation $\varphi_1^{\mathcal{S}}$ such that $\varphi_1^{\mathcal{S}} = \phi'(\mathcal{S})$. The set of variables $\mathsf{var}(\varphi_1)$ is $\bigcup_{i=1}^{s} \mathsf{free}(\phi_i) = \mathsf{free}(\phi')$. For each combination $t_1, \ldots, t_s$ of tuples in $\phi_1(\mathcal{S}), \ldots, \phi_s(\mathcal{S})$ we add the tuple $t := t_1 \bowtie \ldots \bowtie t_s$ to the relation $\varphi_1^{\mathcal{S}}$ if

- the tuples $t_1, \ldots, t_s$ are consistent, i.e. they coincide on shared variables,

- the ACQ instance that we get from ϕ' by fixing the variables in $\mathsf{var}(\varphi_1)$ to the values specified by the tuples $t_1, \ldots, t_s$ is satisfiable.

Observe that we have to only consider $\prod_{i=1}^{s} |\phi_i(\mathcal{S})| \leq \|\Phi\|^s \leq \|\Phi\|^k$ combinations $t_1, \ldots, t_s$ and the resulting Boolean queries can each be evaluated in time $O(|\phi'| \cdot |\mathcal{S}|)$ by Yannakakis' algorithm (see [Yan81]). Thus the construction of φ_1 and the relation $\varphi_1^{\mathcal{S}}$ can be done in time $\|\Phi\|^{O(k)}$. Let us call $\mathcal{S}'$ the union of the structure $\mathcal{S}$ and $\varphi_1^{\mathcal{S}}$. We then have a new query $\varphi_1 \wedge \psi$ such that $[\varphi_1 \wedge \psi](\mathcal{S}') = \phi(\mathcal{S})$.

Let $\mathcal{T}$ be a join tree of ϕ. We can choose a subtree $\mathcal{T}'$ of $\mathcal{T}$ such that $\mathcal{T}'$ is a join tree of $\mathcal{H}'$ by considering the induced subgraph of the vertices $\{t \mid \exists e \in E', \lambda(t) = e\}$. Also $\mathsf{free}(\phi') = V' \cap S = \mathsf{var}(\varphi_1)$ and recall that $(\mathsf{var}(\phi') \backslash \mathsf{free}(\phi')) \cap \mathsf{var}(\psi) = \emptyset$. Hence, contracting $\mathcal{T}'$ into a single node whose label is the constraint φ_1, results in a join tree of the formula $\varphi_1 \wedge \psi$. Thus this latter formula is acyclic.

We iterate this process with the S-components of the subformula ψ. When each S-component has been treated, ϕ is replaced by a quantifier free formula $\varphi = \varphi_1 \wedge \ldots \wedge \varphi_m \wedge \phi_0$ where each φ_i for $i = 1, \ldots, m$ is atomic and m is the number of S-components of ϕ. Furthermore, ϕ_0 is the conjunction of all atomic formulas of ϕ that contain only free variables. Also $\mathsf{var}(\varphi) = S$. Similarly $\mathcal{S}$ is replaced by a structure $\mathcal{S}'$ of size bounded by $\|\Phi\|^{O(k)}$ (recall that each component is treated separately) such that $\phi(\mathcal{S}) = \varphi(\mathcal{S}')$. In each iteration step the formula stays acyclic. Hence, φ is acyclic and we conclude by applying Theorem 3. $\qquad\square$

5.4 #W[1]-hardness of parameterized #ACQ

In this section we show that several parameterized versions of quantified #ACQ are not fixed parameter tractable under standard assumptions from parametrized complexity. We consider the following parameterized counting problems:

- p-star-#ACQ: counting parameterized by the quantified star size,

- p-var-#ACQ: counting parameterized by the number of free variables,

- p-#ACQ: counting parameterized by the size of the conjunctive formula.

Clearly, for every ACQ instance Φ with formula φ we have that the quantified star size is at most $|\mathsf{var}(\varphi)| \leq |\varphi|$. Thus we get from p-var-#ACQ and p-#ACQ might be easier than p-star-#ACQ. The next lemma states that – unless there is a severe collapses in parameterized complexity – all three problems are not fixed parameter tractable. This is in contrast to the decision version which is even in **P** for all three problems.

Lemma 19. *p-star-#ACQ, p-var-#ACQ and p-#ACQ are all #**W**[**1**]-hard.*

Proof. We reduce p-#DirPath, i.e. counting of paths of length k, to #ACQ on stars. With the #**W**[**1**]-hardness of p-#DirPath [FG04] the result will follow. The basic observation is that there are $|E|^k$ ordered choices of k edges with repetitions. Thus it suffices to count the number of choices that are not paths to compute the number of k-paths in a graph. A choice $e_1, \ldots, e_k$ is not a k-path, if and only if it has one of the following defects:

1. It is not a walk, i.e. there is an i such that e_i has as end vertex not the start vertex of e_{i+1}, or

2. a vertex is visited twice.

We will encode these properties into a #ACQ-instance of polynomial size whose hypergraph is a k-star.

So let $G = (V, E)$ be the input in which we are supposed to count k-paths. We construct a #ACQ-instance Φ which has the variables $y_1, \ldots, y_k, z$. The y_i have the domain $V \times V$, while z has the more complicated domain $\{0, 1\} \times [k] \times V \times V \times [k] \times [k] \times V$. Observe that the domains have polynomial size and all constraints will be binary, so Φ has polynomial size in n and k.

For each $i \in [k]$ we add a binary constraint E_i in the variables y_i, z. Out of these we build the formula

$$\varphi := \exists z \bigwedge_{i=1}^{k} E_i(y_i, z).$$

The y_i-variable will choose arbitrary potential start and end points of an edge. The role of the z-variable is to guess one of the defects described above that prevents the chosen vertices from describing a path. We systematically describe tuple set for E_i. The first component of an assignment can only take 0 or 1 and encodes if z guesses either a defect in the walk structure or a double variable.

- z may guess that the end vertex of e_i is v while the start vertex of e_{i+1} is u for $u \neq v$. It does so by taking the value $(0, i, v, u, j_1, j_2, a)$ for arbitrary j_1, j_2, a. If this guess is true, then y_i must have chosen an edge that indeed does end in v. So we add the tuples $\{((b, v), (0, i, v, u, j_1, j_2, a)) \mid a, b, v, u \in V, (b, v) \in E, j_1, j_2 \in [k]\}$.

- The second defect that z may guess is that the edges do not form a path, because the end vertex of e_{i-1} is v while the start vertex of e_i is u. This results in the tuples $\{((u,b),(0,i-1,v,u,j_1,j_2,a)) \mid a,b,v,u \in V, (b,v) \in E, j_1,j_2 \in [k]\}$.

- If z predicts a defect preventing a walk in some other place, E_i does not have to check this, so we accept everything. The resulting tuples are $\{((u,v),(0,j,c,d,j_1,j_2,a)) \mid a,c,d,v,u \in V, (u,v) \in E, j_1,j_2 \in [k], j \notin \{i-1,i\}\}$. These cover all cases of the edges not being a path.

- If z guesses that the vertex v occurs at two different places in the potential path, it does so by specifying edges e_{j_1}, e_{j_2} with $j_1 \leq j_2$ such that the start vertex of e_{j_1} and the end vertex of e_{j_2} is v. It does so by taking a value $(1,\ell,a,b,j_1,j_2,v)$ for ℓ,a,b arbitrary. If $i \notin \{j_1,j_2\}$ then E_i does not have to check for an effect and accepts if y_i encodes an edge. Thus we add the tuples $\{((u,w),(1,\ell,a,b,j_1,j_2,v) \mid a,b,u,v,w \in V, (u,w) \in E, \ell,j_1,j_2 \in [k], i \notin \{j_1,j_2\}\}$.

- If z guesses a double occurence of v and $i = j_1$ we accept only if that guess is correct and $j_2 \geq i$. So we add the tuples $\{((v,w),(1,\ell,a,b,i,j_2,v) \mid a,b,v,w \in V, (v,w) \in E, \ell,j_2 \in [k]\}$. If $i = j_2$ we add analogous tuples.

It is easy to see that Φ accepts assignments to the y_i if and only if each y_i gets the end points of an edge in G and there is a defect that prevents the edges from being a path. Thus the number of satisfying assignments of Φ is the number of ordered choices of edges in G with repetition that are not paths. This completes the proof. $\square$

5.5 Bounded quantified star size is necessary

In this section we show that quantified star size is in a sense the only restriction that makes #ACQ tractable. Not only does bounded quantified star size give tractable instances, but the other way round under a standard assumption from parameterized complexity all classes of tractable #ACQ-instances must have bounded quantified star size. This is somewhat similar to the results of Grohe et al [GSS01] who proved that under reasonable assumptions the only polynomial time decidable subclass of bounded arity CQ is the class of bounded treewidth.

As we have seen in the previous sections, not only the hypergraph of the input formula is decisive for tractability but also the structure of the quantified variables in this hypergraph. We formalize this in the following definition.

Definition 20. *An S-hypergraph is a pair $(\mathcal{H}, S)$ where $\mathcal{H} = (V, E)$ is a hypergraph and $S \subseteq V$. We say that #ACQ is tractable for a class $\mathcal{G}$ of S-hypergraphs if for all #ACQ instances Φ with the associated hypergraph $\mathcal{H}$ of Φ and the set S of free variables of Φ with $(\mathcal{H}, S) \in \mathcal{G}$ we can solve #ACQ in polynomial time.*

Example 8. *Let $\mathcal{G}$ be the class of acyclic S-hypergraphs of S-star size bounded by k. Then the result of Theorem 14 can be expressed as "#ACQ is tractable for $\mathcal{G}$".*

We will use the fact that #ACQ is already hard for very restricted S-hypergraphs, namely for stars in which only the center is not in S. We call this class $\mathcal{G}_S$. Observe that the proof of Lemma 19 gives the following Lemma.

Lemma 21. *#ACQ is #**W**[**1**]-hard for $\mathcal{G}_S$ parameterized by the size of the stars.*

We now show the main result of this section.

Theorem 22. *Assume* **FPT** $\neq$ #**W**[**1**], *and let $\mathcal{G}$ be a recursively enumerable class of acyclic S-hypergraphs. Then #ACQ is polynomial time solvable for $\mathcal{G}$ if and only if $\mathcal{G}$ is of bounded S-star size.*

Proof. One direction of the claim is Theorem 14. For the other direction assume that there is a class $\mathcal{G}$ of unbounded S-star size such that #ACQ is tractable on $\mathcal{G}$. We show that in this case #ACQ on $\mathcal{G}_S$ parameterized by the star size is in **FPT** and with Lemma 21 we get **FPT** = #**W**[**1**].

So all we have to do is to construct a fixed parameter algorithm for #ACQ on $\mathcal{G}_S$. Let Φ be an instance of this problem, i.e. Φ has the formula $\varphi := \exists z \bigwedge_{i=1}^{k} E_i(y_i, z)$. Let the domain of Φ be D. Because $\mathcal{G}$ is recursively enumerable and of unbounded S-star size, there is a computable function $g : \mathbb{N} \to \mathbb{N}$ such that for $k \in \mathbb{N}$ one can compute $(\mathcal{H}, S) \in \mathcal{G}$ such that $\mathcal{H}$ is of S-star size at least k in time $g(k)$. We will embed Φ into $\mathcal{H}$ to construct an #ACQ-instance Ψ of size $g(k)n^{O(1)}$ where n is the size of Φ. Furthermore, Ψ will have the S-hypergraph $\mathcal{H}$ and the same domain D as Φ.

Let $\mathcal{H}' = (V', E')$ be an S-k-star in $\mathcal{H}$ that is formed by $Y = \{y_1, \ldots, y_k\}$, $Y \subseteq S$. For each edge $e \in E$ we define a constraint E_e. Let first $e \in E'$ be an edge that contains y_i for some $i \in [k]$, then E_e has as variables the vertices of e. Let y_i be the first variable of E_e followed by the other variables in $e \cap S$ and after those the variables in $e \setminus S$. Then E_e has the tuples $\{(a, d, \ldots, d, b, \ldots, b) \mid (a, b) \in \mathcal{R}_i\}$, where $\mathcal{R}_i$ is the relation of E_i and d is an arbitrary but fixed value in D. Observe that this forces all variables in $(e \cap S) \setminus \{y_i\}$ to the variable d in satisfying assignments, while the variables in $e \setminus S$ all have a common value b. Furthermore, observe that no two of the y_i share an edge in E', so E_e is always well defined.

Let $e \in E'$ with $e \cap Y = \emptyset$. Again we define a constraint E_e. Let in E_e the first variables be those in $S \cap e$ followed by those in $e \setminus S$, then E_e has the tuples $\{(d, \ldots, d, a, \ldots, a) \mid a \in D\}$ for the same $d \in D$ as before. Again in the satisfying assignments all variables in $e \cap S$ are forced to d, while the variables in $e \setminus S$ can take an arbitrary but equal value.

For $e \in E \setminus E'$ with $e \cap V' = \emptyset$, we add a constraint E_e in with the single tuple $(d, \ldots, d)$. If $e \in E \setminus E'$ with $e \cap V' \neq \emptyset$, we have $e \cap V' \subseteq S$. Furthermore at most one vertex in $e \cap V'$, say y_i, can be in Y, because Y forms a star. If there is no such y_i, we construct a constraint E_e with the only tuple $(d, \ldots, d)$. If there is $y_i \in e$, we construct a constraint E_e in which y_i is the first variable with the tuples $\{(a, d, \ldots, d) \mid a \in D\}$.

These are all constraints of Ψ. Let the formula

$$\psi' := \bigwedge_{e \in E} E_e$$

and ψ the formula that we get from ψ' by quantifying all variables in $V \setminus S$. Let ψ be the formula of Ψ, then it is easy to see that Ψ has has the associated S-hypergraph $(\mathcal{H}, S)$. Furthermore, Ψ has the same number of satisfying assignments as Φ. This is because in each satisfying assignment $\mathbf{a}$ of Ψ all variables in $(V \setminus V') \cup (S \setminus Y)$ are set to d. Furthermore, all variables in $V' \setminus S$ take one common value b in $\mathbf{a}$. Let $\mathbf{a}'$ be an assignment to Φ that we get by setting $\mathbf{a}'(z) := b$ and $\mathbf{a}'(y_i) := \mathbf{a}(y_i)$. It is easy to see that by this construction the satisfying assignments of Ψ and Φ correspond directly, so the number of satisfying assignments is the same.

Now assume that #ACQ is polynomial time solvable for $\mathcal{G}$. It follows that the satisfying assignments of Ψ can be counted in time $(g(k)n)^{O(1)}$ and thus #ACQ on $\mathcal{G}_S$ is in **FPT**. With Lemma 21 this contradicts the assumption which completes the proof. $\qquad\square$

6 Applications to arithmetic circuit complexity

We now show an adaptation of some of the results in this paper to arithmetic circuit complexity, i.e. the so-called Valiant model [Val79]. A *polynomial family* is a sequence (f_n) of multivariate polynomials over a field $\mathbb{F}$. The class of polynomial families of polynomial degree computed by families of polynomial size arithmetic circuits is denoted by **VP**. This class is a natural candidate to formalize efficient computation with arithmetic circuits. A family (f_n) of polynomials is in **VNP**, if there is a family $(g_n) \in$ **VP** and a polynomial p such that $f_n(X) = \sum_{e \in \{0,1\}^{p(n)}} g_n(e, X)$ for all n where X denotes the vector $(X_1, \ldots, X_{q(n)})$ for some polynomial q. By definition **VP** $\subseteq$ **VNP** but the precise relations between **VP** and **VNP** are still unknown. It is however widely conjectured that **VP** $\neq$ **VNP**, i.e. that not all polynomial familes in **VP** are efficiently computable by arithmetic circuits.

A polynomial f is called a *projection* of g (symbol: $f \leq g$), if there are values $a_i \in \mathbb{F} \cup \{X_1, X_2, \ldots\}$ such that $f(X) = g(a_1, \ldots, a_q)$. A family (f_n) of polynomials is a p-projection of (g_n) (symbol: $(f_n) \leq_p (g_n)$), if there is a polynomial r such that $f_n \leq g_{r(n)}$ for all n. As usual we say that (g_n) is hard for an arithmetic circuit class $\mathcal{C}$ if for every $(f_n) \in \mathcal{C}$ we have $(f_n) \leq_p (g_n)$. If further $(g_n) \in \mathcal{C}$ we say that (g_n) is $\mathcal{C}$-complete.

Classes of polynomials are often better understood through the natural polynomial families they contain or, better, which are complete for them. For example, it is well-known that determinant family, denoted (Det_n), of matrices $(X_{i,j})_{i,j \in [n]}$ defined by:

$$\mathrm{Det}_n(X_{i,j}) = \sum_{\sigma \in S_n} \mathrm{sgn}(\sigma) \prod_{i=1}^{n} X_{i,\sigma(i)}.$$

where $\mathrm{sgn}(\sigma) \in \{-1, 1\}$ is the sign of the permutation σ, is contained in **VP** (although presumably not complete). Similarly, the polynomial family (Perm_n) representing the permanent of the matrices $(X_{i,j})_{i,j \in [n]}$ and defined as

$$\mathrm{Perm}_n(X_{i,j}) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} X_{i,\sigma(i)}.$$

has been shown to be **VNP**-complete (see [Val81]). Recently, the complexity of polynomial families defined by natural extensions of counting solutions to constraint satisfaction problems (like the the Q-polynomial of this paper) has been investigated (see [BKM11, Men11] and also [Bri11]) and several non-circuit characterizations of **VP**, **VNP** and other classes have been obtained. In this section, we generalize some of these results by showing that acyclic conjunctive queries can define polynomial families that characterizes **VP** and **VNP**.

6.1 Tractable polynomials

The weighted counting problem $\#_{\mathbb{F}}\mathrm{ACQ}$ can be seen as the problem of computing the polynomial

$$Q(\Phi) := \sum_{a \in \phi(\mathcal{S})} \prod_{x \in \mathsf{var}(\phi)} X_{a(x)},$$

in the variables $\{X_d \mid d \in D\}$ for a Φ in CQ. This naturally makes $\#_{\mathbb{F}}\mathrm{ACQ}$ or equivalently computing $Q(\Phi)$ a question in the Valiant model. We have:

Theorem 23. *If (Φ_n) is a family of* ACQ *of polynomially bounded size and bounded quantified star size, then $(Q(\Phi_n)) \in$* **VP**. *Moreover, any family in* **VP** *is a p-projection of $(Q(\Phi_n))$, where the Φ_n are polynomial size quantifier free conjunctive queries whose hypergraph is a tree.*

Proof. The upper bound follows from the proof of Theorem 14 and Theorem 3. The lower bound is already true for acyclic queries on graphs and follows from [Men11]. □

This result shows that while $\#_{\mathbb{F}}\mathrm{ACQ}$ is a tractable counting problem, it is probably harder than computing the determinant, which is quite rare in counting complexity (see [MP08] for the role of the determinant in the Valiant model).

As a corollary, we also obtain the following result from [Men11].

Corollary 24. *Let k, d be integers. Let (Φ_n) be a family of* CSP *of polynomially bounded size with arity bounded by d and tree-width bounded by k. Then $(Q(\Phi_n)) \in$* **VP**.

Proof. Comes from the fact that any CSP of tree-width k built on relation of arity d can be transformed (by taking joins of atoms in each bag of the tree decompositions) in time $O(n^{f(k,d)})$, for some function f, into an acyclic CSP with the same set of solutions. □

6.2 The power of existentially quantified variables

In this section we show a version of Pichler and Skritek's hardness result for #ACQ for the Valiant model, i.e. the polynomial $Q(\Phi)$ appears to be harder to compute than in the unquantified case. We state the upper bound in a more general way:

Let D be a finite set with $|D| = d$. For a positive integer n we encode an elements of $\mathbf{a} = (a_1, \ldots, a_n) \in D^n$ by a $d \times n$-matrix $M = (m_{i,j})_{i \in D, j \in [n]}$ such that $m_{i,j} = 1$ if $a_j = i$ and $m_{i,j} = 0$ otherwise. Observe that a 0-1-matrix encodes an element in D^n if and only if in each column there is exactly one 1. We define the monomial $q(M) := q(a) = \prod_{j \in [n]} X_{a_j}$

if M encodes a and $q(M) := 0$ otherwise. With this notation we can prove a version of Valiant's classic criterion [Val79] for the Q-polynomial.

Proposition 25. *Let $\alpha : \{0,1\}^* \to \mathbb{Z}$ a function in **gapP**. Furthermore let (D_n) be a polynomially bounded family of sets and let $p(n)$ be a polynomial. Then the family (f_n) defined by*

$$f_n = \sum_{M \in \{0,1\}^{|D_n| \times p(n)}} \alpha(M) q(M)$$

*is in **VNP**.*

Proof. It is folklore that α can be computed as $\alpha(x) = \sum_{e \in \{0,1\}^{r(n)}} g_{|x|}(x,e)$ where r is a polynomial and g_n is a family of uniform arithmetic formulas of polynomial size (see e.g. [BF91]). In the proof of Lemma 4 in the full version of [Men11] it is shown how to compute the function q with small arithmetic formulas. Combining this we directly get the proposition. $\square$

Proposition 26. *If (Φ_n) is a family of acyclic conjunctive queries of polynomial size, then $(Q(\Phi_n)) \in **VNP**$. Moreover, any family in **VNP** is a p-projection of such a $(Q(\Phi_n))$. The (Φ_n) family can be supposed of arity bounded by two.*

Proof. We start off with the containment in **VNP**, which follows easily from the fact that given an assignment a and an acyclic conjunctive query Φ one can in polynomial time decide if the a satisfies Φ. Applying Proposition 25 we get the upper bound.

The hardness is obtained by reduction from the family (Perm_n). Perm_n can be seen as the sum of weights of perfect matchings in the weighted bipartite graph $K_{n,n}$.

Let $K_{n,n} = (A \cup B, E)$ be the complete bipartite graph with $A = \{a_1, ..., a_n\}$, $B = \{b_1, ..., b_n\}$. We denote by $e_{i,j}$ the edge between vertices $a_i \in A$ and $b_j \in B$. We construct a structure $S = \langle \mathcal{U}, F, G, H, I, J, K, a_1, ..., a_n \rangle$ of domain $\mathcal{U}$ as follows.

- The universe $\mathcal{U} = A \cup B \cup \{e_{i,j} \mid i,j \in [n]\} \cup \{p, n, l\}$,

- $G = \{p, n\}$,

- $F = \{(n,l)\} \cup \{(p, e_{i,j}) \mid i,j \in [n]\}$,

- $H = \{(e_{i,j}, a_i) \mid i,j \in [n]\} \cup \{(l, a_i) \mid i \in [n]\}$,

- $I = \{(p,l)\} \cup \{(n, b_i) \mid i \in [n]\}$,

- $K = \{(b_j, e_{i,k}) : i,j,k \in [n], j \neq k\} \cup \{(l,l)\}$

Note that the maximal arity of a predicate is two. Let $\phi(x_1, ..., x_n, x, x'_1, ..., x'_n)$ be the following acyclic conjunctive query:

$$G(x) \wedge \bigwedge_{i=1}^{n} F(x, x_i) \wedge \bigwedge_{i=1}^{n} H(x_i, a_i) \wedge \exists y I(x, y) \wedge \bigwedge_{i=1}^{n} K(y, x'_i) \wedge \bigwedge_{i=1}^{n} H(x'_i, a_i) \qquad (1)$$

There are two types of satisfying assignments:

- If x takes the value p, then y and all x_i' must take the value l. The x_i take as values the edges $e_{i,j}$ in such a way that for each $i \in [n]$ there is an edges $e_{i,j}$. Thus the vertices in A are mapped to the vertices of B in the original graph in an arbitrary way.

- If x takes the value n, then all x_i must take the value l. Furthermore y takes a value $b \in B$. The x_i' then take as values the edges $e_{i,j}$ in such a way that each a_i is mapped to a vertex $b_j \in B \setminus \{b\}$ by this edge. Thus the assignment to the x_i' is an arbitrary non-injective assignment of the vertices in A to those in B.

Thus the query $\Phi = (\mathcal{S}, \phi)$ defines a polynomial $Q(\Phi)$ with the following property (with $e_{i,j}$ corresponding to variable $X_{i,j}$, p to X_p, n to X_n and l to X_l).

$$
\begin{aligned}
Q(\Phi) \;\; &= \sum_{\phi(S)} \prod_{t \in \mathsf{var}(\Phi)} X_{a(t)} \\
&= X_p X_l^n \sum_{\sigma \in [n]^{[n]}} \prod_{i=1}^{n} X_{i,\sigma(i)} + X_n X_l^n \sum_{\sigma \in [n]^{[n]} \setminus S_n} \prod_{i=1}^{n} X_{i,\sigma(i)}.
\end{aligned}
$$

Projecting correctly, we get

$$
\mathrm{Perm}_n(X_{i,j}) = Q(\Phi)(X_1, ..., X_n, X_p, X_n, X_l)|_{X_p=1, X_n=-1, X_l=1}.
$$

$\square$

6.3 Unions and intersections of ACQ

We now show that a version of Proposition 5 is also true for the Valiant model.

Proposition 27. *If* (Φ_n) *is a family of queries of polynomial size that are conjunction (resp. disjunction) of two acyclic conjunctive queries, then* $(Q(\Phi_n)) \in \mathbf{VNP}$. *Moreover, any family in* $\mathbf{VNP}$ *is a p-projection of such a* $(Q(\Phi_n))$. *The result remains true for the polynomial family* $(P(\Phi_n))$.

Proof (Sketch). The upper bound follows directly from Proposition 25. The proof of the lower bound for conjunction of acyclic queries follows directly like Proposition 5. The case of disjunction is obtained by reduction from the case of conjunction. Let $\Phi = (\mathcal{S}, \phi(\mathbf{x}))$ and $\Psi = (\mathcal{S}, \psi(\mathbf{x}))$ be two acyclic conjunctive queries. W.l.o.g. we can suppose they both are on the same structure $\mathcal{S}$ of signature σ and domain D. We denote by $\Phi \wedge \Psi$ the instance $(\mathcal{S}, \phi(\mathbf{x}) \wedge \psi(\mathbf{x}))$ and by $\Phi \vee \Psi$ the instance $(\mathcal{S}, \phi(\mathbf{x}) \vee \psi(\mathbf{x}))$. Let $\mathcal{S}'$ be a new structure of domain $D' = D \cup \{\alpha_1, \alpha_2, \alpha_3\}$ where $\alpha_1, \alpha_2, \alpha_3$ are not in D. Structure $\mathcal{S}'$ includes $\mathcal{S}$ and is equipped with two new unary relations R and S which are defined as follows:

$$
R = \{\alpha_1, \alpha_2\}, S = \{\alpha_2, \alpha_3\}.
$$

Let us now consider the following disjunction of two acyclic formulas:

$$\varphi(\mathbf{x}, y) \equiv (\phi(\mathbf{x}) \wedge R(y)) \vee (\psi(\mathbf{x}) \wedge S(y)).$$

The query problem $\Upsilon = (\mathcal{S}', \varphi(\mathbf{x}, y))$ has the following tuples as solutions:

- $(\mathbf{a}, \alpha_1)$ for $\mathbf{a} \in \phi(\mathcal{S})$.

- $(\mathbf{a}, \alpha_3)$ for $\mathbf{a} \in \psi(\mathcal{S})$.

- $(\mathbf{a}, \alpha_2)$ for $\mathbf{a} \in \phi(\mathcal{S}) \cup \psi(\mathcal{S})$.

Then, associating each value α_i with variable Y_i:

$$Q(\Upsilon) \quad = Y_1 Q(\Phi) + Y_3 Q(\Psi) + Y_2 Q(\Phi \vee \Psi).$$

By projection, we get

$$Q(\Phi \wedge \Psi) = Q(\Upsilon)(\mathbf{X}, Y_1, Y_2, Y_3)|_{Y_1=1, Y_2=-1, Y_3=1}.$$

This shows that polynomials obtained by disjunction of two acyclic queries can represented as projections of polynomials obtained by conjunction and hence this is true for all polynomial families in **VNP**. $\qquad\square$

7 Conclusion

We have presented a complete picture of tractability for weighted #ACQ. However, there are still many open questions that could be explored in the future.

It seems clear that the notion of quantified star size may be adapted to other classes of queries. For example, it is easily seen that one can characterize tractability for counting queries for larger classes and that Theorem 16 can be extended for all queries of bounded generalized hypertreewidth [GLS02, Mik08]. However, in these cases, deciding whether a conjunctive query belonging to some nicely decomposable class is of bounded quantified star size, may be difficult. We feel that the complexity of this so-called "discovery" problem for different class of queries deserves to be studied and address this problem in an upcoming paper.

Another challenging problem is whether one can adapt the notion of star size to quantified i.e. $\{\exists, \forall, \wedge\}$-formulas and characterize tractability of decision and counting for such queries. Also, an interesting application concerns aggregate queries, i.e. queries in which the language may be enriched by predicates or operators for counting, summing, averaging... It seems that the star size approach could be helpful for the design of tractable query language in this setting.

Finally, one motivation for this work was to exhibit a simple characterization of polynomial families belonging to **VP**. Classical families such as the determinant or the permanent are complete for classes respectively below and above **VP** and till now, it is open whether there exists a natural complete family for **VP**. We believe that our approach may help in understanding the structure of **VP**.

References

[BCC+05] M. Bauland, P. Chapdelaine, N. Creignou, M. Hermann, and H. Vollmer. An algebraic approach to the complexity of generalized conjunctive queries. In *Theory and Applications of Satisfiability Testing*, pages 30–45. Springer, 2005.

[BDG07] G. Bagan, A. Durand, and G. Grandjean. On acyclic conjunctive queries and constant delay enumeration. In *Computer Science Logic, 16th Annual Conference of the EACSL*, volume 4646 of *LNCS*, pages 208–222. Springer, 2007.

[BDG+10] A. A. Bulatov, M.E. Dyer, L.A. Goldberg, M. Jalsenius, M. Jerrum, and D. Richerby. The complexity of weighted and unweighted #CSP. *CoRR*, abs/1005.2678v2, 2010.

[BF91] L. Babai and L. Fortnow. Arithmetization: A new method in structural complexity theory. *Computational Complexity*, 1(1):41–66, 1991.

[BKM11] I. Briquel, P. Koiran, and K. Meer. On the expressive power of CNF formulas of bounded tree- and clique-width. *Discrete Applied Mathematics*, 159(1):1–14, 2011.

[Bol98] B. Bollobas. *Modern graph theory*. Springer Verlag, 1998.

[Bri11] I. Briquel. *Complexity issues in counting, polynomial evaluation and zero finding*. PhD thesis, ENS Lyon - City University of Hong Kong, 2011.

[Bul08] Andrei A. Bulatov. The complexity of the counting constraint satisfaction problem. In *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland*, volume 5125 of *Lecture Notes in Computer Science*, pages 646–661. Springer, 2008.

[CCL11] Jin-Yi Cai, Xi Chen, and Pinyan Lu. Non-negatively weighted #csp: An effective complexity dichotomy. In *Proceedings of the 26th Annual IEEE Conference on Computational Complexity, CCC 2011, San Jose, California*, pages 45–54, 2011.

[CJG08] David Cohen, Peter Jeavons, and Marc Gyssens. A unified theory of structural tractability for constraint satisfaction problems. *Journal of Computer and System Sciences*, 74(5):721 – 743, 2008.

[CM77] A.K. Chandra and P.M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *9h annual ACM STOC*, pages 77–90. ACM, 1977.

[DGJ09] M. E. Dyer, L. A. Goldberg, and M. Jerrum. The Complexity of Weighted Boolean #CSP. *SIAM J. Comput.*, 38(5):1970–1986, 2009.

[DHK05] A. Durand, M. Hermann, and P.G. Kolaitis. Subtractive reductions and complete problems for counting complexity classes. *Theoretical Computer Science*, 340(3):496–513, 2005.

[DJ04] Víctor Dalmau and Peter Jonsson. The complexity of counting homomorphisms seen from the other side. *Theor. Comput. Sci.*, 329(1-3):315–323, 2004.

[DR10] M. E. Dyer and D. Richerby. On the complexity of #CSP. In *42nd ACM STOC, Cambridge, Massachusetts, USA*, pages 725–734, 2010.

[Fag83] R. Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *Journal of the ACM*, 30(3):514–550, 1983.

[FG04] J. Flum and M. Grohe. The parameterized complexity of counting problems. *SIAM Journal on Computing*, 33(4):892–922, 2004.

[FG06] J. Flum and M. Grohe. *Parameterized complexity theory*. Springer-Verlag New York Inc, 2006.

[FMR08] E. Fischer, J.A. Makowsky, and E.V. Ravve. Counting truth assignments of formulas of bounded tree-width or clique-width. *Discrete Applied Mathematics*, 156(4):511–529, 2008.

[GLS01] G. Gottlob, N. Leone, and F. Scarcello. The complexity of acyclic conjunctive queries. *J. ACM*, 48(3):431–498, 2001.

[GLS02] G. Gottlob, N. Leone, and F. Scarcello. Hypertree decompositions and tractable queries. *Journal of Computer and System Sciences*, 64(3):579–627, 2002.

[GN06] J. Guo and R. Niedermeier. Exact algorithms and applications for tree-like weighted set cover. *J. Discrete Algorithms*, 4(4):608–622, 2006.

[Gro07] Martin Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *Journal of the ACM*, 54(1), 2007.

[GSS01] M. Grohe, T. Schwentick, and L. Segoufin. When is the evaluation of conjunctive queries tractable? In *33rd ACM STOC*, pages 657–666. ACM, 2001.

[KPZ99] A. Kiayias, A. Pagourtzis, and S. Zachos. Cook reductions blur structural differences between functional complexity classes. In *Panhellenic Logic Symposium*, pages 132–137, 1999.

[Lib04] L. Libkin. *Elements of finite model theory*. EATCS Series. Springer, 2004.

[Men11] S. Mengel. Characterizing Arithmetic Circuit Classes by Constraint Satisfaction Problems. In *Automata, Languages and Programming - 38th International Colloquium, ICALP 2011*, volume 6755 of *LNCS*, pages 700–711. Springer, 2011.

[Mik08] Z. Miklós. *Understanding Tractable Decompositions for Constraint Satisfaction*. PhD thesis, University of Oxford, 2008.

[MP08] G. Malod and N. Portier. Characterizing Valiant's algebraic complexity classes. *J. Complexity*, 24(1):16–38, 2008.

[PS11] R. Pichler and A. Skritek. Tractable Counting of the Answers to Conjunctive Queries. In *AMW*, 2011.

[Thu06] M. Thurley. Tractability and Intractability of Parameterized Counting Problems. Diploma thesis, Humbold-Universiät zu Berlin, 2006.

[Val79] L. G. Valiant. Completeness classes in algebra. In *11th ACM STOC*, pages 249–261, New York, NY, USA, 1979. ACM.

[Val81] L. G. Valiant. Universality considerations in VLSI circuits. *IEEE Transactions on Computers*, 30(2):135–140, 1981.

[Yan81] M. Yannakakis. Algorithms for acyclic database schemes. In *Proceedings of the 7th International Conference on Very Large Databases*, pages 82–94, 1981.