Logical Methods in Computer Science
Volume 21, Issue 1, 2025, pp. 9:1–9:35
https://lmcs.episciences.org/

Submitted Nov. 01, 2023
Published Jan. 28, 2025

DIVERSITY OF ANSWERS TO CONJUNCTIVE QUERIES

TIMO CAMILLO MERKL [a], REINHARD PICHLER [a,b], AND SEBASTIAN SKRITEK [a]

TU Wien, Vienna, Austria
e-mail address: timo.merkl@tuwien.ac.at, reinhard.pichler@tuwien.ac.at, sebastian.skritek@tuwien.ac.at

Abstract. Enumeration problems aim at outputting, without repetition, the set of solutions to a given problem instance. However, outputting the entire solution set may be prohibitively expensive if it is too big. In this case, outputting a small, sufficiently diverse subset of the solutions would be preferable. This leads to the Diverse-version of the original enumeration problem, where the goal is to achieve a certain level d of diversity by selecting k solutions. In this paper, we look at the Diverse-version of the query answering problem for Conjunctive Queries and extensions thereof. That is, we study the problem if it is possible to achieve a certain level d of diversity by selecting k answers to the given query and, in the positive case, to actually compute such k answers.

1. Introduction

The notion of *solutions* is ubiquitous in Computer Science and there are many ways of defining computational problems to deal with them. Decision problems, for instance, may ask if the set of solutions is non-empty or test for a given candidate if it indeed is a solution. Search problems aim at finding a concrete solution and counting problems aim at determining the number of solutions. In recent time, *enumeration problems*, which aim at outputting, without repetition, the set of solutions to a given problem instance have gained a lot of interest, which is, for instance, witnessed by two recent Dagstuhl seminars on this topic [BKPS19, FGS18]. Also in the Database Theory community, enumeration problems have played a prominent role on the research agenda recently, see e.g., [AJMR22, KKW22, LP22, DST23, MR23, CS23]. Here, the natural problem to consider is query answering with the answers to a given query constituting the "solutions" to this problem.

It is well known that even seemingly simple problems, such as answering an acyclic Conjunctive Query, can have a huge number of solutions. Consequently, specific notions of tractability were introduced right from the beginning of research on enumeration problems [JPY88] to separate the computational intricacy of a problem from the mere size of the solution space. However, even with these refined notions of tractability, the usefulness of flooding the user with tons of solutions (many of them possibly differing only minimally)

Key words and phrases: Query Answering, Diversity of Solutions, Complexity, Algorithms.
* This is an extended and enhanced version of an article published at ICDT 2023 [MPS23].
* This work was supported by the Austrian Science Fund (FWF) project P30930-N35 and by the Vienna Science and Technology Fund (WWTF) [10.47379/ICT2201].

DOI:10.46298/LMCS-21(1:9)2025

may be questionable. If the solution space gets too big, it would be more useful to provide an overview by outputting a "meaningful" subset of the solutions. One way of pursuing this goal is to randomly select solutions (also known as "sampling") as was, for instance, done in [ACJR19, CZB+22]. In fact, research on sampling has a long tradition in the Database community [CM99] – above all with the goal of supporting more accurate cardinality estimations [LRG+17, LWYZ19, ZCL+18].

A different approach to providing a "meaningful" subset of the solution space aims at outputting a small *diverse* subset of the solutions. This approach has enjoyed considerable popularity in the Artificial Intelligence community [BFJ+22, EEEF13, Nad11] – especially when dealing with Constraint Satisfaction Problems (CSPs) [HHOW05, IdlBST20, PT15]. For instance, consider a variation of the car dealership example from [HHOW05]. Suppose that I models the preferences of a customer and $\mathcal{S}(I)$ are all cars that match these restrictions. Now, in a large dealership, presenting all cars in $\mathcal{S}(I)$ to the customer would be infeasible. Instead, it would be better to go through a rather small list of cars that are significantly different from each other. With this, the customer can point at those cars which the further discussion with the clerk should concentrate on.

Due to the inherent hardness of achieving the maximal possible diversity [HHOW05], the Database community – apart from limited exceptions [DF14] – focused on heuristic and approximation methods to find diverse solutions (see [ZWQ+17] for an extensive survey). Also, there, diversification is usually treated as a post-processing task that is applied to a set of solutions after materializing it.

The goal of our work is therefore to broaden the understanding of the theoretical boundaries of diverse query answering and develop complementary exact algorithms. More specifically, we want to analyze diversity problems related to answering Conjunctive Queries (CQs) and extensions thereof. As pointed out in [IdlBST20], to formalize the problems we are thus studying, we, first of all, have to fix a notion of *distance* between any two solutions and an *aggregator* to combine pairwise distances to a *diversity measure* for a set of solutions. For the distance between two answer tuples, we will use the Hamming distance throughout this paper, that is, counting the number of positions on which two tuples differ. Our developed techniques naturally extend to weighted Hamming distances, i.e., where each attribute is assigned a constant weight and we sum over the weights of those attributes the two answers differ on. However, for the sake of simplicity, we stick to the unweighted version in our discussion. As far as the choice of an aggregator f is concerned, we impose the general restriction that it must be computable in polynomial time[1]. As will be detailed below, we will sometimes also consider more restricted cases of aggregators. Formally, for a class $\mathcal{Q}$ of queries and diversity measure δ that maps k answer tuples to an aggregated distance, we will study the following problem Diverse-$\mathcal{Q}$ (δ is fixed and not part of the input):

Diverse-$\mathcal{Q}$
Input: A database instance I, query $Q \in \mathcal{Q}$, and integers k and d.
Question: Do there exist pairwise distinct answers $\gamma_1, \ldots, \gamma_k$ to Q over I such that $\delta(\gamma_1, \ldots, \gamma_k) \geq d$?

[1]The reason for this restriction is simply to not dilute our discussion of the complexity of the diversity problem with the inherent complexity of computing the diversity of a set of answers itself. If we are content with higher complexity classes, like FPT, the time required to compute the aggregator may also take longer, e.g., FPT time.

That is, we ask if a certain level d of diversity can be achieved by choosing k pairwise distinct answers to a given query Q over the database instance I. We refer to $\{\gamma_1, \ldots, \gamma_k\}$ as the desired *diversity set*. In the literature, one can find examples of duplicates being allowed and disallowed in the diversity set. In the present work, we disallow duplicates if not mentioned otherwise since it seems counter-intuitive to expect a user to get a broader picture of the solution space when being presented with the same element multiple times. However, we note that all results in this paper remain the same no matter whether we exclude duplicates or not – with one single exception: for the query complexity of the $\mathsf{Diverse}_{\mathsf{sum}}$-$\mathsf{ACQ}$ (formally defined below) problem, we manage to show P-membership if duplicates are allowed (see Theorem 3.10), but only FPT-membership if duplicates are excluded (see Theorem 3.9).

As far as the notation is concerned, we will denote the Hamming distance between two answers γ, γ' by $\Delta(\gamma, \gamma')$. With diversity measure δ, we denote the aggregated Hamming distances of all pairs of k answer tuples for an arbitrary, polynomial-time computable aggregate function f. That is, let $f\colon \bigcup_{k\geq 1} \mathbb{N}^{\frac{k(k-1)}{2}} \to \mathbb{R}$ and let $d_{i,j} = \Delta(\gamma_i, \gamma_j)$ for $1 \leq i < j \leq k$. Then we define $\delta(\gamma_1, \ldots, \gamma_k) := f((d_{i,j})_{1\leq i<j\leq k})$. Sometimes it will be necessary to restrict our attention to concrete aggregators or concrete classes of aggregators. To that end, we write δ_{sum} if the aggregator f is the sum, δ_{mon} if the aggregator f is a monotone function, i.e., $f(d_1, \ldots, d_N) \leq f(d'_1, \ldots, d'_N)$ whenever $d_i \leq d'_i$ holds for every $i \in \{1, \ldots, N\}$ with $N = \frac{k(k-1)}{2}$, and δ_{wsm} if the aggregator f is weakly strictly monotone (ws-monotone) in the sense that $f(d_1, \ldots, d_N) < f(d, \ldots, d)$ whenever $d_i \leq d$ holds for every $i \in \{1, \ldots, N\}$ and at least one $d_i < d$. Note that most natural measures of diversity are ws-monotone, e.g., aggregating via sum or min but may not be strictly monotone, e.g., min. To emphasize the class of diversity measures in question, we denote the corresponding diversity problems by $\mathsf{Diverse}_{\mathsf{sum}}$-$\mathcal{Q}$, $\mathsf{Diverse}_{\mathsf{mon}}$-$\mathcal{Q}$, and $\mathsf{Diverse}_{\mathsf{wsm}}$-$\mathcal{Q}$, respectively.

When we prove upper bounds on the complexity of several variations of the $\mathsf{Diverse}$-$\mathcal{Q}$ problem (in the form of membership in some favorable complexity class), we aim at the most general setting, i.e., membership for all polynomial-time computable aggregation functions. However, in some cases, the restriction to $\mathsf{Diverse}_{\mathsf{sum}}$-$\mathcal{Q}$ or $\mathsf{Diverse}_{\mathsf{mon}}$-$\mathcal{Q}$ will be needed in order to achieve the desired upper bound on the complexity. In contrast, to prove lower bounds (in the form of hardness results), we consider arbitrary ws-monotone diversity measures, i.e., $\mathsf{Diverse}_{\mathsf{wsm}}$-$\mathcal{Q}$. This means that our hardness results do not just hold for a particular diversity measure but for *all* fixed, ws-monotone diversity measures.

We will analyze the $\mathsf{Diverse}$-$\mathcal{Q}$ problem for several query classes $\mathcal{Q}$ – starting with the class CQ of Conjunctive Queries and then extending our studies to the classes UCQ and $\mathsf{CQ}^\neg$ of unions of CQs and CQs with negation. In one case, we will also look at the class FO of all first-order queries. Recall that, for combined complexity and query complexity, even the question, if an answer tuple exists at all, is NP-complete for CQs [CM77]. We therefore mostly restrict our study to acyclic CQs (ACQs, for short) with the corresponding query classes ACQ and UACQ, allowing only ACQs and unions of ACQs, respectively. For CQs with negation, query answering remains NP-complete even if we only allow ACQs [Lan23]. Hence, for $\mathsf{CQ}^\neg$ we have to impose a different restriction. We thus restrict ourselves to CQs with bounded treewidth. Finally note that, even if we have formulated $\mathsf{Diverse}$-$\mathcal{Q}$ as a decision problem, we also care about actually computing k solutions in case of a yes-answer.

We aim at a thorough complexity analysis of the $\mathsf{Diverse}$-$\mathcal{Q}$ problem from various angles. For the most part, we consider the problem parameterized by the size k of the diversity set. In the non-parameterized case (i.e., if k is simply part of the input) we assume k to

be given in unary representation. This assumption is motivated by the fact that for binary representation of k, the size k of the diversity set can be exponentially larger than the input: this contradicts the spirit of the diversity approach which aims at outputting a *small* (not an exponentially big) number of diverse solutions. As is customary in the Database world, we will distinguish combined, query, and data complexity.

Summary of results.

- We start our analysis of the Diverse-$\mathcal{Q}$ problem with the class of ACQs and study data complexity, query complexity, and combined complexity. With the size k of the diversity set as the parameter, we establish XP-membership for combined complexity, which is strengthened to FPT-membership for data complexity. The XP-membership of combined complexity is complemented by a W[1]-lower bound of the $\mathsf{Diverse}_{\mathsf{wsm}}$-ACQ problem. For the non-parameterized case, we show that even the data complexity is NP-hard.
- The FPT-result of data complexity is easily extended to unions of ACQs. Actually, it even holds for arbitrary FO-queries. However, rather surprisingly, we show that the combined complexity and even query complexity of the $\mathsf{Diverse}_{\mathsf{wsm}}$-UACQ problem is NP-complete even when only looking for a pair of diverse answers. That is, the hardness still holds if the size k of the diversity set is 2 and the UACQs are restricted to unions of 2 ACQs.
- Finally, we study the Diverse-$\mathcal{Q}$ problem for the class $\mathsf{CQ}^\neg$. As was mentioned above, the restriction to ACQs is not even enough to make the query answering problem tractable. We, therefore, study the Diverse-$\mathsf{CQ}^\neg$ problem by allowing only classes of CQs of bounded treewidth. The picture is then quite similar to the Diverse-ACQ problem, featuring analogous XP-membership, FPT-membership, W[1]-hardness, and NP-hardness results.

Structure. We present some basic definitions and results in Section 2. In particular, we will formally introduce all concepts of parameterized complexity (complexity classes, reductions) relevant to our study. We then analyze various variants of the Diverse-$\mathcal{Q}$ problem, where $\mathcal{Q}$ is the class of CQs in Section 3, the class of unions of CQs in Section 4, and the class of CQs with negation in Section 5, respectively. Some conclusions and directions for future work are given in Section 6.

2. Preliminaries

Basics. We assume familiarity with relational databases. For basic notions such as schema, (arity of) relation symbols, relations, (active) domain, etc., the reader is referred to any database textbook, e.g., [AHV95]. A CQ is a first-order formula of the form

$$Q(X) := \exists Y \bigwedge_{i=1}^{\ell} A_i,$$

with free variables $X = (x_1, \ldots, x_m)$ and bound variables $Y = (y_1, \ldots, y_n)$ such that each A_i is an atom with variables from $x_1, \ldots, x_m, y_1, \ldots, y_n$. An answer to such a CQ $Q(X)$ over a database instance (or simply "database", for short) I is a mapping $\gamma\colon X \to dom(I)$ which can be extended to a mapping $\bar{\gamma}\colon (X \cup Y) \to dom(I)$ such that instantiating each variable $z \in (X \cup Y)$ to $\bar{\gamma}(z)$ sends each atom A_i into the database I. We write $dom(I)$ to denote the (finite, active) domain of I. By slight abuse of notation, we also refer to the tuple $\gamma(X) = (\gamma(x_1), \ldots, \gamma(x_m))$ as an answer (or an answer tuple). A UCQ is a disjunction

$\bigvee_{i=1}^{N} Q_i(X)$, where all Q_i's are CQs with the same free variables. The set of answers of a UCQ is the union of the answers of its CQs. In a CQ with negation, we allow the A_i's to be either (positive) atoms or literals (i.e., negated atoms) satisfying a safety condition, i.e., every variable has to occur in some positive atom. An answer to a CQ with negation $Q(X)$ over a database I has to satisfy the condition that each positive atom is sent to an atom in the database while each negated atom is not. The set of answers to a query Q over a database I is denoted by $Q(I)$.

For two mappings α and α' defined on variable sets Z and Z', respectively, we write $\alpha \cong \alpha'$ to denote that the two mappings coincide on all variables in $Z \cap Z'$. If this is the case, we write $\alpha \cap \alpha'$ and $\alpha \cup \alpha'$ to denote the mapping obtained by restricting α and α' to their common domain or by combining them to the union of their domains, respectively. That is, $(\alpha \cap \alpha')(z) = \alpha(z)$ for every $z \in Z \cap Z'$ and $(\alpha \cup \alpha')(z)$ is either $\alpha(z)$ if $z \in Z$ or $\alpha'(z)$ otherwise. For another variable set X and $z \in Z$, we write $\alpha|_X$ and $\alpha|_z$ for the mappings resulting from the restriction of α to the set $X \cap Z$ or the singleton $\{z\}$, respectively. Also, the Hamming distance between two mappings can be restricted to a subset of the positions (or, equivalently, of the variables): by $\Delta_X(\alpha, \alpha')$ we denote the number of variables in X on which α and α' differ.

Acyclicity and widths. In a landmark paper [Yan81], Yannakakis showed that query evaluation is tractable (combined complexity) if restricted to *acyclic* CQs. A CQ is acyclic if it has a *join tree*. Given a CQ $Q(X) := \exists Y \bigwedge_{i=1}^{\ell} A_i$ with $At(Q(X)) = \{A_i : 1 \leq i \leq \ell\}$, a join tree of $Q(X)$ is a triple $\langle T, \lambda, r\rangle$ such that $T = (V(T), E(T))$ is a rooted tree with root r and $\lambda\colon V(T) \to At(Q(X))$ is a node labeling function that satisfies the following properties:

(1) The labeling λ is a bijection.
(2) For every $v \in X \cup Y$, the set $T_v = \{t \in V(T) : v \text{ occurs in } \lambda(t)\}$ induces a subtree $T[T_v]$ of T.

As is common, for a graph $T = (V(T), E(T))$ and $H \subseteq V(T)$, we use $T[H]$ to denote the subgraph of T induced by H, i.e. the subgraph consisting of H and all edges in $E(T)$ between nodes in H.

Testing if a given CQ is acyclic and, in case of a yes-answer, constructing a join tree is feasible in polynomial time by the GYO-algorithm, named after the authors of [Gra79, YO79].

Another approach to making CQ answering tractable is by restricting the *treewidth* (*tw*), which is defined via *tree decompositions* [RS84]. Treewidth does not generalize acyclicity, i.e., a class of acyclic CQs can have unbounded *tw*. We consider *tw* here only for CQs with negation. Let $Q(X) := \exists Y \bigwedge_{i=1}^{\ell} L_i$, be a CQ with negation, i.e., each L_i is a (positive or negative) literal. Moreover, let $var(L_i)$ denote the variables occurring in L_i. A tree decomposition of $Q(X)$ is a triple $\langle T, \chi, r\rangle$ such that $T = (V(T), E(T))$ is a rooted tree with root r and $\chi\colon V(T) \to 2^{X \cup Y}$ is a node labeling function with the following properties:

(1) For every L_i, there exists a node $t \in V(T)$ with $var(L_i) \subseteq \chi(t)$.
(2) For every $v \in X \cup Y$, the set $T_v = \{t \in V(T) : v \in \chi(t)\}$ induces a subtree $T[T_v]$ of T.

The property (2) is called the connectedness condition for join trees and tree decomposition. The sets $\chi(t)$ of variables are referred to as "bags" of the tree decomposition T. The width of a tree decomposition is defined as $\max_{t \in V(T)}(|\chi(t)| - 1)$. The treewidth of a CQ with negation Q is the minimum width of all tree decompositions of Q. For fixed ω, it is feasible in linear time w.r.t. the size of the query Q to decide if $tw(Q) \leq \omega$ holds and, in case of a yes-answer, to actually compute a tree decomposition of width $\leq \omega$ [Bod96].

Tree decompositions can be extended to hypertree decompositions (HDs) [GLS02], generalized hypertree decompositions (GHDs) [AGG07], and fractional hypertree decompositions (FHDs) [GM14] by defining an integral edge cover number (in case of HDs and GHDs) or a fractional edge cover number (in case of FHDs) for each bag. Then the width of such a decomposition is the maximum size of all edge cover numbers in the HD, GHD, or FHD. Analogously to treewidth, the hypertree width $hw(Q)$, generalized hypertree width $ghw(Q)$, and fractional hypertree width $\mathit{fhw}(Q)$ of a given query Q is defined as the minimum width attainable over all HDs, GHDs, or FHDs of Q, respectively. Acyclic queries are the ones with $hw = ghw = \mathit{fhw} = 1$.

Note that bounded treewidth and acyclicity are incomparable. Indeed, in case of unbounded arity, even the class of single-atom queries (clearly, all such queries are trivially acyclic) has unbounded treewidth. On the other hand, for instance, the triangle query and its generalization to any cycles has constant treewidth 2, but these queries are, of course, not acyclic. However, the classes of queries with bounded hw, ghw, or fhw properly contain the classes of queries with bounded tw. Moreover, query evaluation of queries with bounded hw, ghw, or fhw can be reduced in polynomial time to query evaluation of acyclic queries. Essentially, the relations of the resulting acyclic query are obtained by carrying out the joins corresponding to each edge cover.

Complexity. We follow the categorization of the complexity of database tasks introduced in [Var82] and distinguish combined/query/data complexity of the Diverse-$\mathcal{Q}$ problem. That is, for data complexity, we consider the query Q as arbitrarily chosen but fixed, while for query complexity, the database instance I is considered fixed. In case of combined complexity, both the query and the database are considered as variable parts of the input.

We assume familiarity with the fundamental complexity classes P (polynomial time) and NP (non-deterministic polynomial time). We study the Diverse-$\mathcal{Q}$ problem primarily from a parameterized complexity perspective [DF99]. An instance of a *parameterized problem* is given as a pair (x, k), where x is the actual problem instance and k is a parameter – usually a non-negative integer. The effort for solving a parameterized problem is measured by a function that depends on both, the size $|x|$ of the instance and the value k of the parameter. The asymptotic worst-case time complexity is thus specified as $\mathcal{O}(f(n, k))$ with $n = |x|$.

The parameterized analogue of *tractability* captured by the class P is *fixed-parameter tractability* captured by the class FPT of fixed-parameter tractable problems. A problem is in FPT, if it can be solved in time $\mathcal{O}(f(k) \cdot n^c)$ for some computable function f and constant c. In other words, the running time only depends polynomially on the size of the instance, while a possibly exponential explosion is confined to the parameter. In particular, if for a class of instances, the parameter k is bounded by a constant, then FPT-membership means that the problem can be solved in polynomial time. This also applies to problems in the slightly less favorable complexity class XP, which contains the problems solvable in time $\mathcal{O}(n^{f(k)})$.

Parameterized complexity theory also comes with its own version of reductions (namely "FPT-reductions") and hardness theory based on classes of fixed-parameter intractable problems. An FPT-reduction from a parameterized problem P to another parameterized problem P' maps every instance (x, k) of P to an equivalent instance (x', k') of P', such that k' only depends on k (i.e., independent of x) and the computation of x' is in FPT (i.e., in time $\mathcal{O}(f(k) \cdot |x|^c)$ for some computable function f and constant c). For *fixed-parameter intractability*, the most prominent class is W[1]. It has several equivalent definitions, for

instance, W[1] is the class of problems that allow for an FPT-reduction to the Independent Set problem parameterized by the desired size k of an independent set. We have $\mathsf{FPT} \subseteq \mathsf{W}[1] \subseteq \mathsf{XP}$. It is a generally accepted assumption in parameterized complexity theory that $\mathsf{FPT} \neq \mathsf{W}[1]$ holds – similar but slightly stronger than the famous $\mathsf{P} \neq \mathsf{NP}$ assumption in classical complexity theory, i.e., $\mathsf{FPT} \neq \mathsf{W}[1]$ implies $\mathsf{P} \neq \mathsf{NP}$, but not vice versa.

3. Diversity of Conjunctive Queries

3.1. **Combined and Query Complexity.** We start our study of the Diverse-ACQ problem by considering the combined complexity and then, more specifically, the query complexity. We will thus present our basic algorithm in Section 3.1.1, which allows us to establish the XP-membership of this problem. We will then prove W[1]-hardness in Section 3.1.2 and present some further improvements of the basic algorithm in Section 3.1.3.

3.1.1. *Basic Algorithm.* Our algorithm for solving Diverse-ACQ is based on a dynamic programming idea analogous to the Yannakakis algorithm. Given a join tree $\langle T, \lambda, r\rangle$ and database I, the Yannakakis algorithm decides in a bottom-up traversal of T at each node $t \in V(T)$ and for each answer α to the single-atom query $\lambda(t)$ if it can be extended to an answer to the CQ consisting of all atoms labeling the nodes in the complete subtree T' rooted at t. It then stores this (binary) information by either keeping or dismissing α. Our algorithm for Diverse-ACQ implements a similar idea. At its core, it stores *k-tuples* $(\alpha_1, \ldots, \alpha_k)$ of answers to the single-atom query $\lambda(t)$, each k-tuple describing a set of (partial) diversity sets. We extend this information by the various vectors $(d_{i,j})_{1\leq i<j\leq k}$ of Hamming distances that are attainable by possible extensions $(\gamma_1, \ldots, \gamma_k)$ to the CQ consisting of the atoms labeling the nodes in T'.

In the following, we consider an ACQ $Q(X) := \exists Y \bigwedge_{i=1}^{\ell} A_i$ where each atom is of the form $A_i = R_i(Z_i)$ for some relation symbol R_i and variables $Z_i \subseteq X \cup Y$. For an atom $A = R(Z)$ and a database instance I, define $A(I)$ as the set of mappings $\{\alpha\colon Z \to dom(I) : \alpha(Z) \in R^I\}$. We extend the definition to sets (or conjunctions) $\psi(Z)$ of atoms $A_i(Z_i)$ with $Z_i \subseteq Z$. Then $\psi(I)$ is the set of mappings $\{\alpha\colon Z \to dom(I) : \alpha(Z_i) \in R_i^I \text{ for all } R_i(Z_i) \in \psi(Z)\}$. Let $\langle T, \lambda, r\rangle$ be a join tree. For a subtree T' of T we define $\lambda(T') = \{\lambda(t) : t \in V(T')\}$ and, by slight abuse of notation, we write $t(I)$ and $T'(I)$ instead of $\lambda(t)(I)$ and $\lambda(T')(I)$. Now consider T' to be a subtree of T with root t. For tuples

$$e \in \{(\alpha_1, \ldots, \alpha_k, (d_{i,j})_{1\leq i<j\leq k}) : \alpha_1, \ldots, \alpha_k \in t(I), d_{i,j} \in \{0, \ldots, |X|\} \text{ for } 1 \leq i < j \leq k\},$$

we define

$$ext_{T'}(e) = \{(\gamma_1, \ldots, \gamma_k) : \gamma_1, \ldots, \gamma_k \in T'(I) \text{ s.t. } \alpha_i \cong \gamma_i \text{ for } 1 \leq i \leq k \text{ and } \Delta_X(\gamma_i, \gamma_j) = d_{i,j} \text{ for } 1 \leq i < j \leq k\}.$$

Intuitively, the algorithm checks for each such tuple e whether there exist extensions γ_i of α_i that

(1) are solutions to the subquery induced by T', and
(2) exhibit $d_{i,j}$ as their pairwise Hamming distances. If this is the case, the tuple e is kept, otherwise, e is dismissed.

In doing so, the goal of the algorithm is to compute sets $D_{T'}$ that contain exactly those e with $ext_{T'}(e) \neq \emptyset$. Having computed D_T (i.e., for the whole join tree), Diverse-ACQ can now be decided by computing for each $e \in D_T$ the diversity measure from the values $d_{i,j}$.

To do so, in a first phase, at every node $t \in V(T)$, we need to compute and store the set $D_{T'}$ (for T' being the complete subtree rooted in t). We compute this set by starting with some set D_t and updating it until eventually, it is equal to $D_{T'}$. In addition, to every entry e in every set D_t, we maintain a set $\rho_{D_t}(e)$ containing provenance information on e. Afterwards, in the recombination phase, the sets $D_{T'}$ and $\rho_{D_t}(\cdot)$ are used to compute a diversity set with the desired diversity – if such a set exists.

Algorithm 3.1. Given $Q(X)$, I, $\langle T, \lambda, r\rangle$, k, d, and a diversity measure δ defined via some aggregate function f, the first phase proceeds in three main steps:

- **Initialization:** In this step, for every node $t \in V(T)$, initialize the set D_t as

$$D_t = \{(\alpha_1, \dots, \alpha_k, (d_{i,j})_{1\leq i<j\leq k}) : \alpha_i \in t(I), d_{i,j} = \Delta_X(\alpha_i, \alpha_j)\}.$$

  That is, D_t contains one entry for every combination $\alpha_1, \dots, \alpha_k \in t(I)$, and each value $d_{i,j}$ $(1 \leq i < j \leq k)$ is the Hamming distance of the mappings $\alpha_i|_X$ and $\alpha_j|_X$.

  For every $e \in D_t$, initialize $\rho_{D_t}(e)$ as the empty set.

  Finally, set the status of all non-leaf nodes in T to "not-ready" and the status of all leaf nodes to "ready".
- **Bottom-Up Traversal:** Then repeat the following action until no "not-ready" node is left: Pick any "not-ready" node t that has at least one "ready" child node t'. Update D_t to D_t^{new} as

$$\begin{aligned} D_t^{new} = \{(\alpha_1, \dots, \alpha_k, (\bar{d}_{i,j})_{1\leq i<j\leq k}) : {} & (\alpha_1, \dots, \alpha_k, (d_{i,j})_{1\leq i<j\leq k}) \in D_t, \\ & (\alpha'_1, \dots, \alpha'_k, (d'_{i,j})_{1\leq i<j\leq k}) \in D_{t'}, \\ & \alpha_i \cong \alpha'_i \text{ for } 1 \leq i \leq k, \\ & \bar{d}_{i,j} = d_{i,j} + d'_{i,j} - \Delta_X(\alpha_i \cap \alpha'_i, \alpha_j \cap \alpha'_j) \\ & \qquad\qquad \text{for } 1 \leq i < j \leq k\}. \end{aligned}$$

  Expressed in a more procedural style: Take every entry $e \in D_t$ and compare it to every entry $e' \in D_{t'}$. If the corresponding mappings $\alpha_i \in D_t$ and $\alpha'_i \in D_{t'}$ agree on the shared variables, the new set D_t^{new} contains an entry $\bar{e}$ with the mappings α_i from e and the Hamming distances computed from e and e' as described above.

  Set $\rho_{D_t^{new}}(\bar{e}) = \rho_{D_t}(e) \cup \{(t', e')\}$. If the same entry $\bar{e}$ is created from different pairs (e, e'), choose an arbitrary one of them for the definition of $\rho_{D_t^{new}}(\bar{e})$.

  Finally, change the status of t' from "ready" to "processed". The status of t becomes "ready" if the status of all its child nodes is "processed" and remains "not-ready" otherwise.
- **Finalization:** Once the status of root r is "ready", remove all $(\alpha_1, \dots, \alpha_k, (d_{i,j})_{1\leq i<j\leq k}) \in D_r$ where $f((d_{i,j})_{1\leq i<j\leq k}) < d$. To ensure that all answers in the diversity set are pairwise distinct, also remove all entries where $d_{i,j} = 0$ for some (i, j) with $1 \leq i < j \leq k$.

  If, after the deletions, D_r is empty, then there exists no diversity set of size k with a diversity of at least d. Otherwise, at least one such diversity set exists.

Clearly, the algorithm is well-defined and terminates. In the following theorem, we show that the algorithm decides Diverse-ACQ and we give an upper bound on its running time.

Theorem 3.2. *The Diverse-ACQ problem is in* XP *in combined complexity when parameterized by the size k of the diversity set. More specifically, for an ACQ $Q(X)$, a database I, and integers k and d, Algorithm 3.1 decides the Diverse-ACQ problem in time $\mathcal{O}\big(|R^I|^{2k} \cdot (|X|+1)^{k(k-1)} \cdot poly(|Q|,k)\big)$ where R^I is the relation from I with the most tuples and $poly(|Q|,k)$ is a polynomial in $|Q|$ and k.*

This result is a consequence of the correctness of Algorithm 3.1. We show both, the correctness of the algorithm and Theorem 3.2 using a sequence of lemmas, discussed and proven next. Consider an ACQ $Q(X)$ with join tree $\langle T, \lambda, r\rangle$, a database instance I, and integers k (the number of elements in the diversity set) and d (the required diversity). For a subtree T' of T with root t, let $D_{T'}$ be the set of tuples

$$e \in \{(\alpha_1, \ldots, \alpha_k, (d_{i,j})_{1\le i<j\le k}) : \alpha_1, \ldots, \alpha_k \in t(I) \text{ and } d_{i,j} \in \{0, \ldots, |X|\} \text{ for } 1 \le i < j \le k\}$$

such that the set

$$ext_{T'}(e) = \{(\gamma_1, \ldots, \gamma_k) : \gamma_1, \ldots, \gamma_k \in T'(I) \text{ s.t. } \alpha_i \cong \gamma_i \text{ for } 1 \le i \le k \text{ and } \Delta_X(\gamma_i, \gamma_j) = d_{i,j} \text{ for } 1 \le i < j \le k\}$$

is not empty. To prove the correctness of Algorithm 3.1, it is sometimes more convenient to work with the following, obviously equivalent, definition of $D_{T'}$:

$$D_{T'} = \{(\alpha_1, \ldots, \alpha_k, (d_{i,j})_{1\le i<j\le k}) : \alpha_1, \ldots, \alpha_k \in t(I), \gamma_1, \ldots, \gamma_k \in T'(I), \gamma_1 \cong \alpha_1, \ldots, \gamma_k \cong \alpha_k, d_{i,j} = \Delta_X(\gamma_i, \gamma_j) \text{ for } 1 \le i < j \le k\}.$$

The overall goal of the following lemmas is to

(1) show that once the status of a node $t \in V(T)$ is "ready", the equality $D_t = D_{T'}$ holds, where T' is the complete subtree of T rooted in t (i.e., the subtree of T containing t and all of its descendants). This proves the correctness of the algorithm.
(2) provide a bound on the running time of the different steps of the algorithm.

The first lemma, describing the size of the sets D_t, follows immediately from the definition and the observation that the value of each of the entries $d_{i,j}$ is at most $|X|$.

Lemma 3.3. *Let $Q(X)$ be an ACQ, I a database instance, and $\langle T, \lambda, r\rangle$ a join tree for $Q(X)$. Throughout the running time of Algorithm 3.1, for every node $t \in V(T)$ the set D_t contains at most $|R^I|^k \cdot (|X|+1)^{\frac{k(k-1)}{2}}$ tuples, where R^I is the relation in I containing the most tuples. The size of each tuple is polynomial in the size of the input.*

Proof. The polynomial size of each tuple is immediate. For the number of entries, observe that the number of different elements in every $t(I)$ is $|t(I)| \le |R^I|$ since $\lambda(t)$ consists of a single atom. Also, since each $d_{i,j}$ describes the Hamming-Distance between two mappings with at most $|X|$ variables, its value is in $\{0, \ldots, |X|\}$. The expression $|R^I|^k \cdot (|X|+1)^{\frac{k(k-1)}{2}}$ thus describes the number of all possible combinations of these values for tuples of size $k + \frac{k(k-1)}{2}$ where the first k are elements of $t(I)$ and the remaining elements from $\{0, \ldots, |X|\}$. □

The next lemma shows that the initialization correctly computes $D_{T'}$ for all subtrees T' of T consisting of a single node, and states the running time of this step.

Lemma 3.4. *Let $Q(X)$ be an ACQ, I a database instance, and $\langle T, \lambda, r\rangle$ a join tree for $Q(X)$. Once the "Initialization"-step of Algorithm 3.1 is complete, the equality $D_t = D_{T'}$ holds for all nodes $t \in V(T)$, where T' is the subtree of T consisting only of t. Furthermore, D_t can be computed in time $\mathcal{O}(|t(I)|^k \cdot k^2 \cdot |var(t)|)$.*

Proof. One helpful observation for both, correctness and the running time, is that no collection $\alpha_1, \ldots, \alpha_k$ may occur twice, each time with different values $(d_{i,j})_{1\leq i<j\leq k}$ and $(d'_{i,j})_{1\leq i<j\leq k}$, in $D_{T'}$: because the subtree T' consists only of t, all $(\alpha_1, \ldots, \alpha_k)$ are their only extension and thus determines the values $(d_{i,j})_{1\leq i<j\leq k}$. The equality $D_t = D_{T'}$ then follows immediately from the definitions of D_t and $D_{T'}$.

For the time bound, observe that the given time allows one to iterate through all possible tuples $(\alpha_1, \ldots, \alpha_k)$ with $\alpha_1, \ldots, \alpha_k \in t(I)$ ($|t(I)|^k$ many), and for each such tuple to compute, for each pair α_i, α_j with $1 \leq i < j \leq k$ (less than k^2 many) the value $d_{i,j} = \Delta_X(\alpha_i, \alpha_j)$ ($X \cap var(t) \subseteq var(t)$ many variables to compare), which constitutes a naive implementation of the "Initialization" step. □

The next lemma will be essential in proving the "Bottom-Up Traversal" step of Algorithm 3.1 being correct, and provides a bound on the running time for a single iteration of this step.

Lemma 3.5. *Let $Q(X)$ be an ACQ, I a database instance, and $\langle T, \lambda, r\rangle$ a join tree for $Q(X)$. Let $\langle T_1, t_1\rangle$ and $\langle T_2, t_2\rangle$ be two disjoint rooted subtrees of $\langle T, r\rangle$ (i.e. $V(T_1) \cap V(T_2) = \emptyset$) such that t_1 is the parent node of t_2 in $\langle T, r\rangle$, and for $\hat{T} = T[V(T_1) \cup V(T_2)]$ consider the rooted subtree $\langle \hat{T}, t_1\rangle$ of $\langle T, r\rangle$. Then*

$$\begin{aligned} D_{\hat{T}} = \{(\alpha_1, \ldots, \alpha_k, (\hat{d}_{i,j})_{1\leq i<j\leq k}) : {} & (\alpha_1, \ldots, \alpha_k, (d_{i,j})_{1\leq i<j\leq k}) \in D_{T_1}, \\ & (\alpha'_1, \ldots, \alpha'_k, (d'_{i,j})_{1\leq i<j\leq k}) \in D_{T_2}, \\ & \alpha_i \cong \alpha'_i \text{ for } 1 \leq i \leq k, \\ & \hat{d}_{i,j} = d_{i,j} + d'_{i,j} - \Delta_X(\alpha_i \cap \alpha'_i, \alpha_j \cap \alpha'_j) \\ & \qquad \text{for } 1 \leq i < j \leq k\}. \end{aligned} \tag{3.1}$$

Also, given D_{T_1} and D_{T_2}, the set $D_{\hat{T}}$ can be computed in time $\mathcal{O}(|D|^2 \cdot k^2 \cdot |Z|)$ where D is the larger of the two sets D_{T_1} and D_{T_2}, and Z is the larger of the sets $var(t_1)$ and $var(t_2)$.

Proof. For this proof, we will use $D_{\hat{T}}$ to describe the set of tuples according to the initial definition, and $\hat{D}_{\hat{T}}$ for the set defined by the right hand side of Equation 3.1. We show $D_{\hat{T}} = \hat{D}_{\hat{T}}$ by proving $D_{\hat{T}} \subseteq \hat{D}_{\hat{T}}$ and $\hat{D}_{\hat{T}} \subseteq D_{\hat{T}}$ separately.

$D_{\hat{T}} \subseteq \hat{D}_{\hat{T}}$:

Let $e = (\alpha_1, \ldots, \alpha_k, (\hat{d}_{i,j})_{1\leq i<j\leq k})) \in D_{\hat{T}}$. Consider an arbitrary $(\hat{\gamma}_1, \ldots, \hat{\gamma}_k) \in ext_{\hat{T}}(e)$, and define $\gamma_i = \hat{\gamma}_i|_{var(T_1)}$ and $\gamma'_i = \hat{\gamma}_i|_{var(T_2)}$ for all $1 \leq i \leq k$. By definition, $\gamma_i \in T_1(I)$ and $\gamma'_i \in T_2(I)$, thus we have

$$\begin{aligned} e_1 &= (\gamma_1|_{var(t_1)}, \ldots, \gamma_k|_{var(t_1)}, (\Delta_X(\gamma_i, \gamma_j))_{1\leq i<j\leq k}) \in D_{T_1} \text{ and} \\ e_2 &= (\gamma'_1|_{var(t_2)}, \ldots, \gamma'_k|_{var(t_2)}, (\Delta_X(\gamma'_i, \gamma'_j))_{1\leq i<j\leq k}) \in D_{T_2}. \end{aligned}$$

This is the case since clearly

$$\begin{aligned} &(\gamma_1, \ldots, \gamma_k) \in ext_{T_1}((\gamma_1|_{var(t_1)}, \ldots, \gamma_k|_{var(t_1)}, (\Delta_X(\gamma_i, \gamma_j))_{1\leq i<j\leq k})) \text{ and} \\ &(\gamma'_1, \ldots, \gamma'_k) \in ext_{T_2}((\gamma'_1|_{var(t_2)}, \ldots, \gamma'_k|_{var(t_2)}, (\Delta_X(\gamma'_i, \gamma'_j))_{1\leq i<j\leq k})). \end{aligned}$$

Now $\alpha_i = \gamma_i|_{var(t_1)}$ and we define $\alpha'_i = \gamma'_i|_{var(t_2)}$. We get $\alpha_i \cong \alpha'_i$ for all $1 \leq i \leq k$. Next, for $1 \leq i < j \leq k$, we have

$$\begin{aligned}\hat{d}_{i,j} &= \Delta_X(\hat{\gamma}_i, \hat{\gamma}_j) = \Delta_{X \cap \left(var(T_1) \cup var(T_2)\right)}(\hat{\gamma}_i, \hat{\gamma}_j) \\ &= \Delta_{X \cap \left(var(T_1) \cup (var(T_2) \setminus var(T_1))\right)}(\hat{\gamma}_i, \hat{\gamma}_j) = \Delta_{X \cap var(T_1)}(\hat{\gamma}_i, \hat{\gamma}_j) + \Delta_{X \cap \left(var(T_2) \setminus var(T_1)\right)}(\hat{\gamma}_i, \hat{\gamma}_j).\end{aligned}$$

Now

$$\Delta_X(\gamma_i, \gamma_j) = \Delta_{X \cap var(T_1)}(\hat{\gamma}_i, \hat{\gamma}_j) \text{ and}$$
$$\Delta_X(\gamma'_i, \gamma'_j) = \Delta_{X \cap var(T_2)}(\hat{\gamma}_i, \hat{\gamma}_j) = \Delta_{X \cap (var(T_2) \setminus var(T_1))}(\hat{\gamma}_i, \hat{\gamma}_j) + \Delta_{X \cap (var(T_2) \cap var(T_1))}(\hat{\gamma}_i, \hat{\gamma}_j).$$

We end up with $\hat{d}_{i,j} = \Delta_X(\gamma_i, \gamma_j) + \Delta_X(\gamma'_i, \gamma'_j) - \Delta_{X \cap (var(T_2) \cap var(T_1))}(\hat{\gamma}_i, \hat{\gamma}_j)$. Given that γ_i and γ'_i share exactly the variables from $var(T_1) \cap var(T_2)$, we get $\Delta_{X \cap (var(T_2) \cap var(T_1))}(\hat{\gamma}_i, \hat{\gamma}_j) = \Delta_X(\gamma_i \cap \gamma'_i, \gamma_j \cap \gamma'_j)$. Because of the connectedness condition, all variables shared between any γ_i and γ'_i are also contained in $\gamma_i|_{var(t_1)}$ and $\gamma'_i|_{var(t_2)}$ and therefore $\Delta_X(\gamma_i \cap \gamma'_i, \gamma_j \cap \gamma'_j) = \Delta_X(\alpha_i \cap \alpha'_i, \alpha_j \cap \alpha'_j)$. Thus $e \in \hat{D}_{\hat{T}}$ is verified by the tuples e_1 and e_2, which, together with e satisfy all conditions stated on the right-hand side of Equation 3.1, which concludes this direction of the proof.

$\hat{D}_{\hat{T}} \subseteq D_{\hat{T}}$: Consider an arbitrary tuple $\hat{e} = (\alpha_1, \ldots, \alpha_k, (\hat{d}_{i,j})_{1 \leq i < j \leq k}) \in \hat{D}_{\hat{T}}$, and let $e_1 = (\alpha_1, \ldots, \alpha_k, (d_{i,j})_{1 \leq i < j \leq k}) \in D_{T_1}$ and $e_2 = (\alpha'_1, \ldots, \alpha'_k, (d'_{i,j})_{1 \leq i < j \leq k}) \in D_{T_2}$ be two tuples witnessing $\hat{e} \in \hat{D}_{\hat{T}}$ (i.e. $\hat{e}$, e_1, and e_2 satisfy all conditions on the right-hand side of Equation 3.1). Then $ext_{T_1}(e_1)$ and $ext_{T_2}(e_2)$ are both not empty. Choose $(\gamma_1, \ldots, \gamma_k) \in ext_{T_1}(e_1)$ and $(\gamma'_1, \ldots, \gamma'_k) \in ext_{T_2}(e_2)$ arbitrarily. By definition, $\Delta_X(\gamma_i, \gamma_j) = d_{i,j}$ and $\Delta_X(\gamma'_i, \gamma'_j) = d'_{i,j}$ for all $1 \leq i < j \leq k$. Because of the connectedness condition for join trees,

(1) the mapping $\hat{\gamma}_i = \gamma_i \cup \gamma'_i$ is a valid mapping,
(2) $\hat{\gamma}_i|_{var(t_1)} = \alpha_i$, and
(3) $\hat{\gamma}_i \in \hat{T}(I)$ (and thus $\alpha_i \in t_1(I)$).

Thus $(\alpha_1, \ldots, \alpha_k, (\Delta_X(\hat{\gamma}_i, \hat{\gamma}_j))_{1 \leq i < j \leq k}) \in D_{\hat{T}}$, and proving $\Delta_X(\hat{\gamma}_i, \hat{\gamma}_j) = \hat{d}_{i,j}$ concludes the proof. Towards this goal, by an equivalent development as for proving the other direction, we get

$$\begin{aligned}\Delta_X(\hat{\gamma}_i, \hat{\gamma}_j) &= \Delta_X(\gamma_i, \gamma_j) + \Delta_X(\gamma'_i, \gamma'_j) - \Delta_X(\gamma_i \cap \gamma'_i, \gamma_j \cap \gamma'_j) \\ &= d_{i,j} + d'_{i,j} - \Delta_X(\alpha_i \cap \alpha'_i, \alpha_j \cap \alpha'_j) \\ &= \hat{d}_{i,j}.\end{aligned}$$

To prove that $\hat{D}_{\hat{T}}$ can in fact be computed within the stated time bound, consider the following naive implementation: Iterate through all $e_1 \in D_{T_1}$, and for each such e_1 – in a nested loop – look at each $e_2 \in D_{T_2}$ ("Loop"). For each such pair with $e_1 = (\alpha_1, \ldots, \alpha_k, (d_{i,j})_{1 \leq i < j \leq k})$ and $e_2 = (\alpha'_1, \ldots, \alpha'_k, (d'_{i,j})_{1 \leq i < j \leq k})$, check whether $\alpha_i \cong \alpha'_i$ ("Check"). If $\alpha_i \cong \alpha'_i$, compute $\hat{d}_{i,j}$ as defined ("Compute"). "Check" requires to compare the values of $|Z|$ variables on k pairs (α_i, α'_i) (possible in $\mathcal{O}(k \cdot |Z|)$ time), and "Compute" computes $\frac{k(k-1)}{2}$ many values (thus in $\mathcal{O}(k^2)$). These two steps are performed $\mathcal{O}(|D_{T_1}| \cdot |D_{T_2}|)$ times (number of iterations of "Loop"). Despite "Check" and "Compute" being sequential, for simplicity we bound the running time by $\mathcal{O}(|D|^2 \cdot k^2 \cdot |Z|)$ instead of $\mathcal{O}(|D|^2 \cdot (k^2 + k \cdot |Z|))$. □

With this result at hand, we can show that the bottom-up traversal of the join tree is correct.

Lemma 3.6. *Let $Q(X)$ be an ACQ, I a database instance, and $\langle T, \lambda, t\rangle$ a join tree for $Q(X)$. At the end of every iteration, the "Bottom-Up Traversal" step of Algorithm 3.1 guarantees the following two properties:*

(1) *For all nodes $t \in V(T)$ with status "ready", the equality $D_t = D_{T'}$ holds, where T' is the subtree of T consisting of t and all its descendants.*
(2) *For a node $t \in V(T)$, let $t_1, \ldots, t_p$ be the child nodes with status "processed". Then $D_t = D_{T'}$ where T' is the subtree of T consisting of t and all t_i and all their descendants, for $1 \leq i \leq p$.*

Proof. We show both properties by induction on the number of steps in the bottom-up traversal of the join tree. Throughout this proof, for a node $t \in V(T)$, we will use T_t to denote the complete subtree of T rooted in t, i.e. the subtree of T containing t and all its descendants.

For the base case, consider the situation before the first iteration of the bottom-up traversal. At this point, the set of nodes with status "ready" are exactly the leaf nodes. Since they have no child nodes, the statement $D_t = D_{T'}$ in property (1) is equivalent to $D_t = D_{T[\{t\}]}$. By Lemma 3.4, this equality holds for all nodes once the "Initialization" step is finished. We next observe that there are no nodes with status "processed". Thus for all nodes $t \in V(T)$ property (2) also states $D_t = D_{T[\{t\}]}$, which again holds because of Lemma 3.4.

For the induction step, consider the node $t \in V(T)$ for which D_t was updated to D_t^{new} in the "Bottom-Up Traversal" step, and let t' be the child node of t that was used to compute the update. As induction hypothesis, we know that the first property holds for the child node t' (status before the step: "ready"), and the second property holds for t w.r.t. all the child nodes $t_1, \ldots, t_p$ of t with status "processed" (possibly none). We have to show that after the "Bottom-Up Traversal" step:

a) The second property holds for t and the child nodes $t_1, \ldots, t_p, t'$.
b) If t' was the only remaining child node of t with a status different from "processed", then the first property now holds for t.

To prove a), let $T' = T[\{t\} \cup V(T_{t_1}) \cup \ldots V(T_{t_p})]$ and $\hat{T} = T[\{t\} \cup V(T_{t_1}) \cup \cdots \cup V(T_{t_p}) \cup V(T_{t'})]$. The induction hypothesis guarantees that Lemma 3.5 applies (i.e. all the preconditions are satisfied w.r.t. T', $T_{t'}$, and $\hat{T}$). Observe that the set described in Lemma 3.5 is exactly the set D_t^{new} computed from D_t by the "Bottom-Up Traversal" step. We thus have $D_t = D_{T'}$ by the induction hypothesis, and $D_t^{new} = D_{\hat{T}}$ by Lemma 3.5, which completes the proof of a).

For b), observe that the only node whose status can switch to "ready" is t. If this happened at the end of the step, then t' was the last child of t whose status was not "processed", and we now have $\hat{T} = T_t$. Thus $D_t = D_{T_t}$ follows immediately from a), concluding the proof of the lemma. □

We now have everything in place to prove Theorem 3.2.

Proof (of Theorem 3.2). We start by proving the correctness of the algorithm, before discussing its running time.

For the correctness, from Lemma 3.6 we know that once the "Bottom-Up Traversal" step is finished (i.e. there is no more node with status "not-ready"; in other words, the root has status "ready" and all other nodes have status "processed"), then $D_r = D_T$ (r is the root of T). As a result, for any $e = (\alpha_1, \ldots, \alpha_k, (d_{i,j})_{1 \leq i < j \leq k}) \in D_r$ and every $(\gamma_1, \ldots, \gamma_k) \in ext_T(e)$ we have $\gamma_i \in T(I)$ and $\gamma_i|_X \in Q(I)$ for all $1 \leq i \leq k$. Hence

$\delta(\gamma_1, \ldots, \gamma_k) = f((\Delta_X(\gamma_i, \gamma_j))_{1 \leq i < j \leq k}) = f((d_{i,j})_{1 \leq i < j \leq k})$ (with f being the polynomial time computable function defining δ).

Thus the correctness of the "Finalization" step follows immediately from

$$\begin{aligned}
\max_{\substack{\gamma_1,\ldots,\gamma_k \in Q(I) \\ \gamma_i \neq \gamma_j \text{ for } i \neq j}} \delta(\gamma_1, \ldots, \gamma_k) &= \max_{\substack{\gamma_1,\ldots,\gamma_k \in T(I) \\ \gamma_i|_X \neq \gamma_j|_X \text{ for } i \neq j}} \delta(\gamma_1|_X, \ldots, \gamma_k|_X) \\
&= \max_{\substack{\gamma_1,\ldots,\gamma_k \in T(I) \\ \Delta_X(\gamma_i,\gamma_j) > 0 \text{ for } i \neq j}} f((\Delta_X(\gamma_i, \gamma_j))_{1 \leq i < j \leq k}) \\
&= \max_{\substack{(\alpha_1,\ldots,\alpha_k,(d_{i,j})_{1 \leq i < j \leq k}) \in D_r \\ d_{i,j} > 0 \text{ for } 1 \leq i < j \leq k}} f((d_{i,j})_{1 \leq i < j \leq k}).
\end{aligned}$$

For the bound on the running time, by Lemma 3.4, the "Initialization" step takes time in $\mathcal{O}(|t(I)|^k \cdot k^2 \cdot |var(t)|)$ for each node, i.e. $\mathcal{O}(|R^I|^k \cdot k^2 \cdot |var(A)| \cdot |Q|)$ in total (the join tree contains one node for each atom in $Q(X)$), where R^I is the relation in I with the highest number of tuples, and A is the atom in Q with the highest number of variables.

By Lemma 3.5, one iteration of the "Bottom-Up Traversal" step takes time in $\mathcal{O}((|R^I|^k \cdot (|X|+1)^{\frac{k(k-1)}{2}})^2 \cdot k^2 \cdot |var(A)|)$ using the size bound on D_t from Lemma 3.3. Since every node (except the root node) is merged into its parent node exactly once, we get $\mathcal{O}(|R^I|^{2k} \cdot (|X|+1)^{k(k-1)} \cdot k^2 \cdot |var(A)|) \cdot |Q|)$ in total. The "Finalization" step takes time $\mathcal{O}(|R^I|^{2k} \cdot (|X|+1)^{k(k-1)} \cdot poly_f(|X|, k))$, where $poly_f(|X|, k)$ is a polynomial describing the time needed to compute the function $f((d_{i,j})_{1 \leq i < j \leq k})$. Thus the running time of the "Bottom-Up Traversal" dominates the running time of the "Initialization" step, which is why we can omit it, providing the running time stated in the theorem. □

Theorem 3.2 shows that the algorithm *decides* in XP the existence of a diversity set with a given diversity. Computing a witness diversity set now means computing one element $(\gamma_1, \ldots, \gamma_k) \in ext_T(e)$ for some $e \in D_T$ with $f((d_{i,j})_{1 \leq i < j \leq k}) \geq d$ and $d_{i,j} \neq 0$ for all i, j. Similarly to the construction of an answer tuple by the Yannakakis algorithm for CQs, we can compute an arbitrary element from $ext_T(e)$ by making use of the information stored in the final sets $\rho_{D_t}(e)$. By construction, for every node $t \in V(T)$ and every entry $e \in D_{T'}$, the final set $\rho_{D_t}(e)$ contains exactly one pair (t', e') for every child node t' of t. Moreover, for the mappings $\alpha_1, \ldots, \alpha_k$ from e and $\alpha'_1, \ldots, \alpha'_k$ from e', $\alpha_i \cong \alpha'_i$ holds for all $1 \leq i \leq k$, hence $\alpha_i \cup \alpha'_i$ are again mappings. Thus, to compute the desired witness $(\gamma_1, \ldots, \gamma_k) \in ext_T(e)$ for the chosen $e \in D_T$, start with $(\alpha_1, \ldots, \alpha_k)$ from e, take all (t', e') from $\rho_{D_r}(e)$, extend each α_i with α'_i from e', and repeat this step recursively.

Example 3.7. An example execution of the basic algorithm for $k = 2$ on the query

$$Q(x_1, \ldots, x_8) :- R_1(x_1, x_2, x_3) \wedge R_2(x_2, x_3, x_4) \wedge R_3(x_4, x_5) \wedge R_4(x_4) \wedge R_5(x_5, x_6) \wedge R_6(x_7, x_8)$$

which, together with a possible join tree, is shown in Figure 1. The database I consists of (very small) relations $R^I_1, \ldots, R^I_6$ and each relation R^I_i is shown in the figure next to the node R_i. For a node t, the set D_t is computed by considering the subtrees rooted at the children of t from left to right. For the sake of succinctness, tuples $(\alpha_1, \alpha_2, d_{1,2}) \in D_t$ are omitted if there is a strictly better tuple, i.e., a $(\alpha_1, \alpha_2, d'_{1,2}) \in D_t$ such that $d_{1,2} < d'_{1,2}$. (Formally, this is only justified if the aggregator is monotone.) We do not specify an aggregator and skip the "Finalization" step as we are only looking for a diverse pair. This pair is $\{(3, 2, 2, 0, 0, 1, 5, 7), (3, 2, 2, 0, 0, 2, 8, 8)\}$.

We now discuss the "Initialization" and "Bottom-Up Traversal" of Algorithm 3.1 in more detail: We first carry out the initialization step of Algorithm 3.1 for all nodes. That is, we set up a table with all possible pairs of tuples from R_i^I together with their Hamming distance (recall that we are looking for pairs since we have $k = 2$ in this example). In particular, if a pair consists of two identical tuples, then we get a distance of 0.

We next discuss the result of carrying out the bottom-up traversal. To this end, we inspect the two internal nodes R_3 and R_2. First, look at the tables to the right of the node R_3: the tuple $(3, 1)$ has no join partner in the leftmost child ($= R_4$). Hence, in the table to the right of the initial one, we delete all pairs that contain the tuple $(3, 1)$. Therefore, we only consider pairs built from the first two tuples in R_3^I, i.e., $(0, 0)$ and $(1, 1)$. Clearly, extending these pairs to the leftmost child does not add to the distance, since that node ($= R_4$) has no additional variable. In the left table below, we show the result of extending these 4 pairs of tuples to the second child. It turns out that, for the first 3 pairs, the maximum achievable distance increases by 1 because we could extend the tuples of such a pair in two different ways to x_6. Now let us also look at the last pair in this table, i.e., combining $(1, 1)$ with $(1, 1)$. That is, in both tuples, x_5 is set to 1. But when we look at the table corresponding to R_5, it turns out that the only possible extension to x_6 is 1. Hence, the distance of this pair cannot be increased by an extension to R_5 and it remains 2. We then carry out the bottom-up step also from the child node R_6 to R_3. Now we can indeed extend the tuples of each pair to different values of x_7 and also x_8, which leads to an increase of the distance by 2. That is, we end up with (maximally achievable) distances $3, 5, 5$, and 2, respectively, for the node R_3.

We finally also discuss the tables above the root node. The leftmost table is the result of the initialization step. For the bottom-up step from the left child ($= R_1$) to the root node, we observe that only the tuple $(2, 2, 0)$ of R_2^I has a join partner in R_1^I. Moreover, if we fix $x_2 = 2$ and $x_3 = 2$, then there exists only one possible extension to x_1 in R_1^I, namely $x_1 = 3$. Hence, the second table above node R_2 consists of a single pair and its initial distance (namely 0) cannot be increased by an extension to the left child. Now consider also the right child of R_2. We are only considering the pair from R_2^I where both tuples are the same, namely $(2, 2, 0)$. We see in the last table attached to R_3 that the maximum distance achievable by pairs where both tuples have $x_4 = 0$ is 3. Hence, this is then also the maximum distance achievable by the only pair in the last table attached to R_2. By tracing back top-down the pairs from the bottom-up traversal which contributed to the maximally achievable distance at the parent node, we get the pair $(3, 3, 2, 0, 0, 1, 5, 7)$ and $(3, 3, 2, 0, 0, 2, 8, 8)$ of query answers with maximum distance. ◊

3.1.2. *W[1]-Hardness.* Having proved XP-membership in combined complexity of the Diverse-ACQ problem in Theorem 3.2, we now show that, for any ws-monotone diversity measure, a stronger result in the form of FPT-membership is very unlikely to exist. More specifically, we prove W[1]-hardness for combined complexity in these cases. The reduction we use only takes polynomial time and, thus, we spontaneously prove NP-hardness of $\mathsf{Diverse}_{\mathsf{wsm}}$-ACQ in combined complexity when considering the problem unparameterized

Theorem 3.8. *The problem $\mathsf{Diverse}_{\mathsf{wsm}}$-ACQ, parameterized by the size k of the diversity set, is* W[1]*-hard in combined complexity. It remains* W[1]*-hard even if all relation symbols are of arity at most two and $Q(X)$ contains no existential variables. Furthermore, viewed as an unparameterized problem, $\mathsf{Diverse}_{\mathsf{wsm}}$-ACQ is* NP*-hard in combined complexity*

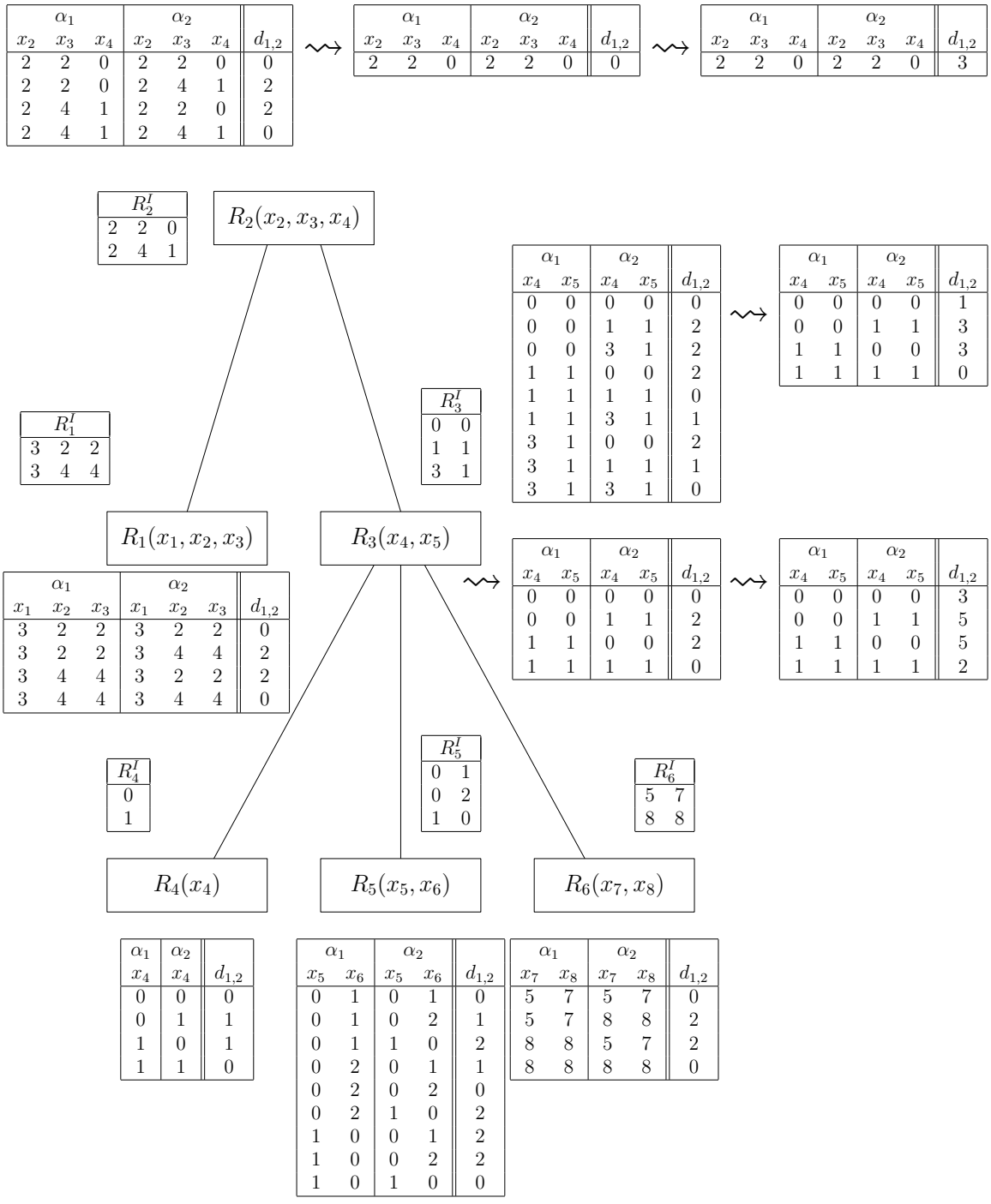

Figure 1: Example Execution of the Basic Algorithm.

Proof. We reduce from the Independent Set problem parameterized by the size of the independent set.

Let (G, s) be an arbitrary instance of Independent Set with $V(G) = \{v_1, \ldots, v_n\}$ and $E(G) = \{e_1, \ldots, e_m\}$. We define an instance $\langle I, Q, k, d\rangle$ of $\mathsf{Diverse}_{\mathsf{wsm}}$-$\mathsf{ACQ}$ as follows. The schema consists of a relation symbol R of arity one and m relation symbols $R_1, \ldots, R_m$ of arity two. The CQ $Q(X)$ is defined as

$$Q(v, x_1, \ldots, x_m) := R(v) \wedge R_1(v, x_1) \wedge \cdots \wedge R_m(v, x_m)$$

and the database instance I with $dom(I) = \{0, 1, \ldots, n\}$ is

$R^I = \{(i) : v_i \in V(G)\}$ and

$R^I_j = \{(i, i) : v_i \text{ is not incident to } e_j\} \cup \{(i, 0) : v_i \text{ is incident to } e_j\}$ for all $j \in \{1, \ldots, m\}$.

Finally, set $k = s$ and $d = f(m+1, \ldots, m+1)$, where f is the aggregator of δ_{wsm} aggregating the value $m + 1$ exactly $\binom{k}{2}$ times. Clearly, this reduction is feasible in polynomial time and the resulting problem instances satisfy all the restrictions stated in the theorem. The correctness of this reduction depends on two main observations.

O1) For each $i \in \{1, \ldots, n\}$, independently of G, there exists exactly one solution $\gamma_i \in Q(I)$ with $\gamma_i(v) = i$, and these are in fact the only solutions in $Q(I)$. Thus, there is a natural one-to-one association between vertices $v_i \in V(G)$ and solutions $\gamma_i \in Q(I)$.

O2) Due to ws-monotonicity, the desired diversity $d = f(m + 1, \ldots, m + 1)$ can only be achieved by k solutions that pairwisely differ on all variables.

Observation O1 is immediate: $R(\gamma(v)) \in R^I$ if and only if $\gamma(v) \in \{1, \ldots, n\}$, and for every $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$, there exists exactly one pair $(i, b) \in R^I_j$ (with b being either 0 or i). For Observation O2, note that the Hamming distance between two answers is at most $m + 1$. Thus, if two answers γ_i, γ_j are equal on some variable, $\Delta(\gamma_i, \gamma_j) < m + 1$, and hence due to ws-monotonicity, $\delta_{\mathsf{wsm}}(\gamma_1, \ldots, \gamma_k) = f((\Delta(\gamma_i, \gamma_j))_{1 \le i < j \le k}) < f(m+1, \ldots, m+1) = d$.

Observation O2 allows us to prove the correctness of the reduction by showing that G has an independent set of size s if and only if there exists a diversity set of size k where all answers differ pairwise on all variables.

To do so, first assume that there exists an independent set $S \subseteq V(G)$ of size s in G. We define the diversity set as $D = \{\gamma_i : v_i \in S\}$. By Observation O1, D is well-defined and thus contains k answers. We show that for any two distinct solutions $\gamma_i, \gamma_j \in D$ we have $\gamma_i(x) \neq \gamma_j(x)$ for all variables $x \in var(Q)$. To do so, first note that for all solutions $\gamma \in Q(I)$ and all variables $x \in var(Q)$, the fact that $\gamma(x) \neq 0$ implies $\gamma(x) = \gamma(v)$. Because of Observation O1, this implies that $\gamma_i(x) = \gamma_j(x)$ is only possible if $\gamma_i(x) = \gamma_j(x) = 0$ (since O1 implies that there do not exist any two distinct solutions $\gamma_i, \gamma_j \in Q(I)$ with $\gamma_i(v) = \gamma_j(v)$). Thus, towards a contradiction, assume $\gamma_i(x_\ell) = \gamma_j(x_\ell) = 0$ for some $\ell \in \{1, \ldots, m\}$. Then both, $(i, 0)$ and $(j, 0)$ must be contained in R_ℓ, and by the definition of I this implies that both, v_i and v_j are incident to e_ℓ. This however contradicts that v_i and v_j are both part of the same independent set, which proves that any two solutions $\gamma_i \neq \gamma_j$ must differ on all variables.

Next assume that there exists a diversity set $D \subseteq Q(I)$ of size k such that all distinct answers in D differ on all variables. We define a set $S = \{v_i : \gamma_i \in D\}$. By Observation O1, this set is well-defined and contains exactly k vertices. Towards a contradiction, assume that S contains two adjacent vertices $v_i \neq v_j$ and let e_ℓ be the edge connecting v_i and v_j. By definition of I, we get $(i, 0) \in R^I$ and $(j, 0) \in R^I$. However, this implies $\gamma_i(x_\ell) = 0 = \gamma_j(x_\ell)$.

This, however, contradicts the assumption that all solutions differ pairwise on all variables, which concludes the proof. □

3.1.3. *Speeding up the Basic Algorithm.* Algorithm 3.1 works for any polynomial-time computable diversity measures δ. To compute the diversity at the root node, we needed to distinguish between all the possible values for $d_{i,j}$ $(1 \leq i < j \leq k)$, which heavily increases the size of the sets D_t. The reason we had to explicitly distinguish all these values in the basic algorithm is that, in general, given two collections $(\gamma_1, \ldots, \gamma_k)$ and $(\gamma'_1, \ldots, \gamma'_k)$ of mappings that agree on the shared variables, we cannot derive $\delta(\hat{\gamma}_1, \ldots, \hat{\gamma}_k)$ for $\hat{\gamma}_i = \gamma_i \cup \gamma'_i$ from $\delta(\gamma_1, \ldots, \gamma_k)$ and $\delta(\gamma'_1, \ldots, \gamma'_k)$. However, for specific diversity measures, this is possible. As a result, significantly less information needs to be maintained, as will now be exemplified for δ_{sum}.

Theorem 3.9. *The Diverse$_{\mathsf{sum}}$-ACQ problem is in* FPT *in query complexity when parameterized by the size k of the diversity set. More specifically, Diverse$_{\mathsf{sum}}$-ACQ for an ACQ $Q(X)$, a database instance I, and integers k and d, can be solved in time $\mathcal{O}(|R^I|^{2k} \cdot 2^{k(k-1)} \cdot poly(|Q|, k))$, where R^I is the relation from I with the most tuples and $poly(|Q|, k)$ is a polynomial in $|Q|$ and k.*

Proof. Note that $poly(|Q|, k)$ is the same as in Theorem 3.2. For query complexity, the size $|R^I|$ of a relation in I is considered as constant. Hence, the above-stated upper bound on the asymptotic complexity indeed entails FPT-membership. To prove this upper bound, the crucial property is that for a collection of mappings $\gamma_1, \ldots, \gamma_k$ over variables Z, the equality $\delta_{\mathsf{sum}}(\gamma_1, \ldots, \gamma_k) = \sum_{z \in Z} \delta_{\mathsf{sum}}(\gamma_1|_z, \ldots, \gamma_k|_z)$ holds. Hence, in principle, it suffices to store in $D_{T'}$ for each collection $(\alpha_1, \ldots, \alpha_k)$ with $\alpha_i \in t(I)$ (t being the root of T') such that there exists $\gamma_i \in T'(I)$ with $\gamma_i \cong \alpha_i$ (for all $1 \leq i \leq k$) the value

$$d_{T'}(\alpha_1, \ldots, \alpha_k) = \max_{\substack{\gamma_1, \ldots, \gamma_k \in T'(I) \\ \text{s.t. } \gamma_i \cong \alpha_i \text{ for all } i}} \delta_{\mathsf{sum}}(\gamma_1|_X, \ldots, \gamma_k|_X).$$

I.e., each entry in $D_{T'}$ now is of the form $(\alpha_1, \ldots, \alpha_k, v)$ with $v = d_{T'}(\alpha_1, \ldots, \alpha_k)$. In the bottom-up traversal step of the algorithm, when updating some D_t to D_t^{new} by merging $D_{t'}$, for every entry $(\alpha_1, \ldots, \alpha_k, v) \in D_t$ there exists an entry $(\alpha_1, \ldots, \alpha_k, \bar{v}) \in D_t^{new}$ if and only if there exists at least one $(\alpha'_1, \ldots, \alpha'_k, v') \in D_{t'}$ such that $\alpha_i \cong \alpha'_i$ for $1 \leq i \leq k$. Then $\bar{v}$ is

$$\bar{v} = \max_{\substack{(\alpha'_1, \ldots, \alpha'_k, v') \in D_{t'} \\ \text{s.t. } \alpha_i \cong \alpha'_i \text{ for all } i}} (v + v' - \delta_{\mathsf{sum}}((\alpha_1 \cap \alpha'_1)|_X, \ldots, (\alpha_k \cap \alpha'_k)|_X)).$$

In order to make sure that the answer tuples in the final diversity set are pairwise distinct, the following additional information must be maintained at each $D_{T'}$: from the partial solutions $\alpha_1, \ldots, \alpha_k$ it is not possible to determine whether the set of extensions $\gamma_1, \ldots, \gamma_k$ contains duplicates or not. Thus, similar to the original values $d_{i,j}$ describing the pairwise diversity of partial solutions, we now include binary values $b_{i,j}$ for $1 \leq i < j \leq k$ that indicate whether extensions γ_i and γ_j of α_i and α_j to $var(T')$ differ on at least one variable of X ($b_{i,j} = 1$) or not in order to be part of $ext_{T'}(e)$. This increases the maximal size of $D_{T'}$ to $|R^I|^{2k} \cdot 2^{k(k-1)}$. The bottom-up traversal step can be easily adapted to consider in the computation of $\bar{v}$ for an entry in D_t^{new} only those entries from D_t and $D_{t'}$ that are consistent with the values of $b_{i,j}$, giving the stated running time. □

Actually, if we drop the condition that the answer tuples in the final diversity set must be pairwise distinct, the query complexity of $\mathsf{Diverse}_{\mathsf{sum}}$-ACQ can be further reduced. Clearly, in this case, we can drop the binary values $b_{i,j}$ for $1 \leq i < j \leq k$ from the entries in $D_{T'}$, which results in a reduction of the asymptotic complexity to $\mathcal{O}(|R^I|^{2k} \cdot poly(|Q|, k))$. At first glance, this does not seem to improve on the FPT-membership result. However, a further, generally applicable improvement (not restricted to a particular aggregate function and not restricted to query complexity) is possible via the observation that the basic algorithm computes (and manages) redundant information: for an arbitrary node $t \in V(T)$ and set D_t, if D_t contains an entry of the form $(\alpha_1, \ldots, \alpha_k, \ldots)$, then D_t also contains entries of the form $(\alpha_{\pi(1)}, \ldots, \alpha_{\pi(k)}, \ldots)$ for all permutations π of $(1, \ldots, k)$. But we are ultimately interested in *sets* of answer tuples and do not distinguish between permutations of the members inside a set. Keeping these redundant entries made the algorithm conceptually simpler and had no significant impact on the running times (especially since we assume k to be small compared to the size of the relations in I). However, given the improvements for $\mathsf{Diverse}_{\mathsf{sum}}$-ACQ from Theorem 3.9 and dropping the binary values $b_{i,j}$ for $1 \leq i < j \leq k$ from the entries in D_t, we can get a significantly better complexity classification:

Theorem 3.10. *The problem $\mathsf{Diverse}_{\mathsf{sum}}$-ACQ is in* P *in query complexity when the diversity set may contain duplicates and k is given in unary.*

Proof. We claim the number of rows in D_t for any $t \in V(T)$ to be in $\mathcal{O}(k^{|t(I)|-1})$. In the following, we verify the claim.

To remove redundant rows from the sets D_t, we introduce some order $\preceq$ on partial solutions $\alpha \in t(I)$ for each $t \in V(T)$ (e.g. based on some order on the domain elements), and only consider such collections $\alpha_1, \ldots, \alpha_k \in t(I)$ where $\alpha_1 \preceq \cdots \preceq \alpha_k$ together with the value $d_{T'}(\alpha_1, \ldots, \alpha_k)$. Thus the number of such different collections is described by $\binom{|t(I)|+k-1}{k}$. Applying basic combinatorics we get

$$\binom{|t(I)|+k-1}{k} = \binom{|t(I)|+k-1}{(|t(I)|+k-1)-k} = \binom{|t(I)|+k-1}{|t(I)|-1}.$$

By definition, this is the same as

$$\frac{(|t(I)|+k-1) \cdot (|t(I)|+k-2) \cdot \ldots \cdot (k+1)}{(|t(I)|-1)!} \leq (|t(I)|+k)^{|t(I)|-1}.$$

Since we assume query complexity, we consider the size of I to be a constant. Thus, since $\lambda(t)$ consists of a single atom, also $|t(I)|$ can be considered to be a constant. As a result we have that $(|t(I)|+k)^{|t(I)|-1}$ is in $\mathcal{O}(k^{|t(I)|-1})$ as claimed. □

3.2. **Data Complexity.** We now inspect the data complexity of Diverse-ACQ both from the parameterized and non-parameterized point of view. For the parameterized case, we will improve the XP-membership result from Theorem 3.2 (for combined complexity) to FPT-membership for arbitrary monotone aggregate functions. Actually, by considering the query as fixed, we now allow arbitrary FO-queries, whose evaluation is well-known to be feasible in polynomial time (data complexity) [Var82]. Thus, as a preprocessing step, we can evaluate Q and store the result in a table R^I. We may therefore assume w.l.o.g. that the query is of the form $Q(x_1, \ldots, x_m) := R(x_1, \ldots, x_m)$ and the database I consists of a single relation R^I.

To show FPT-membership, we apply a problem reduction that allows us to iteratively reduce the size of the database instance until it is bounded by a function of m and k, i.e., the query and the parameter. Let $X = \{x_1, \ldots, x_m\}$ and define $\binom{X}{s} := \{Z \subseteq X : |Z| = s\}$ for $s \in \{0, \ldots, m\}$. Moreover, for every assignment $\alpha \colon Z \to dom(I)$ with $Z \subseteq X$ let $Q(I)_\alpha := \{\gamma \in Q(I) \colon \gamma \cong \alpha\}$, i.e., the set of answer tuples that coincide with α on Z. The key to our problem reduction is applying the following reduction rule $\mathbf{Red}_t$ for $t \in \{1, \ldots, m\}$ exhaustively in order $\mathbf{Red}_1$ through $\mathbf{Red}_m$:

($\mathbf{Red}_t$) If for some $\alpha \colon Z \to dom(I)$ with $Z \in \binom{X}{m-t}$, the set $Q(I)_\alpha$ has at least $t!^2 \cdot k^t$ elements, then do the following: select (arbitrarily) $t \cdot k$ solutions $\Gamma \subseteq Q(I)_\alpha$ that pairwisely differ on all variables in $X \setminus Z$. Then remove the tuples corresponding to assignments $Q(I)_\alpha \setminus \Gamma$ from R^I.

The intuition of the reduction rule is best seen by looking at $\mathbf{Red}_1$. Our ultimate goal is to achieve maximum diversity by selecting k answer tuples. Now suppose that we fix all but 1 position – say x_1 – in the answer relation R^I to be equal to some assignment α. Furthermore, let $Q(I)_\alpha \subseteq R^I$ be the matching tuples and $\Gamma \subseteq Q(I)_\alpha$ be k-many of these matching tuples, chosen arbitrarily. Now, given a diversity set $D \subseteq Q(I)$, we claim that we can replace every element $\gamma \in D \cap (Q(I)_\alpha \setminus \Gamma)$ with an element in Γ while preserving optimality. This means that it is safe to remove $Q(I)_\alpha \setminus \Gamma$ from R^I. The claim holds as γ and Γ agree on all positions but x. Thus, we only need to find a $\gamma' \in \Gamma$ that differs from each element in $D \setminus \{\gamma\}$ on x as then γ' is at least as far away from those elements as γ is. Such an element always exists due to Γ containing more elements than $D \setminus \{\gamma\}$ has unique x-values.

This can be generalized to fixing fewer positions but the intuition stays the same. When fixing $m - t$ positions, there is also no need to retain all different value combinations in the remaining t positions. Concretely, if there exist at least $t!^2 \cdot k^t$ different value combinations (possibly sharing values on some positions), there also exist $t \cdot k$ tuples with pairwise maximum Hamming distance on the remaining t positions (no shared values pairwise) and it is sufficient to only keep those. Note that here a recursive argument is needed to ensure the existence of the $t \cdot k$ pairwise maximally distant tuples and, hence, it is necessary to first apply $\mathbf{Red}_{t-1}$ exhaustively before we can continue with $\mathbf{Red}_t$.

Formally, the crucial properties of the reduction rule $\mathbf{Red}_t$ with $t \in \{1, \ldots, m\}$ is as follows:

Lemma 3.11. *Let Q be a CQ of the form $Q(x_1, \ldots, x_m) := R(x_1, \ldots, x_m)$, I a corresponding database, $t \in \{1, \ldots, m\}$ and suppose that all sets $Q(I)_{\alpha'}$ with $\alpha' \colon Z' \to dom(I)$ and $Z' \in \binom{X}{m-(t-1)}$ have cardinality at most $(t-1)!^2 \cdot k^{t-1}$. Then the reduction rule $\mathbf{Red}_t$ is well-defined and safe. That is:*

- "well-defined". *If for some $\alpha : Z \to dom(I)$ with $Z \in \binom{X}{m-t}$, the set $Q(I)_\alpha$ has at least $t!^2 \cdot k^t$ elements, then there exist at least $t \cdot k$ solutions $\Gamma \subseteq Q(I)_\alpha$ that pairwisely differ on all variables in $X \setminus Z$.*
- "safe". *Let I_{old} denote the database instance before an application of $\mathbf{Red}_t$ and let I_{new} denote its state after applying $\mathbf{Red}_t$. Let $\gamma_1, \ldots, \gamma_k$ be pairwise distinct solutions in $Q(I_{old})$. Then there exist pairwise distinct solutions $\gamma'_1, \ldots, \gamma'_k$ in $Q(I_{new})$ with $\delta(\gamma'_1, \ldots, \gamma'_k) \geq \delta(\gamma_1, \ldots, \gamma_k)$, i.e., the diversity achievable before deleting tuples from the database can still be achieved after the deletion.*

Moreover, a set of $t \cdot k$ solutions $\Gamma \subseteq Q(I)_\alpha$ that pairwisely differ on all variables in $X \setminus Z$ can be computed by iteratively choosing solutions γ_i (for $i \in \{1, \ldots, t \cdot k\}$) arbitrarily from $Q(I)_\alpha$ that differ from all solutions $\gamma_1, \ldots, \gamma_{i-1}$ on all variables in $X \setminus Z$.

Proof. Let $t \in \{1, \ldots, m\}$ and suppose that all sets $Q(I)_{\alpha'}$ with $\alpha' \colon Z' \to dom(I)$ and $Z' \in \binom{X}{m-(t-1)}$ have cardinality at most $(t-1)!^2 \cdot k^{t-1}$.

"well-defined". Let α be of the form $\alpha : Z \to dom(I)$ with $Z \in \binom{X}{m-t}$ and assume that $|Q(I)_\alpha| > t!^2 \cdot k^t$. For arbitrary $\gamma \in Q(I)_\alpha$, we define the set C_γ as

$$C_\gamma := \{\gamma' \in Q(I)_\alpha : \Delta(\gamma, \gamma') < t\},$$

i.e., C_γ contains the solutions whose distance from γ is less than t or, equivalently, that agree with γ on at least one variable from $X \setminus Z$. Hence, we have

$$C_\gamma = \bigcup_{x \in X \setminus Z} Q(I)_{\alpha \cup \{x \mapsto \gamma(x)\}}$$

and thus, the size of C_γ is at most $t \cdot (t-1)!^2 \cdot k^{t-1}$ by the assumption of the lemma.

Now, iteratively select elements γ_i for $i \in \{1, \ldots, t \cdot k\}$ with $\gamma_i \in Q(I)_\alpha \setminus \bigcup_{j=1}^{i-1} C_{\gamma_j}$, i.e., arbitrarily choose $\gamma_1 \in Q(I)_\alpha$, then $\gamma_2 \in Q(I)_\alpha \setminus C_{\gamma_1}$, then $\gamma_3 \in Q(I)_\alpha \setminus (C_{\gamma_1} \cup C_{\gamma_2})$, etc.

We claim that such elements γ_i for $i \in \{1, \ldots, t \cdot k\}$ indeed exist, i.e., for every $i \in \{1, \ldots, t \cdot k\}$, $|Q(I)_\alpha \setminus \bigcup_{j=1}^{i-1} C_{\gamma_j}| > 0$. Indeed, by the assumption $|Q(I)_\alpha| \geq t!^2 \cdot k^t$ and the above considerations on the size of C_γ for arbitrary γ, we have:

$$|Q(I)_\alpha \setminus \bigcup_{j=1}^{i-1} C_{\gamma_j}| \geq t!^2 \cdot k^t - (i-1) \cdot t \cdot (t-1)!^2 \cdot k^{t-1} > t!^2 \cdot k^t - (t \cdot k) \cdot t \cdot (t-1)!^2 \cdot k^{t-1} = 0.$$

Now set $\Gamma = \{\gamma_1, \ldots, \gamma_{t \cdot k}\} \subseteq Q(I)_\alpha$. By the construction, we have that γ_i differs from γ_j for $j < i$ on all variables $X \setminus Z$ as $\gamma_i \notin C_{\gamma_j}$. Hence, $\mathbf{Red}_t$ is well-defined, i.e., the desired $t \cdot k$ solutions indeed exist.

Moreover, the proof also demonstrates that the set Γ can be constructed by starting with one solution γ_1 and then iteratively adding arbitrary solutions γ_i that just need to differ from all solutions $\gamma_1, \ldots, \gamma_{i-1}$ selected so far.

"safe". Let I_{old} denote the database instance before applying $\mathbf{Red}_t$ and let I_{new} denote its state after an application of $\mathbf{Red}_t$, i.e., $Q(I_{new}) = (Q(I_{old}) \setminus Q(I_{old})_\alpha) \cup \Gamma$. Now consider arbitrary pairwise distinct solutions $\gamma_1, \ldots, \gamma_k \in Q(I_{old})$. We have to show that there exist pairwise distinct solutions $\gamma'_1, \ldots, \gamma'_k$ in $Q(I_{new})$ with $\delta(\gamma'_1, \ldots, \gamma'_k) \geq \delta(\gamma_1, \ldots, \gamma_k)$.

Assume that, for some $i \in \{1, \ldots, k\}$, γ_i gets removed by $\mathbf{Red}_t$, i.e., $\gamma_i \in Q(I_{old})_\alpha \setminus \Gamma$. We claim that there exists $\gamma'_i \in \Gamma \subseteq Q(I_{new})$ with $\delta(\gamma_1, \ldots, \gamma_{i-1}, \gamma'_i, \gamma_{i+1}, \ldots, \gamma_k) \geq \delta(\gamma_1, \ldots, \gamma_k)$ and is different to $\gamma_1, \ldots, \gamma_{i-1}, \gamma_{i+1}, \ldots, \gamma_k$.

For arbitrary $j \neq i$, we define the set $\Gamma_j \subseteq \Gamma$ as $\Gamma_j = \{\gamma' \in \Gamma : \Delta(\gamma', \gamma_j) < \Delta(\gamma_i, \gamma_j)\}$, i.e., Γ_j contains those elements of Γ whose distance from γ_j is smaller than the distance between γ_i and γ_j. We will show below that $|\Gamma_j| \leq t$ holds. In this case, we have

$$|\Gamma \setminus \bigcup_{i \neq j} \Gamma_j| \geq t \cdot k - t \cdot (k-1) = t \geq 1.$$

That is, $\Gamma \setminus \bigcup_{i \neq j} \Gamma_j \neq \emptyset$. In other words, we can choose a solution γ'_i from Γ that differs from all γ_j at least as much as γ_i did. Hence, such γ'_i indeed has the property $\delta(\gamma_1, \ldots, \gamma_{i-1}, \gamma'_i, \gamma_{i+1}, \gamma_k) \geq \delta(\gamma_1, \ldots, \gamma_k)$ and is different to $\gamma_1, \ldots, \gamma_{i-1}, \gamma_{i+1}, \ldots, \gamma_k$. By

iterating this argument for every $i \in \{1, \ldots, k\}$, we may conclude that there exist pairwise distinct solutions $\gamma'_1, \ldots, \gamma'_k \in I_{new}$ with $\delta(\gamma'_1, \ldots, \gamma'_k) \geq \delta(\gamma_1, \ldots, \gamma_k)$.

It only remains to show that $|\Gamma_j| \leq t$ indeed holds. As γ_i and any element $\gamma' \in \Gamma \subseteq Q(I_{old})_\alpha$ agree on the variables Z, a lower diversity can only be achieved by γ', if γ_j and γ' agree on some variable $x \in X \setminus Z$. We define

$$\Gamma_j^{(x)} = \{\gamma' \in \Gamma : \gamma'(x) = \gamma_j(x)\}.$$

Hence,

$$\Gamma_j \subseteq \bigcup_{x \in X \setminus Z} \Gamma_j^{(x)}.$$

Now, if some γ' is in $\Gamma_j^{(x)}$, all other $\gamma'' \in \Gamma, \gamma' \neq \gamma''$ are not in $\Gamma_j^{(x)}$ as γ' and γ'' differ on $x \in X \setminus Z$ by construction of Γ. Therefore, $|\Gamma_j^{(x)}| \leq 1$ and

$$|\Gamma_j| \leq \sum_{x \in X \setminus Z} |\Gamma_j^{(x)}| \leq |X \setminus Z| = t.$$

This completes the proof of the claim. □

With the reduction rule $\mathbf{Red}_t$ at our disposal, we can design an FPT-algorithm (data complexity) for $\mathsf{Diverse}_{\mathsf{mon}}$-ACQ and, more generally, for the $\mathsf{Diverse}_{\mathsf{mon}}$-FO problem:

Theorem 3.12. *The problem $\mathsf{Diverse}_{\mathsf{mon}}$-FO is in FPT in data complexity when parameterized by the size k of the diversity set. More specifically, an instance $\langle I, Q, k, d\rangle$ of $\mathsf{Diverse}_{\mathsf{mon}}$-FO with m-ary FO-query Q can be reduced in polynomial time (data complexity) to an equivalent instance $\langle I', Q', k, d\rangle$ of $\mathsf{Diverse}_{\mathsf{mon}}$-FO of size $\mathcal{O}(m!^2 \cdot k^m)$.*

Proof. Recall that we may assume that query Q is of the form $Q(x_1, \ldots, x_m) := R(x_1, \ldots, x_m)$ and the database I consists of a single relation R^I. We apply $\mathbf{Red}_1$ through $\mathbf{Red}_m$ to I in this order exhaustively. Initially, we have to check the preconditions of Lemma 3.11 for $t = 1$ for us to safely apply $\mathbf{Red}_1$. Thus, let us consider $Z \in \binom{X}{m}$. We have $Z = X$ and hence, for every $\alpha : Z \to dom(I) \in Q(I)$, we have $Q(I)_\alpha = \{\alpha\}$. In particular, $|Q(I)_\alpha| = 1 \leq 0 \cdot k + k^0$. Hence, the preconditions of Lemma 3.11 are fulfilled and exhaustive application of $\mathbf{Red}_1$ does not alter the status of the $\mathsf{Diverse}_{\mathsf{mon}}$-FO problem. After exhaustive application of $\mathbf{Red}_1$, if now $\mathbf{Red}_2$ is applicable, then the preconditions of Lemma 3.11 are fulfilled and exhaustive application of $\mathbf{Red}_2$ does not alter the status of the $\mathsf{Diverse}_{\mathsf{mon}}$-FO problem, etc.

Finally, after exhaustive application of $\mathbf{Red}_m$, let I^* denote the resulting database instance. Note that, for $t = m$, we have $\binom{X}{0} := \{Z \subseteq X : |Z| = 0\} = \{\emptyset\}$. and $|Q(I^*)_\alpha| \leq m!^2 \cdot k^m$ for any $\alpha : \emptyset \to dom(I^*)$. In particular, this means that such an α does not bind any variables in X. Hence, $Q(I^*)_\alpha = Q(I^*)$ and, therefore, $|Q(I^*)| \leq m!^2 \cdot k^m$. By the form of Q (with a single atom) and I^* (with a single relation), this means I^* is of size $\mathcal{O}(m!^2 \cdot k^m)$.

It remains to show that the exhaustive application of $\mathbf{Red}_1$ through $\mathbf{Red}_m$ is feasible in polynomial time data complexity. In total, we have to consider at most 2^m sets $Z \subseteq X$ of variables with $|Z| = m - t$ for $t \in \{1, \ldots, m\}$ and check if $\mathbf{Red}_t$ is applicable.

For each Z, if the reduction rule is applicable, the following computation is carried out. Let $Z = \{z_1, \ldots, z_{m-t}\}$ and $X \setminus Z = \{z_{m-t+1}, \ldots, z_m\}$. Moreover, let $S \subseteq R^I$ denote the subset of answer tuples that are still left after previous applications of the reduction rule. Then we order S lexicographically for this variable order. That is, tuples with the

same value combination on Z occur in contiguous positions. In a single pass of the ordered instance S we inspect, for each value combination α on Z, the set $S_\alpha \subseteq S$ of tuples with precisely this value combination α on Z. If $|S_\alpha| < t!^2 \cdot k^t$, then we do nothing. Otherwise, we select $t \cdot k$ tuples from S_α. By the last property of Lemma 3.11, we can apply the following steps: choose the first tuple $\gamma_1 \in S_\alpha$; then, for every $i \in \{2, \dots, t \cdot k\}$, further scan S_α until a tuple $\gamma_i \in S_\alpha$ is found that differs from all tuples $\gamma_1, \dots, \gamma_{i-1}$ on all variables $X \setminus Z$. Since Lemma 3.11 guarantees that we can just pick suitable solutions in an arbitrary order, this approach is guaranteed to produce the required result. Let $\Gamma = \{\gamma_1, \dots, \gamma_{t \cdot k}\}$. We may then delete all tuples in $S_\alpha \setminus \Gamma$ from S.

The total effort for the exhaustive application of the reduction rule $\mathbf{Red}_t$ for $t \in \{1, \dots, m\}$ is obtained by the following considerations:

- Evaluating the original, general FO-formula over the original database instance is feasible in polynomial time data complexity. Also, the size of the resulting answer relation R^I is of course bounded by this polynomial. Let us denote it by p.
- There is an "outer loop" over subsets $Z \subseteq X$. There are 2^m subsets, where m depends only on the query, which is considered as constant in data complexity.
- Inside this loop, we first sort the set of remaining answer tuples $S \subseteq R^I$. The effort for this step is bounded by $\mathcal{O}(p \cdot \log(p) \cdot m)$.
- One pass of (the ordered set) S has cost $\leq p$.
- For each S_α, we check if $|S_\alpha| \geq t!^2 \cdot k^t$. If this is the case, we select $t \cdot k$ tuples from S_α in a single pass of S_α. This step is feasible in time $\mathcal{O}(p \cdot t \cdot k \cdot m)$ – including also the cost for checking if the currently scanned tuple in S_α differs on all variables in $X \setminus Z$ from the already selected tuples γ_i.
- The deletion of the tuples in $S_\alpha \setminus \Gamma$ from S can be done by first of all marking them as deleted when constructing the set Γ. When all α's have been processed, we can actually delete the marked tuples from S by yet another pass of S, which clearly fits into $\mathcal{O}(p)$ time. □

We now study the data complexity of the Diverse-ACQ problem in the non-parameterized case, i.e., the size k of the diversity set is part of the input and no longer considered as the parameter. It will turn out that this problem is NP-hard for any ws-monotone diversity measure. Our NP-hardness proof will be by reduction from the Independent Set problem, where we restrict the instances to graphs of degree at most 3. It was shown in [AK97] that this restricted problem remains NP-complete.

Theorem 3.13. *The problem Diverse$_{\mathsf{wsm}}$-ACQ is* NP*-hard in data complexity. It is* NP*-complete if the size k of the diversity set is given in unary.*

Proof. The NP-membership is immediate: compute $Q(I)$ (which is feasible in polynomial time when considering the query as fixed), then guess a subset $S \subseteq Q(I)$ of size k and check in polynomial time that S has the desired diversity.

We prove hardness by reduction from a restricted version of the Independent Set problem, where we assume all instances of graphs to be of degree at most 3. Before we present the reduction, let us briefly look at this restriction.

It is easy to see that Independent Set remains NP-hard if we restrict the degree of vertices to 4. This result is obtained by combining two classical results [Pap94]: first, 3-SAT remains NP-hard even if every variable in the propositional formulas occurs at most 3 times and each literal occurs at most 2 times. And second, we apply the reduction from 3-SAT to

Independent Set where each clause is represented by a triangle and any two dual literals are connected by an edge. Hence, in the resulting graph, each vertex is adjacent two at most 4 vertices (the other 2 vertices in the triangle plus at most 2 vertices corresponding to dual literals). This result was strengthened in [AK97] where it was shown that we may even further restrict the degree of vertices to 3. The idea of Alimonnti and Kann [AK97] is to apply the following transformation for each vertex of degree greater than 3: suppose that v has degree greater than 3; then replace v by a path v_1, v_2, v_3, where 2 edges containing v are connected to v_1 and the remaining edges of v are connected to v_3. Thus, v_1 and v_2 have degree less than or equal to 3 while the degree of v_3 is strictly less than the degree of v. Furthermore, the original graph has an independent set of size k if and only if the new one has an independent set of size $k+1$ as picking v_1 and v_3 corresponds to picking v. Exhaustive application of this transformation yields an instance of Independent Set where every vertex in the graph has degree ≤ 3.

For the NP-hardness, we define the query Q independently of the instance of the Independent Set problem as $Q(x_1, x_2, x_3, x_4, x_5) := R(x_1, x_2, x_3, x_4, x_5)$, i.e., the only relation symbol R has arity 5. Now let (G, s) be an instance of Independent Set where each vertex of G has degree at most 3.

Let $V(G) = \{v_1, \ldots, v_n\}$ and $E(G) = \{e_1, \ldots, e_m\}$. Then the database I consists of a single relation R^I with n tuples (= number of vertices in G) over the domain $dom(I) = \{\mathbf{free}_1, \ldots, \mathbf{free}_n, \mathbf{taken}_1, \ldots, \mathbf{taken}_m\}$. The i-th tuple in R^I will be denoted $(e_{i,1}, \ldots, e_{i,5})$. For each $v_i \in V(G)$, the values $e_{i,1}, \ldots, e_{i,5} \in dom(I)$ are defined by an iterative process:

(1) The iterative process starts by initializing all $e_{i,1}, \ldots, e_{i,5}$ to $\mathbf{free}_i$ for each $v_i \in V(G)$.
(2) We then iterate through all edges $e_j \in E(G)$ and do the following: Let v_i and $v_{i'}$ be the two incident vertices to e_j and let $t \in \{1, \ldots, 5\}$ be an index such that $e_{i,t}$ and $e_{i',t}$ both still have the values $\mathbf{free}_i$ and $\mathbf{free}_{i'}$, respectively. Then set both $e_{i,t}$ and $e_{i',t}$ to $\mathbf{taken}_j$.

In the second step above when processing an edge e_j, such an index t must always exist. This is due to the fact that, at the moment of considering e_j, the vertex v_i has been considered at most twice (the degree of v_i is at most 3) and thus, for at least three different values of $t \in \{1, \ldots, 5\}$, the value $e_{i,t}$ is still set to $\mathbf{free}_i$. Analogous considerations apply to vertex $v_{i'}$ and thus, for at least 3 values of $t \in \{1, \ldots, 5\}$, we have $e_{i',t} = \mathbf{free}_{i'}$. Hence, by the pigeonhole principle, there exists $t \in \{1, \ldots, 5\}$ with $e_{i,t} = \mathbf{free}_i$ and $e_{i',t} = \mathbf{free}_{i'}$.

After the iterative process, the database I is defined by $R^I = \{(e_{i,1}, e_{i,2}, e_{i,3}, e_{i,4}, e_{i,5}) : i = 1, \ldots, n\}$. Moreover, the size of the desired diversity set is set to $k = s$ and the target diversity is set to $d = f(5, \ldots, 5)$, where f is the aggregator of δ_{wsm} aggregating the value 5 exactly $\binom{k}{2}$ times. The resulting problem instance is of the form $\langle I, Q, k, d\rangle$.

The reduction is clearly feasible in polynomial time. Its correctness, i.e., the graph $G = (V(G), E(G))$ having an independent set of size s if and only if there exists $S \subseteq Q(I)$ with $|S| = k$ and diversity $\geq d$ hinges on the observation that the desired diversity can only be reached by k answer tuples that pairwisely differ in all 5 positions due to ws-monotonicity. Furthermore, the answers $Q(I)$ are trivially $\{\gamma_1, \ldots, \gamma_n\}$ with $\gamma_i(x_t) = e_{i,t}$ for each $t \in \{1, \ldots, 5\}$. We thus have to show that these differ on all values if and only if G has an independent set of size $s = k$.

First, suppose that G has such an independent set, say $\{v_{i_1}, \ldots, v_{i_k}\}$. We claim that then $\{\gamma_{i_1}, \ldots, \gamma_{i_k}\}$ is a subset of $Q(I)$ with the desired diversity, i.e., any two answers γ_{i_r} and γ_{i_s} differ on all 5 variables. Suppose to the contrary that $\gamma_{i_r}(t) = \gamma_{i_s}(t)$ holds for some $t \in \{1, \ldots, 5\}$. By our construction of R^I, this can only happen if $\gamma_{i_r}(t) \neq \mathbf{free}_{i_r}$ and

$\gamma_{i_s}(t) \neq \mathbf{free}_{i_s}$. Hence, $\gamma_{i_r}(t) = \gamma_{i_s}(t) = \mathbf{taken}_j$ for some $j \in \{1, \dots, m\}$ holds. Again by our construction of R^I, this means that both v_{i_r} and v_{i_s} are incident to the edge e_j. This contradicts the assumption that both v_{i_r} and v_{i_s} are contained in an independent set.

Conversely, suppose that there exists a subset $S \subseteq Q(I)$ of size k with the desired target diversity. Let $S = \{\gamma_{i_1}, \dots, \gamma_{i_k}\}$. We claim that then $\{v_{i_1}, \dots, v_{i_k}\}$ is an independent set of G. Suppose to the contrary that it is not, i.e., two vertices v_{i_r} and v_{i_s} are incident to the same edge e_j. Then, by our construction of I, there exists $t \in \{1, \dots, 5\}$ with $\gamma_{i_r}(t) = \gamma_{i_s}(t) = \mathbf{taken}_j$. This means that $\Delta(\gamma_{i_r}, \gamma_{i_s}) < 5$ and hence, the target diversity $f(5, \dots, 5)$ cannot be reached by S due to ws-monotonicity, which is a contradiction. □

4. Diversity of Unions of Conjunctive Queries

We now turn our attention to UCQs. Of course, all hardness results proved for CQs and ACQs in Section 3 carry over to UCQs and UACQs, respectively. Moreover, the FPT-membership result from Theorem 3.12 for general FO-formulas of course also includes UCQs. It remains to study the query complexity and combined complexity of UACQs. It turns out that the union makes the problem significantly harder than for ACQs and we are not able to establish XP-membership. Instead, we show next that $\mathsf{Diverse}_{\mathsf{wsm}}$-UACQ is NP-hard even in a very restricted setting, namely where we are looking for a pair of diverse answers to a union of two ACQs over a fixed database. Put differently, $\mathsf{Diverse}_{\mathsf{wsm}}$-UACQ is NP-hard in query complexity even when fixing the parameter to $k = 2$, making the existence of an XP-algorithm unlikely.

The proof will be by reduction from a variant of the List Coloring problem, which we introduce next: A *list assignment* C assigns each vertex v of a graph G a list of colors $C(v) \subseteq \{1, \dots, l\}, l \in \mathbb{N}$. Then a *coloring* is a function $c : V(G) \to \{1, \dots, l\}$ and it is called *$C-$admissible* if each vertex $v \in V(G)$ is colored in a color of its list, i.e., $c(v) \in C(v)$, and adjacent vertices $u, v \in E(G)$ are colored with different colors, i.e., $c(u) \neq c(v)$. Formally, the problem is defined as follows:

List Coloring
Input: A graph G, an integer $l \in \mathbb{N}$, and a list assignment $C : V(G) \to 2^{\{1,\dots,l\}}$.
Question: Does there exist a C-admissible coloring $c : V(G) \to \{1, \dots, l\}$?

Clearly, List Coloring is a generalization of 3-Colorability and, hence, NP-complete. It was shown in [CC06], that the List Coloring problem remains NP-hard even when assuming that each vertex of G has degree 3, G is bipartite, and $l = 3$. This restriction will be used in the proof of the following theorem. Note that these restrictions also imply that both parts of the bipartition are of the same size.

Theorem 4.1. *The problem $\mathsf{Diverse}_{\mathsf{wsm}}$-UACQ is NP-hard in query complexity (and hence, also in combined complexity). It remains NP-hard even if the desired size of the diversity set is bounded by 2 and the UACQs are restricted to containing at most two CQs and no existential variables. The problem is NP-complete if the size k of the diversity set is given in unary.*

Proof. The NP-membership in case of k given in unary is immediate: guess k assignments to the free variables of query Q, check in polynomial time that they are solutions, and verify in polynomial time that their diversity is above the desired threshold.

For our problem reduction, we consider a fixed database I over a fixed schema, which consists of the 4 domain elements $dom(I) = \{0, 1, 2, 3\}$ and 9 relation symbols

$$R_{\{1\}}, R_{\{2\}}, R_{\{3\}}, R_{\{1,2\}}, R_{\{1,3\}}, R_{\{2,3\}}, R_{\{1,2,3\}}, S, S'.$$

The relations of the database are defined as follows:

$$\begin{aligned} R^I_{\{1\}} &= \{(1,1,1)\}, & R^I_{\{1,2\}} &= \{(1,1,1),(2,2,2)\},\\ R^I_{\{2\}} &= \{(2,2,2)\}, & R^I_{\{1,3\}} &= \{(1,1,1),(3,3,3)\},\\ R^I_{\{3\}} &= \{(3,3,3)\}, & R^I_{\{2,3\}} &= \{(2,2,2),(3,3,3)\},\\ R^I_{\{1,2,3\}} &= \{(1,1,1),(2,2,2),(3,3,3)\}, & S^I &= \{(0)\}, & S'^I &= \{(1)\}. \end{aligned}$$

Now let $\langle G, l, C\rangle$ be an arbitrary instance of List Coloring, where each vertex of G has degree 3, G is bipartite, and $l = 3$. That is, G is of the form $G = (V \cup V', E)$ for vertex sets V, V' and edge set E with $V = \{v_1, \ldots, v_n\}$, $V' = \{v'_1, \ldots, v'_n\}$, and $E = \{e_1, \ldots, e_{3n}\}$. Note that $|V| = |V'|$ and $|E| = 3 \cdot |V|$ as each vertex in G has degree 3 and G is bipartite.

From this, we construct a UACQ Q as follows: we use the $3n+1$ variables $x_1, \ldots, x_{3n}, y$ in our query. For each $i \in \{1, \ldots, n\}$, we write $e_{j_{i,1}}, e_{j_{i,2}}, e_{j_{i,3}}$ to denote the three edges incident to the vertex v_i. Analogously, we write $e_{j'_{i,1}}, e_{j'_{i,2}}, e_{j'_{i,3}}$ to denote the three edges incident to the vertex v'_i.

The UACQ Q is then defined as $Q(x_1, \ldots, x_{3n}, y) := \varphi \vee \psi$ with

$$\varphi = \bigwedge_{i=1}^{n} R_{C(v_i)}(x_{j_{i,1}}, x_{j_{i,2}}, x_{j_{i,3}}) \wedge S(y),$$

$$\psi = \bigwedge_{i=1}^{n} R_{C(v'_i)}(x_{j'_{i,1}}, x_{j'_{i,2}}, x_{j'_{i,3}}) \wedge S'(y).$$

Moreover, we set the target diversity to $d = f(3n+1)$, where f is the aggregator of δ, and we are looking for $k = 2$ solutions to reach this diversity. Observe that each variable appears exactly once in φ and once in ψ, which makes both formulas trivially acyclic. Furthermore, Q contains no existential variables.

The intuition of the big conjunction in φ (resp. ψ) is to "encode" for each vertex v_i (resp. v'_i) the 3 edges incident to this vertex in the form of the 3 x-variables with the corresponding indices. The relation symbol chosen for each vertex v_i or v'_i depends on the color list for this vertex. For instance, if $C(v_1) = \{2, 3\}$ and if v_1 is incident to the edges e_4, e_6, e_7, then the first conjunct in the definition of φ is of the form $R_{\{2,3\}}(x_4, x_6, x_7)$. Note that the order of the variables in this atom is irrelevant since the R-relations contain only tuples with identical values in all 3 positions. Intuitively, this ensures that a vertex (in this case v_1) gets the same color (in this case color 2 or 3) in all its incident edges (in this case e_4, e_6, e_7).

It remains to prove the correctness of this reduction. For this, observe that diversity $d = f(3n+1)$ can only be achieved by two answers γ, γ' that differ on all variables due to ws-monotonicity. Due to the y variable with possible values 0 and 1, one answer has to satisfy φ while the other answer satisfies ψ. W.l.o.g., let γ satisfy φ and let γ' satisfy ψ. The intuition behind the reduction is that γ tells us how to color the vertices in V while γ' tells us how to color the vertices in V'.

We have to show that (G, l, C) is a positive instance of List Coloring if and only if $\langle Q, I, 2, f(3n+1)\rangle$ is a positive instance of $\mathsf{Diverse}_{\mathsf{wsm}}$-UACQ.

For the "only if"-direction, suppose that (G, l, C) is a positive instance of List Coloring, i.e., graph G has a C-admissible coloring $c : V \cup V' \to \{1, 2, 3\}$. From this, we construct the assignments γ and γ' to the $3n + 1$ variables in Q as follows:

$\gamma(y) = 0$ and $\gamma(x_{j_{i,1}}) = \gamma(x_{j_{i,2}}) = \gamma(x_{j_{i,3}}) = c(v_i)$ for every $i \in \{1, \dots, n\}$ and, analogously, $\gamma'(y) = 1$ and $\gamma'(x_{j'_{i,1}}) = \gamma'(x_{j'_{i,2}}) = \gamma(x_{j'_{i,3}}) = c(v'_i)$ for every $i \in \{1, \dots, n\}$.

We first have to verify that γ is a solution of φ and γ' is a solution of ψ. We only do this for γ. The argumentation for γ' is analogous. $S(\gamma(y)) = S(0)$ is clearly contained in database I. Now consider an arbitrary index $i \in \{1, \dots, n\}$. The atom $R_{C(v_i)}(x_{j_{i,1}}, x_{j_{i,2}}, x_{j_{i,3}})$ is sent to $R_{C(v_i)}(c(v_i), c(v_i), c(v_i))$ by γ. By the above construction of database I, the tuple $(c(v_i), c(v_i), c(v_i))$ is indeed contained in relation $R^I_{C(v_i)}$.

It remains to show that the two assignments γ and γ' differ on every variable. Let $x_{j_{r,t}}$ and $x_{j'_{s,u}}$ with $r, s \in \{1, \dots, n\}$ and $t, u \in \{1, 2, 3\}$ denote the same variable. By our construction of the R-atoms in φ and ψ, this means that $e_{j_{r,t}}$ and $e_{j'_{s,u}}$ denote the same edge in G and v_r and v'_s are the two endpoints of this edge. Since c is a C-admissible coloring, we have $c(v_r) \neq c(v'_s)$. Moreover, by our definition of γ and γ', we have $\gamma(x_{j_{r,t}}) = c(v_r)$ and $\gamma'(x_{j'_{s,u}}) = c(v'_s)$. Hence, γ and γ' indeed differ on an arbitrarily chosen variable and, thus, on every variable.

For the "if"-direction, suppose that $\langle Q, I, 2, f(3n + 1)\rangle$ is a positive instance of Diverse-UACQ, i.e., there exist two solutions γ and γ' with diversity $f(3n + 1)$. This means that γ and γ' differ on every variable, in particular on y. Hence, one of the solutions is an answer of φ and one of ψ. W.l.o.g., let γ be an answer of φ and let γ' be an answer of ψ. From this, we construct the following coloring $c : V \cup V' \to \{1, 2, 3\}$:

$c(v_i) = \gamma(x_{j_{i,1}})$ and $c(v'_i) = \gamma'(x_{j'_{i,1}})$ for every $i \in \{1, \dots, n\}$.

We have to show that c is C-admissible. Consider an arbitrary edge e with endpoints v_r and v_s for $r, s \in \{1, \dots, n\}$. By our construction of Q, there exist indices $t, u \in \{1, 2, 3\}$, such that $x_{j_{r,t}}$ and $x_{j'_{s,u}}$ denote the same variable. Since γ and γ' have diversity $f(3n + 1)$, the assignments γ and γ' differ on every variable. In particular, we have $\gamma(x_{j_{r,t}}) \neq \gamma'(x_{j'_{s,u}})$. Moreover, by our definition of coloring c and the database I, we have $c(v_r) = \gamma(x_{j_{r,1}}) = \gamma(x_{j_{r,t}})$ and $c(v'_s) = \gamma'(x_{j'_{s,1}}) = \gamma'(x_{j'_{s,u}})$. Hence, c assigns different colors to the two arbitrarily chosen, adjacent vertices v_r and v'_s and, therefore, to any adjacent vertices of G. That is, c is C-admissible. □

5. Diversity of Conjunctive Queries with Negation

Lastly, we consider CQs$^\neg$. As was recalled in Section 1, the restriction to acyclicity is not sufficient to ensure tractable answering of CQs$^\neg$ [Lan23]. In the following, we thus restrict ourselves to queries of bounded treewidth when analyzing the Diverse-CQ$^\neg$ problem.

The data complexity case has already been settled for arbitrary FO-formulas in Theorem 3.12. Hence, of course, also Diverse-CQ$^\neg$ is in FPT data complexity and NP-hard in the non-parameterized case. Moreover, we observe that the query used in the proof of Theorem 3.8 has a treewidth of one. Hence, it is clear that also Diverse$_{\text{wsm}}$-CQ$^\neg$ is W[1]-hard combined complexity for queries with bounded treewidth. It remains to study the combined complexity upper bound, for which we describe an XP-algorithm next.

Our algorithm is based on so-called *nice* tree decompositions – a normal form introduced in [Klo94]. A nice tree decomposition only allows leaf nodes plus three types of inner nodes: introduce nodes, forget nodes, and join nodes. An *introduce node* t has a single child t' with $\chi(t) = \chi(t') \cup \{z\}$ for a single variable z. Similarly, a *forget node* t has a single child t' with $\chi(t') = \chi(t) \cup \{z\}$ for a single variable z. Finally, a *join node* t has two child nodes t_1, t_2 with $\chi(t) = \chi(t_1) = \chi(t_2)$. It was shown in [Klo94] that every tree decomposition can be transformed in linear time into a nice tree decomposition without increasing the width.

The intuition of the present algorithm is very similar to the intuition of Algorithm 3.1 presented in Section 3.1.1. That is, both algorithms maintain information on tuples of k partial solutions in a set D_t. Concretely, these tuples are again of the form $(\alpha_1, \ldots, \alpha_k, (d_{i,j})_{1 \leq i < j \leq k})$. This time, however, partial solutions α_i are not assignments that satisfy concrete atoms but arbitrary assignments defined on $\chi(t)$. Nevertheless, a tuple gets added to D_t if and only if it is possible to extend the partial solutions to mappings $\gamma_1, \ldots, \gamma_k$ that (a) satisfy the query associated to the subtree rooted in t and (b) for $1 \leq i < j \leq k$ the distance between γ_i and γ_j is exactly $d_{i,j}$.

Formally, for a CQ$^\neg$ $Q(X) := \exists Y \bigwedge_{i=1}^{n} L_i(X, Y)$ and nice tree decomposition $\langle T, \chi, r \rangle$ of Q we define for $t \in V(T)$ the subquery

$$Q_t = \bigwedge_{\substack{i=1,\ldots,n \\ var(L_i) \subseteq \chi(t)}} L_i,$$

i.e., Q_t contains those literals of Q whose variables are covered by $\chi(t)$.

Algorithm 5.1. Given $Q(X)$, I, k, d, a nice tree decomposition $\langle T, \chi, r \rangle$ of minimum width, and a diversity measure δ defined via some aggregate function f, the algorithm proceeds in two main steps: First, sets D_t are computed bottom-up for each $t \in V(T)$, and then, it is determined from D_r whether the diversity threshold d can be met. For the bottom-up step, the type of t determines how D_t is computed:

- **Leaf Node:** For a leaf node $t \in V(T)$ we create D_t as

$$\begin{aligned} D_t = \{(\alpha_1, \ldots, \alpha_k, (d_{i,j})_{1 \leq i < j \leq k}) : {} & \alpha_1, \ldots, \alpha_k \colon \chi(t) \to dom(I), \\ & \alpha_1, \ldots, \alpha_k \text{ satisfy } Q_t, \\ & d_{i,j} = \Delta_X(\alpha_i, \alpha_j), 1 \leq i < j \leq k\}. \end{aligned}$$

  Hence, we exhaustively go through all possible variable assignments $\alpha_1, \ldots, \alpha_k \colon \chi(t) \to dom(I)$, keep those which satisfy the query Q_t, and record their pairwise diversities.
- **Introduce Node:** For an introduce node $t \in V(T)$ with child $c \in V(T)$ which introduces the variable $z \in \chi(t) \setminus \chi(c)$, we create D_t as

$$\begin{aligned} D_t = \{(\alpha_1 \cup \beta_1, \ldots, \alpha_k \cup \beta_k, (d'_{i,j})_{1 \leq i < j \leq k}) : {} & (\alpha_1, \ldots, \alpha_k, (d_{i,j})_{1 \leq i < j \leq k}) \in D_c, \\ & \beta_1, \ldots, \beta_k \colon \{z\} \to dom(I), \\ & \alpha_1 \cup \beta_1, \ldots, \alpha_k \cup \beta_k \text{ satisfy } Q_t, \\ & d'_{i,j} = d_{i,j} + \Delta_X(\beta_i, \beta_j), 1 \leq i < j \leq k\}. \end{aligned}$$

  Thus, we extend the domain of the local variable assignments in D_c by z. We do this by exhaustively going through all $e \in D_c$ in combination with all $\beta_1, \ldots, \beta_k \colon \{z\} \to dom(I)$, check if the extensions $\alpha_1 \cup \beta_1, \ldots, \alpha_k \cup \beta_k$ satisfy all literals for which all variables are covered, and, if this is the case, add the diversity achieved on the z-variable.

- **Forget Node:** For a forget node $t \in V(T)$ with child $c \in V(T)$ we create D_t as
$$D_t = \{(\alpha_1|_{\chi(t)}, \ldots, \alpha_k|_{\chi(t)}, (d_{i,j})_{1\leq i<j\leq k}) : (\alpha_1, \ldots, \alpha_k, (d_{i,j})_{1\leq i<j\leq k}) \in D_c\}.$$
- **Join Node:** For a join node $t \in V(T)$ with children $c_1, c_2 \in V(T)$ we create D_t as
$$\begin{aligned} D_t = \{(\alpha_1, \ldots, \alpha_k, (d_{i,j})_{1\leq i<j\leq k}) : {} & (\alpha_1, \ldots, \alpha_k, (d'_{i,j})_{1\leq i<j\leq k}) \in D_{c_1}, \\ & (\alpha_1, \ldots, \alpha_k, (d''_{i,j})_{1\leq i<j\leq k}) \in D_{c_2}, \\ & d_{i,j} = d'_{i,j} + d''_{i,j} - \Delta_X(\alpha_i, \alpha_j), 1 \leq i < j \leq k\}. \end{aligned}$$
In this step, we match rows of D_{c_1} with rows of D_{c_2} that agree on the local variable assignments and simply combine the diversities achieved in the two child nodes while subtracting the diversity counted twice.

For the second step, the algorithm goes through all $(\alpha_1, \ldots, \alpha_k, (d_{i,j})_{1\leq i<j\leq k}) \in D_r$ and removes those tuples where $d_{i,j} = 0$ for at least one $1 \leq i < j \leq k$ or $f((d_{i,j})_{1\leq i<j\leq k}) < d$. Then, the algorithm returns "yes" if the resulting set is non-empty and otherwise "no".

Clearly, the algorithm is well-defined and terminates. The next theorem states that the algorithm decides Diverse-CQ$^\neg$, and discusses its running time.

Theorem 5.2. *For a class of CQs$^\neg$ of bounded treewidth, the problem Diverse-CQ$^\neg$ is in* XP *in combined complexity when parameterized by the size k of the diversity set. More specifically, let $Q(X)$ be from a class of CQs$^\neg$ which have treewidth $\leq \omega$. Then, for a database instance I and integers k, d, Algorithm 3.1 solves Diverse-CQ$^\neg$ in time $\mathcal{O}(dom(I)^{2\cdot k\cdot(\omega+1)} \cdot (|X|+1)^{k(k-1)} \cdot poly(|Q|, k))$, where $poly(|Q|, k)$ is a polynomial in $|Q|$ and k.*

To prove this statement, we show by a sequence of lemmas that D_t truly captures the intended meaning. As before, let $Q(X)$ be a CQ$^\neg$, $\langle T, \chi, r\rangle$ a nice tree decomposition of Q, k the number of elements in the diversity set, and d the required diversity. Furthermore, we extend the definition of χ and Q_t to subtrees T_t of T rooted in t. To that end, let $\chi(T_t) = \bigcup_{t' \in V(T_t)} \chi(t')$ and
$$Q_{T_t} = \bigwedge_{\substack{i=1,\ldots,n \\ var(L_i)\subseteq\chi(T_t)}} L_i.$$
With this, for a tuple
$$e = (\alpha_1, \ldots, \alpha_k, (d_{i,j})_{1\leq i<j\leq k}) \in (\chi(t) \to dom(I))^k \times \{0, \ldots, |X|\}^{\frac{k(k-1)}{2}}$$
we define a set of witnesses
$$\begin{aligned} wit_t(e) = \{(\gamma_1, \ldots, \gamma_k) : {} & \gamma_1, \ldots, \gamma_k \in Q_{T_t}(I), \\ & \gamma_1 \cong \alpha_1, \ldots, \gamma_k \cong \alpha_k, \\ & d_{i,j} = \Delta_X(\gamma_i, \gamma_j), 1 \leq i < j \leq k\}. \end{aligned}$$
The existence of such extensions $(\gamma_1, \ldots, \gamma_k) \in wit_t(e)$ are precisely guaranteed by $e \in D_t$ as they satisfy the query corresponding to the subtree rooted in t and $(d_{i,j})_{1\leq i<j\leq k}$ are their pairwise distances. Thus, the algorithm should maintain the following invariant:
$$e \in D_t \text{ if and only if } wit_t(e) \neq \emptyset. \qquad (\dagger)$$
We show next that the algorithm preserves the invariant when handling each node $t \in V(T)$.

Lemma 5.3. *Let t be a leaf node of T. Then Invariant (†) holds for*

$$\begin{aligned} D_t = \{(\alpha_1, \dots, \alpha_k, (d_{i,j})_{1 \le i < j \le k}) : \alpha_1, \dots, \alpha_k \colon \chi(t) \to dom(I), \\ \alpha_1, \dots, \alpha_k \in Q_t(I), \\ d_{i,j} = \Delta_X(\alpha_i, \alpha_j), 1 \le i < j \le k\}. \end{aligned}$$

Proof. Observe that $\chi(t) = \chi(T_t)$ and $\alpha_1, \dots, \alpha_k$ are the only extensions of $\alpha_1, \dots, \alpha_k$. Hence,

$$e = (\alpha_1, \dots, \alpha_k, (d_{i,j})_{1 \le i < j \le k}) \in D_t \iff (\alpha_1, \dots, \alpha_k) \in wit_t(e). \qquad \square$$

Lemma 5.4. *Let t be an introduce node of T which introduces the variable z and let c be its child. Then, if Invariant (†) holds for D_c, it also holds for*

$$\begin{aligned} D_t = \{(\alpha_1 \cup \beta_1, \dots, \alpha_k \cup \beta_k, (d'_{i,j})_{1 \le i < j \le k}) : (\alpha_1, \dots, \alpha_k, (d_{i,j})_{1 \le i < j \le k}) \in D_c, \\ \beta_1, \dots, \beta_k \colon \{z\} \to dom(I), \\ \alpha_1 \cup \beta_1, \dots, \alpha_k \cup \beta_k \in Q_t(I), \\ d'_{i,j} = d_{i,j} + \Delta_X(\beta_i, \beta_j), 1 \le i < j \le k\}. \end{aligned}$$

Proof. First notice that $\chi(T_c) \cup \{z\} = \chi(T_t)$ and thus, every literal of Q_{T_c} appears in Q_{T_t}. Furthermore, any variable of $\chi(T_c)$ that appears together with z in a literal has to appear in $\chi(t)$ due to the properties of a tree decomposition. We can therefore also conclude that a literal appears in Q_{T_t} if and only if it appears in Q_{T_c} or Q_t.

Now, let the tuple $e = (\alpha_1 \cup \beta_1, \dots, \alpha_k \cup \beta_k, (d'_{i,j})_{1 \le i < j \le k})$ be in D_t and let $e_c = (\alpha_1, \dots, \alpha_k, (d_{i,j})_{1 \le i < j \le k})$ be a matching tuple in D_c. Thus there is a $(\gamma_1, \dots, \gamma_k) \in wit_c(e_c)$. Importantly, $\gamma_1, \dots, \gamma_k \in I(Q_{T_c}), \alpha_1 \cup \beta_1, \dots, \alpha_k \cup \beta_k \in I(Q_t)$ and thus, $\gamma_1 \cup \beta_1, \dots, \gamma_k \cup \beta_k \in I(Q_{T_t})$. Furthermore, for $1 \le i < j \le k$, we have:

$$\begin{aligned} d'_{i,j} &= d_{i,j} + \Delta_X(\beta_i, \beta_j) \\ &= \Delta_X(\gamma_i, \gamma_j) + \Delta_X(\beta_i, \beta_j) \\ &= \Delta_X(\gamma_i \cup \beta_i, \gamma_j \cup \beta_j). \end{aligned}$$

Thus, $(\gamma_1 \cup \beta_1, \dots, \gamma_k \cup \beta_k) \in wit_t(e)$ by definition.

For the reverse direction, let $e = (\alpha_1, \dots, \alpha_k, (d'_{i,j})_{1 \le i < j \le k})$ be such that there is a $(\gamma_1, \dots, \gamma_k) \in wit_t(e)$. Thus, we can immediately conclude that $\gamma_1|_{\chi(t)}, \dots, \gamma_k|_{\chi(t)} \in I(Q_t)$ while $\gamma_1|_{\chi(T_c)}, \dots, \gamma_k|_{\chi(T_c)} \in I(Q_{T_c})$. Furthermore, for $1 \le i < j \le k$, we have:

$$\begin{aligned} d'_{i,j} - \Delta_X(\gamma_i|_{\{z\}}, \gamma_j|_{\{z\}}) &= \Delta_X(\gamma_i, \gamma_j) - \Delta_X(\gamma_i|_{\{z\}}, \gamma_j|_{\{z\}}) \\ &= \Delta_X(\gamma_i|_{\chi(T_t)}, \gamma_j|_{\chi(T_t)}). \end{aligned}$$

Thus,

$$e_c = (\gamma_1|_{\chi(c)}, \dots, \gamma_k|_{\chi(c)}, (d'_{i,j} - \Delta_X(\gamma_i|_{\{z\}}, \gamma_j|_{\{z\}}))_{1 \le i < j \le k}) \in D_c.$$

as $(\gamma_1|_{\chi(T_c)}, \dots, \gamma_k|_{\chi(T_c)}) \in wit_c(e_c)$. Defining $\beta_1 = \gamma_1|_{\{z\}}, \dots, \beta_k = \gamma_k|_{\{z\}}$ then ensures that $(\gamma_1|_{\chi(T_c)} \cup \beta_1, \dots, \gamma_k|_{\chi(T_c)} \cup \beta_k, (d'_{i,j})_{1 \le i < j \le k}) = (\alpha_1, \dots, \alpha_k, (d'_{i,j})_{1 \le i < j \le k})$ is in D_t. $\square$

Lemma 5.5. *Let t be a forget node of T and c its child. Then, if Invariant (†) holds for D_c, it also holds for*

$$D_t = \{(\alpha_1|_{\chi(t)}, \dots, \alpha_k|_{\chi(t)}, (d_{i,j})_{1 \le i < j \le k}) : (\alpha_1, \dots, \alpha_k, (d_{i,j})_{1 \le i < j \le k}) \in D_c\}.$$

Proof. Let z be the forgotten variable. The claim follows from the fact that $Q_{T_t} = Q_{T_c}$ and $\chi(t) \subseteq \chi(c)$, and thus,

$$wit_t(\alpha'_1, \dots, \alpha'_k, (d_{i,j})_{1\le i<j\le k}) = \bigcup_{\substack{\beta_i\colon \{z\}\to dom(I)\\ l=1,\dots,k}} wit_c(\alpha'_1 \cup \beta_1, \dots, \alpha'_k \cup \beta_k, (d_{i,j})_{1\le i<j\le k}). \quad \square$$

Lemma 5.6. *Let t be a join node of T with children c_1 and c_2. Then, if Invariant (†) holds for D_{c_1} and D_{c_2}, it also holds for*

$$\begin{aligned} D_t = \{(\alpha_1, \dots, \alpha_k, (d_{i,j})_{1\le i<j\le k}) : {} & (\alpha_1, \dots, \alpha_k, (d'_{i,j})_{1\le i<j\le k}) \in D_{c_1}, \\ & (\alpha_1, \dots, \alpha_k, (d''_{i,j})_{1\le i<j\le k}) \in D_{c_2}, \\ & d_{i,j} = d'_{i,j} + d''_{i,j} - \Delta_X(\alpha_i, \alpha_j), 1 \le i < j \le k\}. \end{aligned}$$

Proof. First notice that $\chi(T_{c_1}) \cup \chi(T_{c_2}) = \chi(T_t)$ and thus, literals that appear in $Q_{T_{c_1}}$ or $Q_{T_{c_2}}$ also appear in Q_{T_t}. Moreover, if two variables appear in the same literal in Q_{T_t} but they no longer jointly occur in $\chi(t)$, then these variables have to appear together in either T_{c_1} or T_{c_2}. We can, therefore, observe that a literal appears in Q_{T_t} if and only if it appears in $Q_{T_{c_1}}$ or $Q_{T_{c_2}}$.

We start with some

$$e_1 = (\alpha_1, \dots, \alpha_k, (d'_{i,j})_{1\le i<j\le k}) \in D_{c_1} \text{ and } e_2 = (\alpha_1, \dots, \alpha_k, (d''_{i,j})_{1\le i<j\le k}) \in D_{c_2}.$$

Now let $(\gamma'_1, \dots, \gamma'_k) \in wit_{c_1}(e_1)$ and $(\gamma''_1, \dots, \gamma''_k) \in wit_{c_2}(e_2)$ witness this, respectively. By the above observation, $\gamma'_1 \cup \gamma''_1, \dots, \gamma'_k \cup \gamma''_k \in Q_{T_t}(I)$ and, for $1 \le i < j \le k$, we have:

$$\begin{aligned} \Delta_X(\gamma'_i \cup \gamma''_i, \gamma'_j \cup \gamma''_j) &= \Delta_X(\gamma'_i, \gamma'_j) + \Delta_X(\gamma''_i, \gamma''_j) - \Delta_X(\gamma'_i \cap \gamma''_i, \gamma'_j \cap \gamma''_j) \\ &= d'_{i,j} + d''_{i,j} - \Delta_X(\alpha_i, \alpha_j). \end{aligned}$$

Hence, $e = (\alpha_1, \dots, \alpha_k, (d'_{i,j} + d''_{i,j} - \Delta_X(\alpha_i, \alpha_j))_{1\le i<j\le k}) \in D_t$ is justified as $(\gamma'_1 \cup \gamma''_k, \dots, \gamma'_1 \cup \gamma''_k) \in wit_t(e)$.

Conversely, let $e = (\alpha_1, \dots, \alpha_k, (d_{i,j})_{1\le i<j\le k})$ be a tuple such that there is a $(\gamma_1, \dots, \gamma_k) \in wit_t(e)$. Thus, we can immediately conclude that the restrictions $\gamma_1|_{\chi(T_{c_1})}, \dots, \gamma_k|_{\chi(T_{c_1})}$ are in $I(Q_{T_{c_1}})$ while the restrictions $\gamma_1|_{\chi(T_{c_2})}, \dots, \gamma_k|_{\chi(T_{c_2})}$ are in $I(Q_{T_{c_2}})$. This implies that

$$\begin{aligned} &(\gamma_1|_{\chi(t)}, \dots, \gamma_k|_{\chi(t)}, (\Delta_X(\gamma_i|_{\chi(T_{c_1})}, \gamma_k|_{\chi(T_{c_1})}))_{1\le i<j\le k}) \in D_{c_1}, \\ &(\gamma_1|_{\chi(t)}, \dots, \gamma_k|_{\chi(t)}, (\Delta_X(\gamma_i|_{\chi(T_{c_2})}, \gamma_k|_{\chi(T_{c_2})}))_{1\le i<j\le k}) \in D_{c_2}. \end{aligned}$$

Lastly, we can compute for $1 \le i < j \le k$:

$$\begin{aligned} d_{i,j} &= \Delta_X(\gamma_i, \gamma_j) \\ &= \Delta_X(\gamma_i|_{\chi(T_{c_1})}, \gamma_j|_{\chi(T_{c_1})}) + \Delta_X(\gamma_i|_{\chi(T_{c_2})}, \gamma_j|_{\chi(T_{c_2})}) - \Delta_X(\gamma_i|_{\chi(t)}, \gamma_j|_{\chi(t)}), \end{aligned}$$

implying that $e \in D_t$. $\square$

Importantly, Lemmas 5.3 through 5.6 ensure that after the bottom-up traversal of Algorithm 5.1, Invariant (†) is satisfied for D_r. We now show that the algorithm correctly determines from D_r if there is a diversity set of size k which has diversity exceeding d.

Lemma 5.7. *If Invariant (†) holds for D_r then there exist solutions $\{\gamma_1, \dots, \gamma_k\} \subseteq Q(I)$ with $\delta(\gamma_1, \dots, \gamma_k) \ge d$ if and only if there is a tuple $(\alpha_1, \dots, \alpha_k, (d_{i,j})_{1\le i<j\le k}) \in D_r$ such that $d_{i,j} > 0, 1 \le i < j \le k$ and $f((d_{i,j})_{1\le i<j\le k}) \ge d$*

Proof. First observe that $Q = \exists Y Q_{T_r}$ and thus, $Q(I) = \{\gamma|_X : \gamma \in Q(T_r)\}$. Furthermore, for $\gamma, \gamma' \in Q(T_r)$ we have $\Delta(\gamma|_X, \gamma|_X) = \Delta_X(\gamma, \gamma')$.

Now assume k solutions $\{\gamma_1|_X, \dots, \gamma_k|_X\} \subseteq Q(I)$ with $\delta(\gamma_1|_X, \dots, \gamma_k|_X) \geq d$ to exist. Consider the tuple $e = (\gamma_1|_{\chi(r)}, \dots, \gamma_k|_{\chi(r)}, (\Delta_X(\gamma_i, \gamma_j))_{1 \leq i < j \leq k})$. By definition, we have $(\gamma_1, \dots, \gamma_k) \in wit_t(e)$ and hence, $e \in D_r$. Furthermore, $\Delta_X(\gamma_i, \gamma_j) = \Delta(\gamma_i|_X, \gamma_j|_X) > 0$ for $1 \leq i < j \leq k$ and

$$f((\Delta_X(\gamma_i, \gamma_j))_{1 \leq i < j \leq k}) = f((\Delta(\gamma_i|_X, \gamma_j|_X))_{1 \leq i < j \leq k}) = \delta(\gamma_1|_X, \dots, \gamma_k|_X) \geq d.$$

For the reverse direction assume such a tuple $e = (\alpha_1, \dots, \alpha_k, (d_{i,j})_{1 \leq i < j \leq k}) \in D_r$ with $d_{i,j} > 0, 1 \leq i < j \leq k$, and $f((d_{i,j})_{1 \leq i < j \leq k}) \geq d$ to exist. Thus, there also exists $(\gamma_1, \dots, \gamma_k) \in wit_r(e)$ and we claim that $\{\gamma_1|_X, \dots, \gamma_k|_X\}$ is a diversity set as required. First observe that $\gamma_i|_X$ is different to $\gamma_j|_X$ for $1 \leq i < j \leq k$ as $\Delta(\gamma_i|_X, \gamma_j|_X) = \Delta_X(\gamma_i, \gamma_j) = d_{i,j} > 0$. Secondly, $\delta(\gamma_1|_X, \dots, \gamma_k|_X) = f((\Delta(\gamma_i|_X, \gamma_j|_X))_{1 \leq i < j \leq k}) = f((\Delta_X(\gamma_i, \gamma_j))_{1 \leq i < j \leq k})$. □

We now show that the bottom-up step is also possible in the required time bound.

Lemma 5.8. *Let t be an arbitrary node in T, ω the width of the tree decomposition. Then, D_t can be computed in time $\mathcal{O}(dom(I)^{2 \cdot k \cdot (\omega+1)} \cdot (|X|+1)^{k(k-1)} \cdot (|Q| + k^2 \cdot \omega))$ given the sets of its children.*

Proof. This running time is achieved by a naive implementation. For a leaf node, we simply have to iterate through the $|dom(I)|^{|\chi(t)| \cdot k}$ options for $\alpha_1, \dots, \alpha_k \colon \chi(t) \to dom(I)$, check that all assignments are answers to Q_t (each relevant table of I is at most of size $dom(I)^{\omega+1}$), and compute the Hamming distances.

For an introduce node, we iterate through all elements of D_c and all $|dom(I)|^k$ possibilities for $\beta_1, \dots, \beta_2 \colon \{z\} \to dom(I)$, again check whether all assignments are answers to Q_t, and update the distances. Note that the size of the set D_c is at most $|dom(I)|^{k \cdot (\omega+1)} \cdot (|X|+1)^{\frac{k(k-1)}{2}}$ by definition since $|X|$ is an upper bound on the pairwise Hamming distance.

For a forget node, we simply need to perform a projection. Lastly, for a join node, we iterate through $D_{c_1} \times D_{c_2}$, check whether the assignments $\alpha_1, \dots, \alpha_k$ match, and then update the distances.

All of the cases can therefore clearly be handled in the given time bound. □

With this, we can now show Theorem 5.2.

Proof of Theorem 5.2. To prove the result, we only have to ensure that finding a suitable tree decomposition and applying the algorithm is possible in the required time bound as the correctness of this procedure follows directly from Lemmas 5.3 to 5.7.

Due to [Bod96] and [Klo94] computing a width optimal nice tree decomposition is possible in linear time. Furthermore, the number of nodes in this tree decomposition is linear in $|Q|$ and, thus, performing the bottom-up traversal of the algorithm is possible in the required time bound due to Lemma 5.8. Lastly, we also have to look at the running time of the final step of the algorithm. There, we possibly have to evaluate f for each tuple in D_r. But, since $|D_r| \leq dom(I)^{k(\omega+1)} \cdot (|X|+1)^{\frac{k(k-1)}{2}}$ and f is computable in polynomial time (in k and $|X|$), also this step is possible in the required time bound. □

We conclude this section by again stressing the analogy with Algorithm 3.1 for ACQs: First, we have omitted from our description of Algorithm 5.1 how to compute a concrete witnessing diversity set in the case of a yes-answer. This can be done exactly as in Algorithm 3.1 by maintaining the same kind of provenance information. And second, it is

possible to speed up the present algorithm by applying the same kind of considerations as in Section 3.1.3. It is thus possible to reduce the query complexity to FPT for the diversity measure δ_{sum} and even further to P if we allow duplicates in the diversity set.

6. Conclusion and Future Work

In this work, we have had a fresh look at the Diversity problem of query answering. For CQs and extensions thereof, we have proved a collection of complexity results, both for the parameterized and the non-parameterized case. To get a chance of reaching tractability or at least fixed-parameter tractability (when considering the size k of the diversity set as the parameter), we have restricted ourselves to acyclic CQs and CQs with negation of bounded treewidth, respectively. For the chosen settings, our complexity results are fairly complete. The most obvious gaps left for future work are concerned with the query complexity of ACQs and CQs with negation of bounded treewidth. For the parameterized case, we have XP-membership but no fixed-parameter intractability result in the form of W[1]-hardness. And for the non-parameterized case, it is open if the problems are also NP-hard as we have shown for the data complexity.

It should be noted that the restriction to acyclic CQs is less restrictive than it may seem at first glance. As was mentioned in Section 2, query evaluation of CQs with bounded hypertree width (likewise, CQs with bounded generalized or fractional hypertree width) can be efficiently reduced to query evaluation of acyclic CQs. Hence, our upper bounds (in particular, the XP- and FPT-membership results in Section 3) are easily generalized to CQs of bounded hypertree-width [GLS02]. Moreover, recent empirical studies of millions of queries from query logs [BMT20] and thousands of queries from benchmarks [FGLP21] have shown that CQs typically have hypertree-width at most 3.

A yet more powerful width measure than *hw*, *ghw*, and *fhw* is submodular-width (*smw*) introduced in [Mar13]. In fact, Marx showed that bounded *smw*, in a sense, exactly characterizes the class of CQs for which query evaluation is fixed-parameter tractable, parameterized by the size of the query. In contrast to the other width measures, there is no obvious extension of our results to CQs of bounded *smw*. Indeed, query evaluation based on bounded *smw* works by partitioning the database via so-called "heavy-light splitting" (i.e., treating attribute values with high vs. low frequency differently). Clearly, this reduces the problem of CQ-evaluation to a problem of UCQ-evaluation and we have seen in Section 4 that the diversity problem for UACQs immediately leads to intractability. Hence, a completely different approach would be needed to extend our results to CQs with bounded *smw*.

For CQs with negation, a stronger restriction than acyclicity (or bounded hw, ghw, fhw) is needed to achieve tractability of query evaluation [Lan23]. Consequently, we have studied CQs with negation of bounded treewidth. Another interesting restriction on the structure of CQs with negation to achieve tractable query evaluation is β-acyclicity [Bra12, NNRR14, CI24] or, more generally, bounded nest-set width [Lan23, CI24]. Both these properties are defined via a form of variable elimination. In contrast to acyclic queries or queries of bounded tw (and, likewise, bounded hw, ghw, fhw), there is no form of *decomposition* to characterize β-acyclicity or bounded nest-set width. Hence, there is no obvious way how to extend our results on CQs with negation to β-acyclic queries or queries with bounded nest-set width. We leave both, the diversity study of CQs with bounded *smw* and of CQs with negation of bounded nest-set width as interesting questions for future work.

Another direction for future work is motivated by a closer look at our FPT- and XP-membership results: even though such parameterized complexity results are generally considered as favorable (in particular, FPT), the running times are exponential in the parameter k. As we allow larger values of k, these running times may not be acceptable anymore. It would therefore be interesting to study the diversity problem also from an approximation point of view – in particular, contenting oneself with an approximation of the desired diversity.

A further modification of our settings is related to the choice of a different distance measure between two answer tuples and different aggregators. As far as the distance measure is concerned, we have so far considered data values as untyped and have therefore studied only the Hamming distance between tuples. For numerical values, one might of course take the difference between values into account. More generally, one could consider a metric on the domain, which then induces a metric on tuples that can be used as a distance measure. As far as the aggregator is concerned, we note that most of our upper bounds apply to arbitrary (polynomial-time computable) aggregate functions. On the other hand, our lower bounds hold for any (polynomial-time computable) ws-monotone aggregate functions. This seems quite a natural choice as almost all natural aggregators in this setting – including sum and min – are ws-monotone. A problem strongly related to Diversity is Similarity [EEEF13], where one is interested in finding solutions close to each other. Parts of our approach can naturally be adapted to this setting but we leave the in-depth study of Similarity for future work.

References

[ACJR19] Marcelo Arenas, Luis Alberto Croquevielle, Rajesh Jayaram, and Cristian Riveros. Efficient logspace classes for enumeration, counting, and uniform generation. In Dan Suciu, Sebastian Skritek, and Christoph Koch, editors, *PODS 2019*, pages 59–73. ACM, 2019. doi:10.1145/3294052.3319704.

[AGG07] Isolde Adler, Georg Gottlob, and Martin Grohe. Hypertree width and related hypergraph invariants. *Eur. J. Comb.*, 28(8):2167–2181, 2007. doi:10.1016/J.EJC.2007.04.013.

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[AJMR22] Antoine Amarilli, Louis Jachiet, Martin Muñoz, and Cristian Riveros. Efficient enumeration for annotated grammars. In Leonid Libkin and Pablo Barceló, editors, *PODS 2022*, pages 291–300. ACM, 2022. doi:10.1145/3517804.3526232.

[AK97] Paola Alimonti and Viggo Kann. Hardness of approximating problems on cubic graphs. In Gian Carlo Bongiovanni, Daniel P. Bovet, and Giuseppe Di Battista, editors, *CIAC 1997*, pages 288–298. Springer, 1997. doi:10.1007/3-540-62592-5_80.

[BFJ+22] Julien Baste, Michael R. Fellows, Lars Jaffke, Tomás Masarík, Mateus de Oliveira Oliveira, Geevarghese Philip, and Frances A. Rosamond. Diversity of solutions: An exploration through the lens of fixed-parameter tractability theory. *Artif. Intell.*, 303:103644, 2022. doi:10.1016/j.artint.2021.103644.

[BKPS19] Endre Boros, Benny Kimelfeld, Reinhard Pichler, and Nicole Schweikardt. Enumeration in data management (dagstuhl seminar 19211). *Dagstuhl Reports*, 9(5):89–109, 2019. doi:10.4230/DagRep.9.5.89.

[BMT20] Angela Bonifati, Wim Martens, and Thomas Timm. An analytical study of large SPARQL query logs. *VLDB J.*, 29(2-3):655–679, 2020. doi:10.1007/s00778-019-00558-9.

[Bod96] Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996. doi:10.1137/S0097539793251219.

[Bra12] Johann Brault-Baron. A negative conjunctive query is easy if and only if it is beta-acyclic. In Patrick Cégielski and Arnaud Durand, editors, *CSL 2012*, pages 137–151. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPIcs.CSL.2012.137.

[CC06] Miroslav Chlebík and Janka Chlebíková. Hard coloring problems in low degree planar bipartite graphs. *Discret. Appl. Math.*, 154(14):1960–1965, 2006. doi:10.1016/j.dam.2006.03.014.

[CI24] Florent Capelli and Oliver Irwin. Direct access for conjunctive queries with negations. In Graham Cormode and Michael Shekelyan, editors, *ICDT 2024*, pages 13:1–13:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ICDT.2024.13.

[CM77] Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In John E. Hopcroft, Emily P. Friedman, and Michael A. Harrison, editors, *STOC 1977*, pages 77–90. ACM, 1977. doi:10.1145/800105.803397.

[CM99] Surajit Chaudhuri and Rajeev Motwani. On sampling and relational operators. *IEEE Data Eng. Bull.*, 22(4):41–46, 1999.

[CS23] Nofar Carmeli and Luc Segoufin. Conjunctive queries with self-joins, towards a fine-grained enumeration complexity analysis. In Floris Geerts, Hung Q. Ngo, and Stavros Sintos, editors, *PODS 2023*, pages 277–289. ACM, 2023. doi:10.1145/3584372.3588667.

[CZB+22] Nofar Carmeli, Shai Zeevi, Christoph Berkholz, Alessio Conte, Benny Kimelfeld, and Nicole Schweikardt. Answering (unions of) conjunctive queries using random access and random-order enumeration. *ACM Trans. Database Syst.*, 47(3):9:1–9:49, 2022. doi:10.1145/3531055.

[DF99] Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999. doi:10.1007/978-1-4612-0515-9.

[DF14] Ting Deng and Wenfei Fan. On the complexity of query result diversification. *ACM Trans. Database Syst.*, 39(2):15:1–15:46, 2014. doi:10.1145/2602136.

[DST23] Shiyuan Deng, Francesco Silvestri, and Yufei Tao. Enumerating subgraphs of constant sizes in external memory. In Floris Geerts and Brecht Vandevoort, editors, *ICDT 2023*, pages 4:1–4:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ICDT.2023.4.

[EEEF13] Thomas Eiter, Esra Erdem, Halit Erdogan, and Michael Fink. Finding similar/diverse solutions in answer set programming. *Theory Pract. Log. Program.*, 13(3):303–359, 2013. doi:10.1017/S1471068411000548.

[FGLP21] Wolfgang Fischl, Georg Gottlob, Davide Mario Longo, and Reinhard Pichler. Hyperbench: A benchmark and tool for hypergraphs and empirical findings. *ACM J. Exp. Algorithmics*, 26:1.6:1–1.6:40, 2021. doi:10.1145/3440015.

[FGS18] Henning Fernau, Petr A. Golovach, and Marie-France Sagot. Algorithmic enumeration: Output-sensitive, input-sensitive, parameterized, approximative (dagstuhl seminar 18421). *Dagstuhl Reports*, 8(10):63–86, 2018. doi:10.4230/DagRep.8.10.63.

[GLS02] Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions and tractable queries. *J. Comput. Syst. Sci.*, 64(3):579–627, 2002. doi:10.1006/jcss.2001.1809.

[GM14] Martin Grohe and Dániel Marx. Constraint solving via fractional edge covers. *ACM Trans. Algorithms*, 11(1):4:1–4:20, 2014. doi:10.1145/2636918.

[Gra79] Marc H. Graham. On The Universal Relation. Technical report, University of Toronto, 1979.

[HHOW05] Emmanuel Hebrard, Brahim Hnich, Barry O'Sullivan, and Toby Walsh. Finding diverse and similar solutions in constraint programming. In Manuela M. Veloso and Subbarao Kambhampati, editors, *AAAI 2005*, pages 372–377. AAAI Press / The MIT Press, 2005.

[IdlBST20] Linnea Ingmar, Maria Garcia de la Banda, Peter J. Stuckey, and Guido Tack. Modelling diversity of solutions. In *AAAI 2020*, pages 1528–1535. AAAI Press, 2020.

[JPY88] David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. On generating all maximal independent sets. *Inf. Process. Lett.*, 27(3):119–123, 1988. doi:10.1016/0020-0190(88)90065-8.

[KKW22] Yasuaki Kobayashi, Kazuhiro Kurita, and Kunihiro Wasa. Linear-delay enumeration for minimal steiner problems. In Leonid Libkin and Pablo Barceló, editors, *PODS 2022*, pages 301–313. ACM, 2022. doi:10.1145/3517804.3524148.

[Klo94] Ton Kloks. *Treewidth, Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer, 1994. doi:10.1007/BFb0045375.

[Lan23] Matthias Lanzinger. Tractability beyond β-acyclicity for conjunctive queries with negation and SAT. *Theor. Comput. Sci.*, 942:276–296, 2023. doi:10.1016/J.TCS.2022.12.002.

[LP22] Carsten Lutz and Marcin Przybylko. Efficiently enumerating answers to ontology-mediated queries. In Leonid Libkin and Pablo Barceló, editors, *PODS 2022*, pages 277–289. ACM, 2022. doi:10.1145/3517804.3524166.

[LRG+17] Viktor Leis, Bernhard Radke, Andrey Gubichev, Alfons Kemper, and Thomas Neumann. Cardinality estimation done right: Index-based join sampling. In *CIDR*, 2017.

[LWYZ19] Feifei Li, Bin Wu, Ke Yi, and Zhuoyue Zhao. Wander join and XDB: online aggregation via random walks. *ACM Trans. Database Syst.*, 44(1):2:1–2:41, 2019. doi:10.1145/3284551.

[Mar13] Dániel Marx. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. *J. ACM*, 60(6):42:1–42:51, 2013. doi:10.1145/2535926.

[MPS23] Timo Camillo Merkl, Reinhard Pichler, and Sebastian Skritek. Diversity of answers to conjunctive queries. In Floris Geerts and Brecht Vandevoort, editors, *ICDT 2023*, pages 10:1–10:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[MR23] Martin Muñoz and Cristian Riveros. Constant-delay enumeration for slp-compressed documents. In Floris Geerts and Brecht Vandevoort, editors, *ICDT 2023*, pages 7:1–7:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ICDT.2023.7.

[Nad11] Alexander Nadel. Generating diverse solutions in SAT. In Karem A. Sakallah and Laurent Simon, editors, *SAT 2011*, pages 287–301. Springer, 2011. doi:10.1007/978-3-642-21581-0_23.

[NNRR14] Hung Q. Ngo, Dung T. Nguyen, Christopher Ré, and Atri Rudra. Beyond worst-case analysis for joins with minesweeper. In Richard Hull and Martin Grohe, editors, *PODS 2014*, pages 234–245. ACM, 2014. doi:10.1145/2594538.2594547.

[Pap94] Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

[PT15] Thierry Petit and Andrew C. Trapp. Finding diverse solutions of high quality to constraint optimization problems. In Qiang Yang and Michael J. Wooldridge, editors, *IJCAI 2015*, pages 260–267. AAAI Press, 2015.

[RS84] Neil Robertson and Paul D. Seymour. Graph minors. III. planar tree-width. *J. Comb. Theory, Ser. B*, 36(1):49–64, 1984. doi:10.1016/0095-8956(84)90013-3.

[Var82] Moshe Y. Vardi. The complexity of relational query languages (extended abstract). In Harry R. Lewis, Barbara B. Simons, Walter A. Burkhard, and Lawrence H. Landweber, editors, *STOC 1982*, pages 137–146. ACM, 1982. doi:10.1145/800070.802186.

[Yan81] Mihalis Yannakakis. Algorithms for acyclic database schemes. In *VLDB 1981*, pages 82–94. IEEE Computer Society, 1981.

[YO79] C. T. Yu and M. Z. Özsoyoglu. An algorithm for tree-query membership of a distributed query. In *COMPSAC 1979*, pages 306–312, 1979.

[ZCL+18] Zhuoyue Zhao, Robert Christensen, Feifei Li, Xiao Hu, and Ke Yi. Random sampling over joins revisited. In *SIGMOD Conference 2018*, pages 1525–1539. ACM, 2018. doi:10.1145/3183713.3183739.

[ZWQ+17] Kaiping Zheng, Hongzhi Wang, Zhixin Qi, Jianzhong Li, and Hong Gao. A survey of query result diversification. *Knowl. Inf. Syst.*, 51(1):1–36, 2017. doi:10.1007/s10115-016-0990-4.