Diversity of Answers to Conjunctive Queries

Timo Camillo Merkl
TU Wien, Austria

Reinhard Pichler
TU Wien, Austria

Sebastian Skritek
TU Wien, Austria

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.

2012 ACM Subject Classification Information systems → Data management systems

Keywords and phrases Query Answering, Diversity of Solutions, Complexity, Algorithms

Digital Object Identifier 10.4230/LIPIcs.ICDT.2023.10

Related Version *Full Version*: `https://arxiv.org/abs/2301.08848` [28]

Funding This work was supported by the Austrian Science Fund (FWF) project P30930-N35.

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 [8, 16]. Also in the Database Theory community, enumeration problems have played a prominent role on the research agenda recently, see e.g., [3, 24, 27]. 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 [22] 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) 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 [4, 9]. In fact, research on sampling has a long tradition in the Database community [11] – above all with the goal of supporting more accurate cardinality estimations [25, 26, 36].

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 [5, 15, 29] – especially when dealing with Constraint Satisfaction Problems (CSPs) [20, 21, 30]. For instance, consider a variation of the car dealership example from [20]. 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 [20], the Database community – apart from limited exceptions [13] – focused on heuristic and approximation methods to find diverse solutions (see [37] for an extensive survey). Also, in contrast to the present work, 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 [21], 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. As far as the choice of an aggregator f is concerned, we impose the general restriction that it must be computable in polynomial time. As will be detailed below, we will sometimes also consider more restricted cases of f. 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-$\mathcal{Q}$:

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 $\gamma_1, \ldots, \gamma_k \in Q(I)$ such that $\delta(\gamma_1, \ldots, \gamma_k) \geq d$?

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*. 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 $\delta(\gamma_1, \ldots, \gamma_k) := f((d_{i,j})_{1\leq i<j\leq k})$. Moreover, we write δ_{sum}, δ_{min}, and δ_{mon} if the aggregator f is the minimum, the sum, or an arbitrary (polynomial-time computable) 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 $N = \frac{k(k-1)}{2}$. The corresponding diversity problems are denoted by Diverse$_{\mathsf{sum}}$-$\mathcal{Q}$, Diverse$_{\mathsf{min}}$-$\mathcal{Q}$, and Diverse$_{\mathsf{mon}}$-$\mathcal{Q}$, respectively.

When we prove upper bounds on the complexity of several variations of the Diverse-$\mathcal{Q}$ problem (in the form of membership in some favorable complexity class), we aim at the most general setting, i.e., the Diverse-$\mathcal{Q}$ problem for an arbitrary, polynomial-time computable aggregation function. However, in some cases, the restriction to sum, min, or mon 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 aim at restricted cases (in particular,

$\mathsf{Diverse}_{\mathsf{sum}}$-$\mathcal{Q}$ and $\mathsf{Diverse}_{\mathsf{min}}$-$\mathcal{Q}$). These hardness results, of course, carry over to the more general cases. Somehow surprisingly, our analyses will reveal differences in the complexity between the seemingly similar cases $\mathsf{Diverse}_{\mathsf{sum}}$-$\mathcal{Q}$ and $\mathsf{Diverse}_{\mathsf{min}}$-$\mathcal{Q}$.

We will analyze several variations of the Diverse-$\mathcal{Q}$ problem: as mentioned above, we study various aggregator functions. Moreover, we consider 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 [10]. 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 [32]. 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 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 Diverse-$\mathcal{Q}$ problem from various angles. We thus mainly 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 $\mathsf{Diverse}_{\mathsf{sum}}$-ACQ and $\mathsf{Diverse}_{\mathsf{min}}$-ACQ problems. 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 $\mathsf{Diverse}_{\mathsf{sum}}$-UACQ and $\mathsf{Diverse}_{\mathsf{min}}$-UACQ problems are NP-complete. 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. Due to the space limit, most proofs are only sketched. Full details are provided in [28].

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., [1]. 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 $X \subseteq Z$ and $z \in Z$, we write $\alpha|_X$ and $\alpha|_z$ for the mapping resulting from the restriction of α to the set X 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 [34], 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 $T[T_v]$.

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 [19, 35].

Another approach to making CQ answering tractable is by restricting the *treewidth* (*tw*), which is defined via *tree decompositions* [31]. 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 $T[T_v]$.

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 $\leq \omega$ [6].

Complexity. We follow the categorization of the complexity of database tasks introduced in [33] 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 [14]. 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 run 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 FPT $\subseteq$ W[1] $\subseteq$ XP. It is a generally accepted assumption in parameterized complexity theory that FPT $\neq$ W[1] holds – similar but slightly stronger than the famous P $\neq$ NP assumption in classical complexity theory, i.e., FPT $\neq$ W[1] implies P $\neq$ 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 $\{\alpha\colon Z \to dom(I) : \alpha(Z_i) \in R_i^I$ 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 $e \in \{(\alpha_1, \ldots, \alpha_k, (d_{i,j})_{1\leq i<j\leq k}) : \alpha_1, \ldots, \alpha_k \in t(I), d_{i,j} \in \{0, \ldots, |X|\}$ for $1 \leq i < j \leq k\}$, we define $ext_{T'}(e) = \{(\gamma_1, \ldots, \gamma_k) : \gamma_1, \ldots, \gamma_k \in T'(I)$ s.t. $\alpha_i \cong \gamma_i$ for $1 \leq i \leq k$ and $\Delta_X(\gamma_i, \gamma_j) = d_{i,j}$ for $1 \leq i < j \leq k\}$.

Intuitively, our algorithm checks for each such tuple e whether there exist extensions γ_i of α_i that (a) are solutions to the subquery induced by T' and (b) 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 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

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

  That is, D_t contains one entry for every combination $\alpha_1, \ldots, \alpha_k \in t(I)$, 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.
- **Bottom-Up Traversal:** Set the status of all non-leaf nodes in T to "not-ready" and the status of all leaf nodes to "ready". 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 as

$$D'_t = \{(\alpha_1, \ldots, \alpha_k, (\bar{d}_{i,j})_{1 \le i < j \le k}) : (\alpha_1, \ldots, \alpha_k, (d_{i,j})_{1 \le i < j \le k}) \in D_t,$$
$$(\alpha'_1, \ldots, \alpha'_k, (d'_{i,j})_{1 \le i < j \le k}) \in D_{t'},$$
$$\alpha_i \cong \alpha'_i \text{ for } 1 \le i \le k,$$
$$\bar{d}_{i,j} = d_{i,j} + d'_{i,j} - \Delta_X(\alpha_i \cap \alpha'_i, \alpha_j \cap \alpha'_j)$$
$$\text{for } 1 \le i < j \le k\}.$$

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 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 $\rho_{D'_t}(\bar{e}) = \rho_{D_t}(e) \cup \{(t', e')\}$. If the same entry $\bar{e}$ is created from different pairs (e, e'), choose an arbitrary one of them for the definition of $\rho_{D'_t}(\bar{e})$.
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 $(\alpha_1, \ldots, \alpha_k, (d_{i,j})_{1 \le i < j \le k}) \in D_r$ where $f((d_{i,j})_{1 \le i < j \le k}) < d$. 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 $1 \le i < j \le k$. 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. The following theorem states that the algorithm decides Diverse-ACQ and gives an upper bound on the run time.

▶ **Theorem 1.** *The Diverse-ACQ problem is in XP (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 1 decides the Diverse-ACQ problem in time $\mathcal{O}\big(|R^I|^{2k} \cdot (|X|+1)^{k(k-1)} \cdot pol(|Q|, k)\big)$ where R^I is the relation from I with the most tuples and $pol(|Q|, k)$ is a polynomial in $|Q|$ and k.*

For any node t in the join tree, D_t denotes the data structure manipulated by Algorithm 1. On the other hand, for the complete subtree T' rooted at t, $D_{T'}$ denotes the goal of our computation, namely the set of tuples $e = (\alpha_1, \ldots, \alpha_k, (d_{i,j})_{1 \le i < j \le k})$ with $ext_{T'}(e) \neq \emptyset$. The key to the correctness of Algorithm 1 is to show that, on termination of the bottom-up traversal, $D_t = D_{T'}$ indeed holds for every node t in the join tree.

We briefly discuss the run time of the algorithm. $|R^I|^{2k} \cdot (|X|+1)^{k(k-1)}$ represents $|D_t|^2$, where $|D_t|$ is the maximal number of entries e in any D_t during an execution of the algorithm: $|R^I|$ restricts the number of mappings α_i in any $t(I)$, each $d_{i,j}$ can take at most $|X|+1$ different values (being the Hamming distance of mappings with at most $|X|$ variables), giving $(|X|+1)^{\frac{k(k-1)}{2}}$ different tuples $(d_{i,j})_{1 \le i < j \le k}$. $|D_t|^2$ is because the bottom-up traversal can be implemented via a nested loop, dominating the run time of the initialization and finalization steps. The polynomial factor $pol(|Q|, k)$ represents the computation of $\frac{k(k-1)}{2}$ Hamming distances between at most $|var(A)|$ variables (i.e., $k^2 \cdot |var(A)|$ where A is the atom in Q with the most variables), the number of nodes (i.e., $|Q|$), and the computation of the aggregate function f (i.e., some polynomial $pol_f(|X|, k)$ depending on $|X|$ and k).

Theorem 1 shows that the first phase of 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 $f((d_{i,j})_{1 \le i < j \le k}) \ge d$ 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.

3.1.2 W[1]-Hardness

Having proved XP-membership combined complexity of the Diverse-ACQ problem in Theorem 1, we now show that, for two important aggregators sum and min, a stronger result in the form of FPT-membership is very unlikely to exist. More specifically, we prove W[1]-hardness for query complexity and, hence, also for combined complexity in these cases.

▶ **Theorem 2.** *The problems Diverse$_{\mathsf{sum}}$-ACQ and Diverse$_{\mathsf{min}}$-ACQ, parameterized by the size k of the diversity set, are W[1]-hard. They remain W[1]-hard even if all relation symbols are of arity at most two and $Q(X)$ contains no existential variables.*

Proof sketch. The proof is by simultaneously reducing Independent Set parameterized by the size of the independent set to both Diverse$_{\mathsf{sum}}$-ACQ and Diverse$_{\mathsf{min}}$-ACQ. The only difference between the two reductions will be in how we define the diversity threshold d.

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 Diverse$_{\mathsf{sum}}$-ACQ and Diverse$_{\mathsf{min}}$-ACQ, respectively, 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

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

Finally, set $k = s$ and $d = \binom{k}{2} \cdot (m + 1)$ for Diverse$_{\mathsf{sum}}$-ACQ and $d = m + 1$ for Diverse$_{\mathsf{min}}$-ACQ, respectively. 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. First, 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 $\gamma_i \in Q(I)$. And, second, the desired diversities $d = \binom{k}{2} \cdot (m + 1)$ in case of sum and $d = m + 1$ in case of min, respectively, can only be achieved by k solutions that pairwisely differ on all variables. ◀

3.1.3 Speeding up the Basic Algorithm

Algorithm 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. However, specific diversity measures may require less information as will now be exemplified for δ_{sum}.

▶ **Theorem 3.** *The* $\mathsf{Diverse}_{\mathsf{sum}}$*-ACQ problem is in FPT query complexity when parameterized by the size k of the diversity set. More specifically,* $\mathsf{Diverse}_{\mathsf{sum}}$*-ACQ for an ACQ* $Q(X)$*, a database instance* I*, and integers* k *and* d*, can be solved in time* $\mathcal{O}(|R^I|^{2k} \cdot 2^{k(k-1)} \cdot pol(|Q|, k))$*, where* R^I *is the relation from* I *with the most tuples and* $pol(|Q|, k)$ *is a polynomial in* $|Q|$ *and* k*.*

Proof sketch. Note that $pol(|Q|, k)$ is the same as in Theorem 1. 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 $\gamma_1, \dots, \gamma_k$ over variables Z, the equality $\delta_{\mathsf{sum}}(\gamma_1, \dots, \gamma_k) = \sum_{z \in Z} \delta_{\mathsf{sum}}(\gamma_1|_z, \dots, \gamma_k|_z)$ holds. The reason we had to explicitly distinguish all possible values $(\alpha_1, \dots, \alpha_k, (d_{i,j})_{1 \leq i < j \leq k})$ in the basic algorithm is that, in general, given two collections $(\gamma_1, \dots, \gamma_k)$ and $(\gamma'_1, \dots, \gamma'_k)$ of mappings that agree on the shared variables, we cannot derive $\delta(\hat{\gamma}_1, \dots, \hat{\gamma}_k)$ for $\hat{\gamma}_i = \gamma_i \cup \gamma'_i$ from $\delta(\gamma_1, \dots, \gamma_k)$ and $\delta(\gamma'_1, \dots, \gamma'_k)$. In contrast, for δ_{sum}, this is possible. Hence, in principle, it suffices to store in $D_{T'}$ for each collection $(\alpha_1, \dots, \alpha_k)$ 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

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

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

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

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 $\alpha_1, \dots, \alpha_k$ it is not possible to determine whether the set of extensions $\gamma_1, \dots, \gamma_k$ 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 only those entries from D_t and $D_{t'}$ that are consistent with the values of $b_{i,j}$, giving the stated run 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 $\mathcal{O}(|R^I|^{2k} \cdot pol(|Q|, k))$. At first glance, this does not seem to strengthen 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 $(\alpha_1, \dots, \alpha_k, \dots)$, then D_t also contains entries of the form $(\alpha_{\pi(1)}, \dots, \alpha_{\pi(k)}, \dots)$ for all permutations π of $(1, \dots, k)$. 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 run 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 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 4.** *The problem $\mathsf{Diverse}_{\mathsf{sum}}$-ACQ is in P (query complexity) when the diversity set may contain duplicates and k is given in unary.*

Proof sketch. 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 tuples in $\lambda(t)^I$), 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 $d_{T'}(\alpha_1, \ldots, \alpha_k)$. Applying some basic combinatorics and assuming the size of I (and thus of $t(I)$) to be constant, we get that the number of entries in any D_t is in $\mathcal{O}(k^{|t(I)|-1})$. Using this upper bound for the size of $|D_t|$ instead of $|R^I|^k$ we get a polynomial run time. ◀

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 1 (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) [33]. 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 $\alpha \colon Z \to \mathit{dom}(I)$ 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 $\alpha \colon Z \to \mathit{dom}(I)$ 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 $m-1$ positions in the answer relation R^I. In this case, if there are at least k different values in the m-th component, the maximum is actually achieved by selecting k such tuples. But then there is no need to retain further tuples with the same values in the $m-1$ fixed positions. 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 exists at least $t!^2 \cdot k^t$ different value combinations, there also exist $t \cdot k$ tuples with pairwise maximum Hamming distance on the remaining positions and it is sufficient to only keep those.

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 5.** *The problem $\mathsf{Diverse}_{\mathsf{mon}}$-FO is in FPT 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 sketch. As mentioned above, we can transform in polynomial time any (fixed) FO-query into the form $Q(x_1, \dots, x_m) = R(x_1, \dots, x_m)$ over a database I with a single relation R^I. The reduction to an equivalent problem instance of size $\mathcal{O}(m!^2 \cdot k^m)$ is then achieved by applying $\mathbf{Red}_1$ through $\mathbf{Red}_m$ to I in this order exhaustively. The crucial property of the reduction rule $\mathbf{Red}_t$ with $t \in \{1, \dots, m\}$ is as follows:

Claim A. *Let $t \in \{1, \dots, m\}$ 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 $\boldsymbol{Red}_t$ 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 $\boldsymbol{Red}_t$ and let I_{new} denote its state after applying $\boldsymbol{Red}_t$. Let $\gamma_1, \dots, \gamma_k$ be pairwise distinct solutions in $Q(I_{old})$. Then there exist pairwise distinct solutions $\gamma'_1, \dots, \gamma'_k$ in $Q(I_{new})$ with $\delta(\gamma'_1, \dots, \gamma'_k) \geq \delta(\gamma_1, \dots, \gamma_k)$, i.e., the diversity achievable before deleting tuples from the database can still be achieved after the deletion.*

Note that a naive greedy algorithm always finds a witnessing Γ and the existence of this greedy algorithm implies the well-definedness. The safety follows from the fact that each γ_i that is removed, i.e., $\gamma_i \in Q(I)_\alpha \setminus \Gamma$, can be replaced by a $\gamma'_i \in \Gamma$ that is kept. Concretely, we can pick $\gamma'_i \in \Gamma$ such that $\delta(\dots, \gamma'_i, \dots) \geq \delta(\dots, \gamma_i, \dots)$. ◀

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 the two important aggregator functions sum and min. 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 [2] that this restricted problem remains NP-complete.

▶ **Theorem 6.** *The problems $\mathsf{Diverse}_{\mathsf{sum}}$-ACQ and $\mathsf{Diverse}_{\mathsf{min}}$-ACQ are NP-hard data complexity. They are NP-complete if the size k of the diversity set is given in unary.*

Proof sketch. 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.

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 $V(G) = \{v_1, \dots, v_n\}$ and $E(G) = \{e_1, \dots, e_m\}$. Then the database I consists of a single relation R^I with n tuples (= number of vertices in G) over the domain $dom(I) = \{\mathbf{free}_1, \dots, \mathbf{free}_n, \mathbf{taken}_1, \dots, \mathbf{taken}_m\}$. The i-th tuple in R^I will be denoted $(e_{i,1}, \dots, e_{i,5})$. For each $v_i \in V(G)$, the values $e_{i,1}, \dots, e_{i,5} \in dom(I)$ 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 $d_{\mathsf{sum}} = 5 \cdot \frac{k \cdot (k-1)}{2}$ and $d_{\mathsf{min}} = 5$ in the case of the $\mathsf{Diverse}_{\mathsf{sum}}$-ACQ and $\mathsf{Diverse}_{\mathsf{min}}$-ACQ problems, respectively. The resulting problem instances of $\mathsf{Diverse}_{\mathsf{sum}}$-ACQ and $\mathsf{Diverse}_{\mathsf{min}}$-ACQ are thus of the form $\langle I, Q, k, d_{\mathsf{sum}} \rangle$ and $\langle I, Q, k, d_{\mathsf{min}} \rangle$, respectively.

The reduction is clearly feasible in polynomial time. Its correctness hinges on the observation that the desired diversities $d_{\mathsf{sum}} = 5 \cdot \frac{k \cdot (k-1)}{2}$ and $d_{\mathsf{min}} = 5$ can only be reached by k answer tuples that pairwisely differ in all 5 positions. ◀

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 5 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. We show next that Diverse-UACQ is NP-hard (for the aggregators sum and min) even in a very restricted setting, namely a union of two ACQs and with desired size $k = 2$ of the diversity set.

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 $C(v) \subseteq \{1, \ldots, l\}, l \in \mathbb{N}$. Then a *coloring* is a function $c : V(G) \to \{1, \ldots, l\}$ 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 $C : V(G) \to 2^{\{1,\ldots,l\}}$.
Question: Does there exist a C-admissible coloring $c : V(G) \to \{1, \ldots, l\}$?

Clearly, List Coloring is a generalization of 3-Colorability and, hence, NP-complete. It was shown in [12], 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.

▶ **Theorem 7.** *The problems $\mathsf{Diverse}_{\mathsf{sum}}$-UACQ and $\mathsf{Diverse}_{\mathsf{min}}$-UACQ are NP-hard query complexity (and hence, also combined complexity). They remain 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 conjuncts and no existential variables. The problems are NP-complete if the size k of the diversity set is given in unary.*

Proof sketch. 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 the NP-hardness, first observe that δ_{sum} and δ_{min} coincide if we only allow two solutions. Hence, we may use a single diversity function δ to prove the NP-hardness for both Diverse$_{\mathsf{sum}}$-UACQ and Diverse$_{\mathsf{min}}$-UACQ.

For our problem reduction, we consider a fixed database I over a fixed schema, which consists of 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 $V = \{v_1, \dots, v_n\}$, $V' = \{v'_1, \dots, v'_n\}$, and $E = \{e_1, \dots, e_{3n}\}$. Note that $|V| = |V'|$ and $|E| = 3 \cdot |V|$ as each vertex in G has degree three and G is bipartite.

From this we construct a UACQ Q as follows: we use the $3n + 1$ variables $x_1, \dots, x_{3n}, y$ in our query. For each $i \in \{1, \dots, n\}$, 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 $Q(x_1, \dots, x_{3n}, y) := \varphi \vee \psi$ 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 $d = 3n + 1$ 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). ◀

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$^\neg$ [32]. 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 5. 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 2 has a treewidth of one. Hence, it is clear that also Diverse-CQ$^\neg$ is W[1]-hard combined complexity for queries with bounded treewidth. It remains to study the combined complexity, for which we describe an XP-algorithm next.

Our algorithm is based on so-called *nice* tree decompositions – a normal form introduced in [23]. 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 [23] 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 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 $(\alpha_1, \dots, \alpha_k, (d_{i,j})_{1 \leq i < j \leq k})$. 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 $\gamma_1, \dots, \gamma_k$ 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

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

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

Algorithm 2. 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, the 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

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

  Hence, we exhaustively go through all possible variable assignments $\alpha_1, \dots, \alpha_k \colon \chi(t) \to dom(I)$, 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

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

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 run time.

▶ **Theorem 8.** *For a class of CQs$^\neg$ of bounded treewidth, the problem Diverse-CQ$^\neg$ is in XP 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 2 solves Diverse-CQ$^\neg$ in time $\mathcal{O}(dom(I)^{2 \cdot k \cdot (\omega+1)} \cdot (|X|+1)^{k(k-1)} \cdot pol(|Q|, k))$, where $pol(|Q|, k)$ is a polynomial in $|Q|$ and k.*

Proof sketch. We briefly sketch how to arrive at the given run time. Note that the core ideas are similar to the ones of Algorithm 1. Firstly, for the bottom-up traversal, $dom(I)^{2 \cdot k \cdot (\omega+1)} \cdot (|X|+1)^{k(k-1)}$ is a bound for $|D_t|^2$. Thus, for each node t, we can simply use (nested) loops for the exhaustive searches and, as the checks only require polynomial time, compute each D_t in the required time bound. Then, evaluating f also only requires polynomial time and has to be applied at most $|D_r|$ many times. Lastly, computing an appropriate tree decomposition in the required time bound is possible due to [6] and [23]. ◀

We conclude this section by again stressing the analogy with Algorithm 1 for ACQs: First, we have omitted from our description of Algorithm 2 how to compute a concrete witnessing diversity set in the case of a yes-answer. This can be done exactly as in Algorithm 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 bounded treewidth, respectively. It should be noted that the restriction to acyclic CQs is less restrictive than it may seem at first glance. Indeed, our upper bounds (in particular, the XP- and FPT-membership results in Section 3) are easily generalized to CQs of bounded hypertree-width [18]. Moreover, recent empirical studies of millions of queries from query logs [7] and thousands of queries from benchmarks [17] have shown that CQs typically have hypertree-width at most 3.

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. Moreover, for future work, different settings could be studied. We mention several modifications below.

First, different parameterizations might be of interest. We have only considered the parameterization by the size k of the diversity set. Adding the hypertree-width (for Diverse-ACQ) and the treewidth (for Diverse-CQ$^\neg$) to the parameter would leave our XP-membership results unchanged. On the other hand, different parameterizations such as the threshold d on the diversity are left 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 run times are exponential in the parameter k. As we allow larger values of k, these run 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. As concrete aggregators, we have studied sum and min. This seems quite a natural choice since, for a fixed number k of answer tuples, avg behaves the same as sum and count makes no sense. Moreover, max is unintuitive if we want to achieve diversity *above* some threshold. However, a problem strongly related to Diversity is Similarity [15], where one is interested in finding solutions close to each other. In this case, max (and again sum) seems to be the natural aggregator. We leave the study of Similarity for future work.

References

1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995. URL: `http://webdam.inria.fr/Alice/`.

2 Paola Alimonti and Viggo Kann. Hardness of approximating problems on cubic graphs. In Gian Carlo Bongiovanni, Daniel P. Bovet, and Giuseppe Di Battista, editors, *Algorithms and Complexity, Third Italian Conference, CIAC '97, Rome, Italy, March 12-14, 1997, Proceedings*, volume 1203 of *Lecture Notes in Computer Science*, pages 288–298. Springer, 1997. `doi:10.1007/3-540-62592-5_80`.

3 Antoine Amarilli, Louis Jachiet, Martin Muñoz, and Cristian Riveros. Efficient enumeration for annotated grammars. In Leonid Libkin and Pablo Barceló, editors, *PODS '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*, pages 291–300. ACM, 2022. doi:10.1145/3517804.3526232.

4 Marcelo Arenas, Luis Alberto Croquevielle, Rajesh Jayaram, and Cristian Riveros. Efficient logspace classes for enumeration, counting, and uniform generation. In *Proc. PODS 2019*, pages 59–73. ACM, 2019. doi:10.1145/3294052.3319704.

5 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.

6 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.

7 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.

8 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.

9 Nofar Carmeli, Shai Zeevi, Christoph Berkholz, Benny Kimelfeld, and Nicole Schweikardt. Answering (unions of) conjunctive queries using random access and random-order enumeration. In *Proc. PODS 2020*, pages 393–409. ACM, 2020. doi:10.1145/3375395.3387662.

10 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, *Proceedings of the 9th Annual ACM Symposium on Theory of Computing, May 4-6, 1977, Boulder, Colorado, USA*, pages 77–90. ACM, 1977. doi:10.1145/800105.803397.

11 Surajit Chaudhuri and Rajeev Motwani. On sampling and relational operators. *IEEE Data Eng. Bull.*, 22(4):41–46, 1999. URL: http://sites.computer.org/debull/99dec/surajit.ps.

12 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.

13 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.

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

15 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.

16 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.

17 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.

18 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.

19 Marc H. Graham. On The Universal Relation. Technical report, University of Toronto, 1979.

20 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, *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*, pages 372–377. AAAI Press / The MIT Press, 2005. URL: http://www.aaai.org/Library/AAAI/2005/aaai05-059.php.

21 Linnea Ingmar, Maria Garcia de la Banda, Peter J. Stuckey, and Guido Tack. Modelling diversity of solutions. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 1528–1535. AAAI Press, 2020. URL: https://aaai.org/ojs/index.php/AAAI/article/view/5512.

22 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.

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

24 Yasuaki Kobayashi, Kazuhiro Kurita, and Kunihiro Wasa. Linear-delay enumeration for minimal steiner problems. In Leonid Libkin and Pablo Barceló, editors, *PODS '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*, pages 301–313. ACM, 2022. doi:10.1145/3517804.3524148.

25 Viktor Leis, Bernhard Radke, Andrey Gubichev, Alfons Kemper, and Thomas Neumann. Cardinality estimation done right: Index-based join sampling. In *Proc. CIDR 2017*. www.cidrdb.org, 2017. URL: http://cidrdb.org/cidr2017/papers/p9-leis-cidr17.pdf.

26 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.

27 Carsten Lutz and Marcin Przybylko. Efficiently enumerating answers to ontology-mediated queries. In Leonid Libkin and Pablo Barceló, editors, *PODS '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*, pages 277–289. ACM, 2022. doi:10.1145/3517804.3524166.

28 Timo Camillo Merkl, Reinhard Pichler, and Sebastian Skritek. Diversity of answers to conjunctive queries. *CoRR*, arXiv:2301.08848, 2023. doi:10.48550/arXiv.2301.08848.

29 Alexander Nadel. Generating diverse solutions in SAT. In Karem A. Sakallah and Laurent Simon, editors, *Theory and Applications of Satisfiability Testing - SAT 2011 - 14th International Conference, SAT 2011, Ann Arbor, MI, USA, June 19-22, 2011. Proceedings*, volume 6695 of *Lecture Notes in Computer Science*, pages 287–301. Springer, 2011. doi:10.1007/978-3-642-21581-0_23.

30 Thierry Petit and Andrew C. Trapp. Finding diverse solutions of high quality to constraint optimization problems. In Qiang Yang and Michael J. Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 260–267. AAAI Press, 2015. URL: http://ijcai.org/Abstract/15/043.

31 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.

32 Marko Samer and Stefan Szeider. Algorithms for propositional model counting. *J. Discrete Algorithms*, 8(1):50–64, 2010. doi:10.1016/j.jda.2009.06.002.

33 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, *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*, pages 137–146. ACM, 1982. doi:10.1145/800070.802186.

34 Mihalis Yannakakis. Algorithms for acyclic database schemes. In *Very Large Data Bases, 7th International Conference, September 9-11, 1981, Cannes, France, Proceedings*, pages 82–94. IEEE Computer Society, 1981.

35 C. T. Yu and M. Z. Özsoyoglu. An algorithm for tree-query membership of a distributed query. In *The IEEE Computer Society's Third International Computer Software and Applications Conference, COMPSAC 1979*, pages 306–312, 1979.

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

37 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.