TU WIEN Informatics

Erzeugung verschiedenartiger Lösungen von konjunktiven Anfragen und aussagenlogischen Formeln

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Logic and Computation

eingereicht von

Timo Merkl, BSc BSc
Matrikelnummer 11702806

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Mag.rer.nat. Dr.techn. Reinhard Pichler
Mitwirkung: Senior Lecturer Dipl.-Ing. Dr.techn. Sebastian Skritek

Wien, 1. September 2022

Timo Merkl | Reinhard Pichler

TU WIEN Informatics

Generating Diverse Solutions to Conjunctive Queries and Propositional Formulae

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Logic and Computation

by

Timo Merkl, BSc BSc
Registration Number 11702806

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Mag.rer.nat. Dr.techn. Reinhard Pichler
Assistance: Senior Lecturer Dipl.-Ing. Dr.techn. Sebastian Skritek

Vienna, 1st September, 2022

Timo Merkl | Reinhard Pichler

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Timo Merkl, BSc BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 1. September 2022

Timo Merkl

Acknowledgements

I would like to thank my supervisor Reinhard Pichler as well as my co-supervisor Sebastian Skritek for the pleasant working environment. The many positive discussions were illuminating and unquestionably contributed in enriching this thesis.

Kurzfassung

Die Anzahl der Lösungen für ein computationelles Problem kann enorm sein, weshalb die Ausgabe aller Möglichkeiten oft nicht ratsam ist. Daher sollte ein Solver nur eine kleine Anzahl aller Lösungen berechnen. Dies sollte jedoch keine willkürliche Auswahl an Lösungen sein, sondern eine Sammlung möglichst unterschiedlicher, damit der*die Nutzer*in den gesamten Lösungsraum besser erfassen kann.

Ausgehend davon beschäftigt sich die vorliegende Arbeit mit dem Problem der Berechnung einer Sammlung möglichst unterschiedlicher Antworten auf Datenbankabfragen – mit besonderem Fokus auf konjunktive Abfragen – und der Berechnung einer Sammlung möglichst unterschiedlicher Modelle aussagenlogischer Formeln. Dabei handelt es such um zwei der grundlegendsten Probleme in der Datenbanktheorie und der künstlichen Intelligenz. Zur Analyse dieser Probleme werden Techniken aus der parametrisierten Komplexität verwenden, insbesondere wird die Komplexität der Probleme an Azyklizitätsmaße, i.e., Baumweite und Hyperbaumweite, geknüpft.

Es werden sowohl theoretische Ergebnisse als auch konkrete Algorithmen angegeben, die so detailliert erklärt werden, dass eine Implementierung unkompliziert erfolgen kann. Konkret werden drei XP dynamische Programmieralgorithmen präsentiert, die jeweils für azyklische konjunktive Abfragen, konjunktive Abfragen mit Negation oder aussagenlogische Formeln entwickelt wurden. Für fixe Datenbankabfragen erster Ordnung wird darüber hinaus eine FPT-Kernelisierungsprozedur angegeben. Abschießend werden für die behandelten Probleme auch neue theoretische untere Schranken angeführt, welche außerdem zu den in der gegenwärtigen Arbeit etablierten oberen Schranken passen.

Abstract

The number of solutions to a computational problem can be tremendous and thus, presenting them all to the user is often not advisable. Instead, a solver should only output a small number of all solutions. However, these should not be arbitrary solutions but a diverse collection such that the user obtains a better grasp on the whole solution space.

Tackling this problem, in this thesis, we formally analyze the problem of computing a diverse collection of answers to database queries – in particular conjunctive queries (CQ) – and computing a diverse collection of models of propositional formulae (SAT). These are two of the most fundamental problems that arise in database theory and artificial intelligence. For our analysis, we apply techniques of parameterized complexity and to that end, we tie the complexity of the problems to acyclicity measures, i.e., treewidth and hypertreewidth.

We give theoretical results as well as concrete algorithms that are explained in such detail that an implementation thereof is straightforward. Concretely, we present three XP dynamic programming algorithms. These are designed for acyclic conjunctive queries, conjunctive queries with negation, and propositional formulas, respectively. Furthermore, for fixed first order database queries, we give an FPT kernelization procedure. As for theoretical results, we provide novel lower bounds for the diversity problems which match the upper bounds provided by the algorithms.

Contents

CHAPTER 1

Introduction

Most computational problems are defined with some notion of solution. That is, for an instance I of some computational problem $\mathcal{X}$ there is a set of solutions $\mathcal{S}(I)$. For example, one would call a truth assignment γ that satisfies a propositional formula φ a solution of the propositional satisfiability problem (SAT). Likewise, one can consider the answers to a conjunctive database query to be the solutions of the conjunctive query answering problem (CQ). There are many well established questions to analyze in combination with the problem $\mathcal{X}$ (e.g., "Is there a solutions?" or ''How many solutions are there?"). These questions all ask for/investigate/... different properties of $\mathcal{S}(I)$, therefore highlighting different aspects of $\mathcal{X}$ which in return leave us with a better understanding of the computational problem.

However, we would like to make a case for the fact that there is an important perspective when handling $\mathcal{S}(I)$ which is at the moment not well understood theoretically. That is, how to *navigate* through $\mathcal{S}(I)$ without materializing the possibly enormous number of solutions. The reason for this is it that although all elements of $\mathcal{S}(I)$ are solutions to the instance I, it is often not possible to model every aspect of a real world problem into I. This could be the case because it is not feasible to model some preference or something may have just been forgotten. Thus, an element of $\mathcal{S}(I)$ still may not solve the real world problem and we need to *search* for a real solution in $\mathcal{S}(I)$.

To further motivate this, consider a variation of the car dealership example of Hebrard et al. (2005). Let us say that I models the preference of a customer and $\mathcal{S}(I)$ are all cars that match these restrictions. Now, in a large dealership, it would not be feasible for the clerk to go through all cars $\mathcal{S}(I)$ with the customer. Instead, it would be better for them to go through a rather small list of cars that are very different to each other. With this, the clerk can rule out certain types of cars which the customer dislikes. In contrary, if there is a car among them that almost fits the customers desires, the clerk should concentrate on cars that are similar to this car.

This example gives us two basic problems to solve when wanting to *navigate* through $\mathcal{S}(I)$: the *diversity problem* – how to find a small collection $D \subseteq \mathcal{S}(I)$ of diverse solutions – and the *similarity problem* – how to find a small collection $S \subseteq \mathcal{S}(I)$ of similar solutions. These are dual to each other but require similar techniques (Hebrard et al., 2005; Eiter et al., 2013). We will mostly focus on the *diversity problem*, however we also explain how results carry over to the *similarity problem*.

In the literature, the diversity problem has been considered for many well known computational problems. This was done for (mixed) integer programming (Danna and Woodruff, 2009), answer set programming (Eiter et al., 2013), constraint satisfaction problems (Petit and Trapp, 2015; Ingmar et al., 2020), SAT (Nadel, 2011), and database queries (Drosou and Pitoura, 2010; Vieira et al., 2011; Deng and Fan, 2014). However, most techniques that have been developed are of heuristic nature and/or have worst case exponential runtime bounds. To that end, algorithms which are provably optimal and which are also tractable in the worst case are still missing for a lot of the mentioned problems.

1.1 Problem Statement

In this thesis we consider the problem of finding diverse solutions to database queries and models of propositional formulae (Diverse-SAT). For database queries, we will mostly focus on conjunctive queries (Diverse-CQ) and extensions thereof. The aim of this thesis is to present novel algorithms and complexity theoretical results for these problems.

For this, one first needs to formally specify what a diverse collection of solutions is. There is no one correct way to formalize this as it heavily depends on the context. Yet, as Ingmar et al. (2020) point out, it is natural to define the diversity of a collection by pairwisely comparing elements (solutions) and aggregating these values. Furthermore, one may want each pair of solutions to satisfy a minimal diversity property. The most basic example of such a constraint is that solutions are at least distinct from each other, i.e., to ask for a *set* of diverse solutions.

However, allowing diversity measures in such generality may make the problems appear harder in theory than they are when working with real world diversity measures. Thus, often concrete diversity measures are analyzed in the literature. The most basic are based on the pairwise Hamming distance of solutions (if they are either functions or sets). Usually, either the sum of the Hamming distances or the minimal pairwise Hamming distance is then considered as the diversity of the collection of solutions (Hebrard et al., 2005; Baste et al., 2019, 2022; Hanaka et al., 2021a,b). We will mostly restrict ourselves to these diversity measures in this thesis. Thus, unless mentioned otherwise, we mean this type of diversity (exact formal definitions are given in Chapter 4).

1.2 The Approach Taken

Classical complexity theory does not offer much insight into the problems at hand. This is because for both, SAT and CQ, just asking whether there exists some solution at all is already NP-hard. Hence, also finding multiple solutions which are additionally diverse to each other is intractable (Hebrard et al., 2005). Thus, we use acyclicity measures and parameterized complexity to analyze so-called *islands of tractability*. Informally speaking, we tie the hardness of an instance to its degree of cyclicity which in turn allows us to treat the problems SAT and CQ as if they were (almost) tractable.

With this, we can formally analyze how diversity impacts the hardness of the problems and establish when and in what sense computing diverse collections of solutions is tractable. Recall, that we only want to compute a small number of diverse solutions. Thus, following in the footsteps of Baste et al. (2019), we will consider the number of sought-after solutions as a parameter. This approach based in parameterized complexity sets this thesis apart from previous work in the area of diverse database query answering and SAT.

1.3 Outline and Overview of Results

The structure and the main contributions of this thesis are as follows: Firstly, an overview of related work is given in Chapter 2. Then, in Chapter 3, relevant notions and definitions are fixed. This includes notions of acyclicity, a short recapitulation of parameterized complexity, basic propositional logic, and basic database theory. Chapter 4 then focuses on the paradigm of diversity and the concrete problems that this thesis deals with.

Subsequently, the main results of the thesis are presented in Chapters 5 through 7. We start by tackling Diverse-CQ in Chapter 5, presenting a dynamic programming algorithm and a matching complexity theoretical lower bound. Furthermore, we also give a kernelization algorithm which is significantly better than the dynamic programming algorithm when we can assume the query to be fixed.

In Chapter 6, we slightly extend our query language. On the one hand, we show that introducing unions makes it intractable to find even two diverse solutions to acyclic queries. On the other hand, we show that when introducing negations, an only slightly more restrictive acyclicity measure is sufficient to guarantee tractability. To that end, we also present a dynamic programming algorithm for this acyclicity measure.

Then, in Chapter 7, we turn our attention to Diverse-SAT. Diverse-SAT can be reduced to a fragment of Diverse-CQ with added negations and hence, the previous results are transferrable. Furthermore, we present an improved dynamic programming algorithm, which only depends on a less restrictive acyclicity measure. Lastly, we briefly consider formulae in disjunctive normal form, give a complexity theoretical lower bound, and discuss a worst case optimal brute force approach for certain cases.

Finally, we conclude in Chapter 8 and give some directions for future work.

CHAPTER 2

Related Work

There exists a rich history of computing diverse solutions dating back to the 2000s (Bailleux and Marquis, 1999; Crescenzi and Rossi, 2002; Angelsmark and Thapper, 2004; Ziegler et al., 2005; Hebrard et al., 2005, 2007). In this chapter, we present works with results similar in nature to the ones of this thesis.

Analyzing the diverse variant of problems from the perspective of parameterized complexity with the number of sought-after solutions being a parameter rather recently came up as a research interest. To the best of our knowledge, the starting point can be traced back to an unpublished manuscript by Michael R. Fellows where the *Diverse $\mathcal{X}$ Paradigm* is outlined (according to Baste et al., 2019).

This is picked up by Baste et al. (2019) who consider the d-Hitting Set and Feedback Vertex Set problem. They managed to prove FPT results for both problems by using a network flow formulation where the size of a solution is considered as an additional parameter.

More recently, Baste et al. (2022) present a general method to solve the diverse variants of vertex related problems (e.g., Vertex Cover) with the help of treewidth. They, furthermore, show that several diverse graph theory problems admit polynomial kernels.

In the work of Hanaka et al. (2021b) (see also Hanaka et al., 2021a), the authors show how to use the color-coding technique and randomization to achieve FPT results for many graph related problems. This includes, for example, finding diverse l-paths. Moreover, they show that finding diverse spanning trees is possible in polynomial time.

A related, slightly different line of research focused on finding pairs of diverse solutions. That means that they consider the number of sought-after solutions to be fixed to two. Hardness results thus severely limit which parameterized algorithms can exist.

Crescenzi and Rossi (2002) look at binary constraint satisfaction problems and manage to show a complete classification in style similar to Schaefer (1978). In particular, they show

that computing two diverse solutions of 2-SAT or Horn-SAT instances is intractable in general. In fact, it is only doable in polynomial time in the following two trivial cases: if setting all variables to false and setting all variables to true works, or if flipping all truth values of a satisfying assignment is again a satisfying assignment.

In the work of Fomin et al. (2020), the authors look at the problem of finding a diverse pair of maximum matchings. Although finding a maximum matching in arbitrary graphs is possible in polynomial time, finding a diverse pair is NP-complete (follows from Holyer, 1981). Fomin et al. (2020) show, however, that this is possible in polynomial time on bipartite graphs and they give an FPT algorithm for general graphs where the parameter is the target diversity instead of the number of sought-after solutions.

CHAPTER 3

Preliminaries

In this chapter we will fix the terminology used throughout the whole thesis and introduce some theoretical background. This includes notions of acyclicity, parameterized complexity, basic propositional logic, and basic database theory.

We start by fixing some notation. In formulae, we will use lower case letters (x) to denote variables and upper case letters (X) to denote sets of variables. By slight abuse of notation, we will treat tuples and sets in the same way when no confusion arises. For a set of sets $\mathcal{A}$ we write $\bigcup \mathcal{A}$ as a shorthand for $\bigcup_{A \in \mathcal{A}} A$. For positive integers $n \in \mathbb{N}$ we denote $\{0, \ldots, n\}$ by $[n]$. We use $\binom{A}{n}$ to denote the n element subsets of A.

Let $f\colon X \to A$ be some function and Y an arbitrary set. We call $f|_Y\colon X \cap Y \to A$ defined by $f|_Y(y) = f(y), y \in X \cap Y$ the *restriction* of f on Y. Conversely, if $Y \subseteq X$ we call the function $f\colon X \to A$ an *extension* of $f|_Y\colon Y \to A$. We write $f \cong h$ to denote that two functions $f\colon X \to A, h\colon Z \to B$ agree on $X \cap Z$, i.e., $f|_{X \cap Z} = h|_{X \cap Z}$. Furthermore, for $f \cong h$ we define the (well-defined) functions $f \cup h\colon X \cup Z \to A \cup B$ and $f \cap h\colon X \cap Z \to A \cap B$. For $x \in X, z \in Z, y \in X \cap Z$ we define $(f \cup h)(x) = f(x)$, $(f \cup h)(z) = h(z)$, and $(f \cap h)(y) = f(y) = h(y)$. Lastly, we define $f(Y) = \{f(y) : y \in Y\}$ and $f(x_1, \ldots, x_n) = (f(x_1), \ldots, f(x_n))$ for $(x_1, \ldots, x_n) \subseteq X$.

3.1 Hypergraphs and Acyclicity

We assume the reader is familiar with basic graph theory. We will use the term graph to refer to simple graphs, i.e., finite undirected graphs where edges are only drawn between exactly two vertices. In contrast, a *hypergraph* is a tuple $H = (V, E)$ where V is a finite set of elements called *vertices* or *nodes* and $E \subseteq 2^E \setminus \emptyset$ is a set of *edges*. Therefore, hypergraphs are a strict generalization of graphs. When the vertex and edge set are not explicitly named, we will refer to the vertex set by $V(H)$ and the edge set by $E(H)$. For the sake of simplicity, we neglect hypergraphs with isolated vertices. For graphs G we

denote the subgraph induced by $V' \subseteq V(G)$ as $G[V'] = (V', E(G) \cap \binom{V'}{2})$. When possible, we will always assume that a tree T is rooted in some $r \in V(T)$. Then, for a $t \in V(T)$, we denote the subtree rooted in t by T_t.

Acyclic graphs are an important class of graphs for which many computational problems become efficiently solvable. The same is true for hypergraphs, but there are multiple notions of acyclicity when working with hypergraphs (see Brault-Baron, 2016). We only require the so-called α-acyclicity and we will therefore call hypergraphs *acyclic* if they are α-acyclic.

Definition 1 (Join tree, α-acyclicity)**.** A *join tree* of a hypergraph H is a tree T together with a labeling $\lambda : V(T) \to E(H)$ that satisfies the following properties:

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

A hypergraph H is called (α-)*acyclic* if there exists a join tree of H.

Given a hypergraph, one can check in linear time whether it is acyclic and in the case where it is acyclic, also the join tree can be computed in linear time (Graham, 1979; Yu and Özsoyoğlu, 1979; Tarjan and Yannakakis, 1984).

To extend the class of (hyper)graphs for which computational problems are efficiently solvable, generalizations of acyclicity are regularly considered. The most prominent notions being treewidth (Robertson and Seymour, 1984) to generalize acyclic graphs and hypertreewidth (Gottlob et al., 2002a) to generalize acyclic hypergraphs. These express the degree of cyclicity of a graph, where high values correspond to highly cyclic graphs. To define these notions we first need to define tree and hypertree decompositions.

Definition 2 (Tree decomposition)**.** A *tree decomposition* of a hypergraph H is a tree T together with a labeling $\chi : V(T) \to 2^{V(H)}$ that satisfies the following properties:

1. Every $v \in V(H)$ appears in some $\chi(t), t \in V(T)$.
2. Every edge $e \in E(H)$ is fully contained in some $\chi(t), t \in V(T)$.
3. For every $v \in V(H)$, the set $T_v = \{t \in V(T) : v \in \chi(t)\}$ induces a connected subtree $T[T_v]$.

For a subtree T' of T we write $\chi(T')$ as a shorthand for $\bigcup \chi(V(T'))$.

Definition 3 (Hypertree decomposition)**.** A *hypertree decomposition* of a hypergraph H is a tree T together with two labeling $\chi : V(T) \to 2^{V(H)}$, $\lambda : V(T) \to 2^{E(H)}$ that satisfies the following properties:

1. The tuple (T, χ) is a tree decomposition of H.
2. For each $t \in V(T)$ we have $\chi(t) \subseteq \bigcup \lambda(t)$.
3. For each $t \in V(T)$ we have $\bigcup \lambda(t) \cap \chi(T_t) \subseteq \chi(t)$.

With this we can proceed to define treewidth and hypertreewidth.

Definition 4 (Treewidth)**.** The width of a tree decomposition (T, χ) is $\max_{t \in V(T)} |\chi(t)| - 1$ and the *treewidth* $tw(H)$ of a hypergraph H is the minimal width over all tree decompositions of H.

Definition 5 (Hypertreewidth)**.** The width of a hypertree decomposition (T, χ, λ) is $\max_{t \in V(T)} |\lambda(t)|$ and the *hypertreewidth* $hw(H)$ of a hypergraph H is the minimal width over all hypertree decompositions of H.

Examples of join trees, tree decompositions, and hypertree decompositions are given in Sections 3.3 and 3.4.

Both width optimal tree and hypertree decompositions can be computed in polynomial time for classes of hypergraphs with bounded $tw(H)$ and $hw(G)$ (Bodlaender, 1996; Gottlob et al., 2002a), respectively, and thus, algorithms which work well with low width decompositions can be used on these graph classes. This is also precisely the reason why we need property three of Definition 3. Without this property, even identifying graph classes of bounded hypertreewidth would be NP-hard (Gottlob et al., 2009). Furthermore, it is know that $hw(H) \leq 3 \cdot tw(H) + 2$ (Adler et al., 2007) and thus, algorithms that profit from low hypertreewidth are preferable to algorithms that profit from low treewidth.

To simplify algorithms that use tree decompositions, one often assumes the decomposition to be in some normal form. For that matter, a *nice tree decomposition* (Kloks, 1994) is a tree decomposition (T, χ) where each node $p \in V(T)$ is either a leaf, an *introduce node*, a *forget node*, or a *join node*. An introduce node has a single child $t \in V(T)$ with $\chi(t) \subseteq \chi(p)$ and $|\chi(t)| + 1 = |\chi(p)|$ (the element $x \in \chi(p) \setminus \chi(t)$ is *introduced*). A forget node has a single child $t \in V(T)$ with $\chi(t) \supseteq \chi(p)$ and $|\chi(t)| - 1 = |\chi(p)|$ (the element $x \in \chi(t) \setminus \chi(p)$ is *forgotten*). Lastly, a join node has exactly two children $t_1, t_2 \in V(T)$ and $\chi(t_1) = \chi(p) = \chi(t_2)$. Transforming an arbitrary tree decomposition (T, χ) into a nice tree decomposition (T', χ') can be done efficiently, does not increase the width of the decomposition, and only increases the number of nodes by a constant factor (Kloks, 1994).

3.2 Parameterized Complexity

In classical complexity theory, one only analyzes the impact of the instance size on the asymptotic runtime of an algorithm. This, thus, conceals the fact that different parts of instances may impact the runtime in different ways. Consider for example the problem

of answering a database query. Classically one would call this problem intractable in general. However, the reason why answering database queries is feasible in practice can be attributed in part to the fact that real world queries are usually very small (in comparison to the database) and the size of the query is the root of the intractability (Vardi, 1982).

The framework of parameterized complexity tries to provide a formal body to conduct these more fine grained analyses and was pioneered by Downey and Fellows in the 1980s and 1990s (Downey and Fellows, 1999). A more recent introduction can be found in Cygan et al. (2015), Downey and Fellows (2013). The idea of parameterized complexity is to describe an instance I by its size n and an additional *parameter* $k \in \mathbb{N}$ (in our example the size of the query $|Q|$). This additional parameter k can be any positive integer associated with the instance but should ideally explain the high asymptotic runtime of an algorithm and be reasonably small in practice. Therefore, the number of sought-after solutions k in a diversity problem or acyclicity measures are suitable parameters.

The first important parameterized complexity class is FPT. An algorithm A is called *fixed-parameter tractable* (FPT) if it runs in time $f(k) \cdot n^c$, where f is a computable function, c a constant, k the parameter, and n the size of the instance. A parameterized decision problem $\mathcal{X}$ is called FPT (or is in the class FPT) if there exists an FPT algorithm A that solves $\mathcal{X}$. Importantly, the degree of the polynomial dependency on n does not depend on the parameter. For XP problems precisely this restriction is weakened.

A parameterized algorithm A is said to run in XP time (XP algorithm) if it terminates after at most $\mathcal{O}(n^{f(k)})$ steps, where f is a computable function, k the parameter, and n the size of the instance. Analogously, a parameterized decision problem $\mathcal{X}$ is in the class XP if there exists an XP algorithm A that solves $\mathcal{X}$.

An important question in parameterized complexity is whether a problem admits an FPT algorithm, an XP algorithm, or neither. The most successful tool to establish negative results are, as in classical complexity theory, reductions, in this case *fpt-reductions*. An *fpt-reduction* R maps instances I of $\mathcal{X}$ with parameter k to instances $R(I)$ of $\mathcal{X}'$ with parameter k' such that

- I is a Yes-instance if and only if $R(I)$ is a Yes-instance,
- the parameter k' is less or equal to $h(k)$, where h is a fixed computable function (fixed by R), and
- R can be computed in time $f(k) \cdot n^c$, where n is the size of I, and c a constant (fixed by R).

The classes FPT and XP are closed under fpt-reductions. But not all problems that are assumed to not be in FPT are assumed to be XP-hard. Most importantly, these are problems that lie in the so called *weft hierarchy*, i.e., the classes $\mathsf{W}[1], \mathsf{W}[2], \ldots, \mathsf{W}[P]$. These classes are not defined by the runtime of the algorithms that solve its problems

and a proper definition is omitted at this point. For this theses it is just important to note that they are also closed under fpt-reductions and are nested with the classes FPT and XP as follows:

$$\mathsf{FPT} \subseteq \mathsf{W}[1] \subseteq \mathsf{W}[2] \subseteq \cdots \subseteq \mathsf{W}[\mathrm{P}] \subseteq \mathsf{XP}$$

Deciding whether a graph has an independent set of size at least k is a classical W[1]-complete problem (k is the parameter). The parameterized problem Independent-Set is formally defined as follows:

Independent-Set
Input: A graph G and a positive integer k.
Parameter: k.
Solution: A set $I \subseteq V(G)$ with $\lvert I\rvert \geq k$ and such that each pair of vertices from I is not adjacent in G.

It is widely assumed that FPT $\neq$ W[1] (see Flum and Grohe, 2006; Downey and Fellows, 2013) and hence, for practical purposes W[1]-hardness suffices to rule out the existence of an FPT algorithm. To rule out the existence of an XP algorithm, is suffices to show that a problem remains NP-hard even when the parameter is assumed to be bounded by some constant.

3.3 Propositional Formulae

A *propositional formula* is a well-formed formula where the building blocks are (propositional) variables ($x_1, x_2, \ldots$), the unary negation symbol ($\neg$), and the logical connectives *and* ($\wedge$) and *or* ($\vee$). A positive or negated variable is called *literal* and a disjunction of literals a *clause*. A truth assignment is a mapping $\gamma : X \to \{0, 1\}$ where X is a set of variables. We denote the set of variables that appear in a formula φ by $\mathbf{var}(\varphi)$. Let $\gamma : X \to \{0, 1\}$ be a truth assignment such that $\mathbf{var}(\varphi) \subseteq X$. We evaluate φ under γ by replacing all variables in φ in accordance to γ and by interpretating the logical connectives in the usual way. If γ satisfies φ, i.e., φ evaluates to true (1) under γ, we write $\gamma \vDash \varphi$.

A truth assignment that satisfies φ and is defined on exactly $\mathbf{var}(\varphi)$ is called a model of φ and the set of all models of φ is denoted by $\mathcal{M}(\varphi)$. With this we can define the well-known computational problem SAT:

Propositional Satisfiability (SAT)
Input: A propositional formula φ.
Solution: A model $\gamma \in \mathcal{M}(\varphi)$.

Deciding whether a proposition formula has a model is classically NP-complete (Cook, 1971; Levin, 1973).

A formula φ is in *disjunctive normal form* (DNF) if it is a disjunction of conjunctions, i.e., of the form $\varphi = \bigvee_{i=1}^{n} D_i$, where D_i are of the form $D_i = \bigwedge_{j=1}^{m_i} l_{i,j}$ and $l_{i,j}$ are literals. Conversely, φ is said to be in *conjunctive normal form* (CNF) if it is a conjunction of disjunctions, i.e., of the form $\varphi = \bigwedge_{i=1}^{n} C_i$, where C_i are clauses $C_i = \bigvee_{j=1}^{m_i} l_{i,j}$ and $l_{i,j}$ are again literals. It is often simpler to use set notation when handling formulae in CNF. For this, by abuse of notation, we say $\varphi = \{C_1, \ldots, C_n\}$. W.l.o.g we always assume D_i and C_i to neither have duplicate literals nor dual literals. This can be ensured by a simple preprocessing step.

Example 1. $\varphi = (x \vee y) \wedge (\neg x \vee y) \wedge (y \vee z) \wedge (\neg y \vee u) \wedge (u \vee v)$ is a CNF formula and $\psi = (\neg x \wedge y \vee \wedge z) \vee (y \wedge z) \vee (x \wedge \neg z)$ is a DNF formula. The truth assignment $\gamma = \{x \mapsto 1, y \mapsto 1, z \mapsto 0, u \mapsto 1, v \mapsto 0\}$ is a model of φ.

We will refer to the problems where the input formulae must be in CNF or DNF as CNF-SAT and DNF-SAT, respectively.

It is well known that it remains NP-complete to decide whether a CNF formula has model while it is possible to decide this for a DNF formula in polynomial time (Karp, 1972).

To describe the structure of a propositional formula φ in CNF we will use two graphs. Firstly, the *primal graph* $G_p(\varphi)$ of φ has $\mathbf{var}(\varphi)$ as its vertex set and an edge is drawn between two variables x, y if and only if they appear together in some clause C_i, i.e., $\{x, y\} \subseteq \mathbf{var}(C_i)$. Secondly, the *incidence graph* $G_i(\varphi)$ of φ is a bipartite graph with vertex sets $\mathbf{var}(\varphi)$ and $\varphi = \{C_1, \ldots, C_n\}$. A variable x is connected to a clause C_i if and only if $x \in \mathbf{var}(C_i)$. The *primal treewidth* $tw_p(\varphi)$ is the treewidth of $G_p(\varphi)$ and, likewise, the *incidence treewidth* $tw_i(\varphi)$ is the treewidth of $G_i(\varphi)$. It is known that a formula with primal treewidth k has at most an incidence treewidth of $k + 1$ (Kolaitis and Vardi, 2000).

Example 2. The primal and incidence graph of the formula φ of Example 1 can be seen in Figure 3.1. The clauses are $C_1 = x \wedge y, C_2 = \neg x \wedge y, C_5 = y \wedge z, C_4 = \neg y \wedge u, C_5 = u \wedge v$. In Figure 3.2, width optimal nice tree decompositions thereof are depicted.

Lastly, we note that the primal treewidth can equivalently be defined as the treewidth of the hypergraph of φ. The hypergraph of φ has again $\mathbf{var}(\varphi)$ as its vertex set but for each clause C_i a single edge is drawn, i.e., the edge $\mathbf{var}(C_i)$. Furthermore, in this setting it would make sense to consider the hypertreewidth and CNF formulae with acyclic hypergraphs. However, it is already NP-hard to decide whether a formula has a model even when the formulae are restricted to CNF formulae with acyclic hypergraphs (Samer and Szeider, 2010). Thus, considering this structural restriction cannot lead to tractable algorithms that find diverse solutions and is thus not further considered.

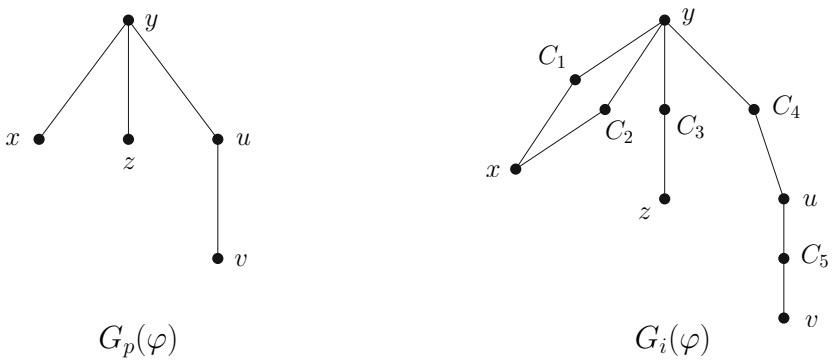


Figure 3.1: The graphs $G_p(\varphi)$ and $G_i(\varphi)$.

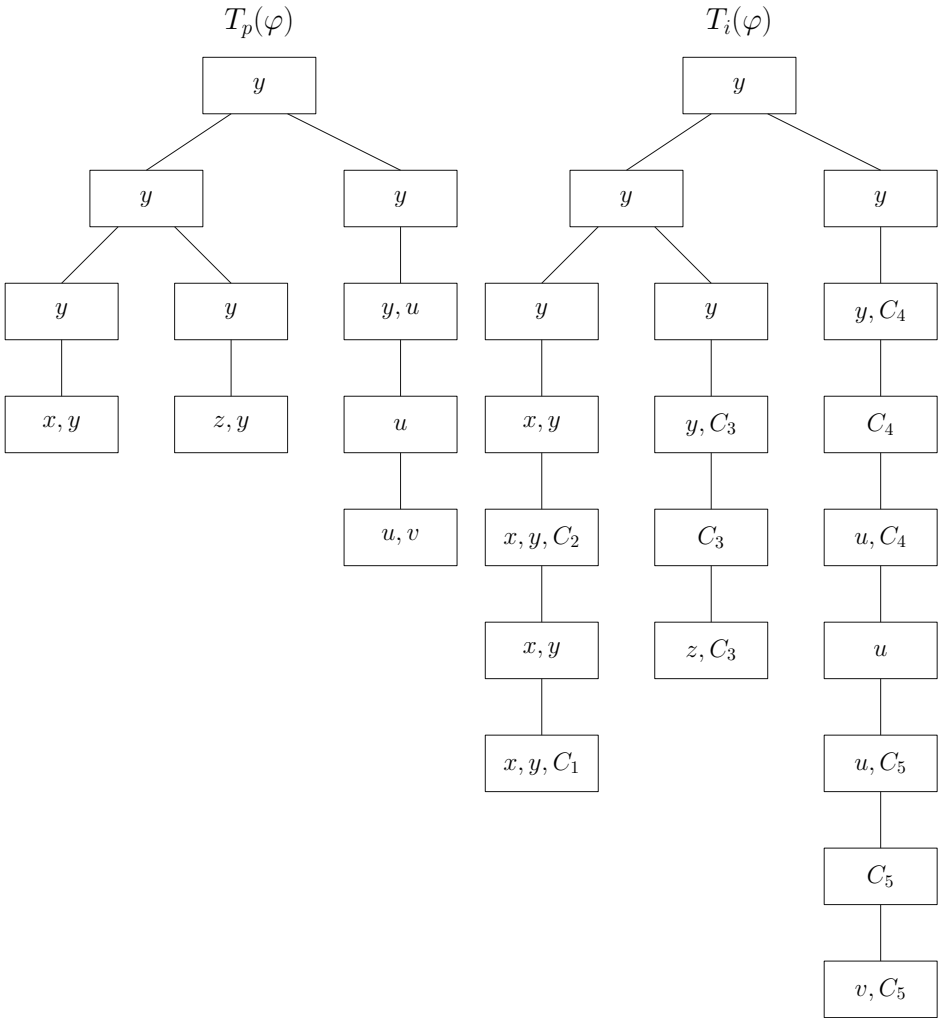


Figure 3.2: Tree decompositions $T_p(\varphi)$ and $T_i(\varphi)$.

3.4 Basic Database Theory

A *relational schema* $\mathbf{R}$ is a set of relation symbols $\mathbf{R} = \{R_1, \ldots, R_n\}$, each having a fixed arity $m_i \in \mathbb{N}$. An *instance* I over a relational schema $\mathbf{R}$ consists of a domain $\mathbf{dom}(I)$ and assigns each relation symbol R_i a relation $R_i^I \subseteq \mathbf{dom}(I)^{m_i}$. We also call I a *database* and the R_i^I are the relations (*tables*) of the database. W.l.o.g. we will assume that $\mathbf{dom}(I)$ only consists of the elements which also appear in at least one table.

Using the relation symbols of a relation schema $\mathbf{R}$ as predicates, we can construct *first order* (FO) formulae φ over $\mathbf{R}$. These consist of atoms $R_i(Z)$, negations ($\neg$), conjunctions ($\wedge$), disjunctions ($\vee$), and universal ($\forall x$) and existential quantifiers ($\exists x$). They are structured in the natural way and Z is an m_i-tuple of variables.

A mapping $\gamma : Z \to \mathbf{dom}(I)$ satisfies an atom $R_i(Z)$ over the database I if $\gamma(Z) \in R_i^I$. By the usual inductive definition we say that a mapping $\gamma : X \to \mathbf{dom}(I)$ satisfies the formula $\varphi(X)$ with free variables X over I if φ evaluates to true under γ (the quantifiers quantify over $\mathbf{dom}(I)$). We omit the reference to I if the database is clear from the context.

A *query* Q over a relational schema $\mathbf{R}$ is of the form $Q\colon ans(X) \leftarrow \varphi(X)$, where φ is a first order formula over $\mathbf{R}$ with free variables X. An *answer* of Q with respect to some database I is a mapping $\gamma : X \to \mathbf{dom}(I)$ that satisfies φ. The set of all answers is denoted as $I(Q)$. With this we can define the problem of first order query evaluation.

First Order Query (FOQ)
Input: A query Q and database I over a relational schema $\mathbf{R}$.
Solution: A mapping $\gamma \in I(Q)$.

Deciding whether an arbitrary first order query has a solutions over a database is well know to be PSPACE-complete. (Chandra and Merlin, 1977):

We call Q a *conjunctive query with negation* ($\text{CQ}^\neg$) if φ is of the form $\exists Y \psi(X, Y)$ and ψ is a conjunction of literals, i.e., $\psi = \bigwedge_{j=1}^{m} L_j$ and each L_j is a literal. If all literals are positive, we refer to Q as a *conjunctive query* (CQ). Conjunctive queries are one of the most basic and important database queries and correspond to select-from-where SQL statements and select-project-join relational algebra expressions.

A slightly larger class of queries are *unions of conjunctive queries* (UCQs). There, φ is of the form $\bigvee_{j=1}^{l} \exists Y_j \psi_j(X, Y_j)$ and each ψ_j is a conjunction of atoms. The set of answers corresponds to the union of the answers to $ans(X) \leftarrow \exists Y_j \psi_i(X, Y_j)$, hence the name.

We denote the restrictions of the computational problem FOQ to $\text{CQ}^\neg$, CQ, and UCQ also by $\text{CQ}^\neg$, CQ, and UCQ, respectively. Deciding whether there exists a solution is NP-complete for all three problems (Chandra and Merlin, 1977).

We remark at this point that we neither allow constants nor equalities in queries as these can be removed by a simple preprocessing step. Furthermore, from this point forward we will assume that each variable appears at most once in an literal of a CQ$^\neg$/CQ and the set of variables of two distinct literals must differ by at least one variable. These restrictions simplify the augmentations below without impairing the generality of the statements made in this thesis (cf. Gottlob et al., 2001).

To describe the structure of a CQ$^\neg$ Q, we define the hypergraph $H(Q)$ of Q as the hypergraph with vertex set $\mathbf{var}(\varphi)$ and an edge Z_i for each literal $L_i(Z_i)$. In accordance with the nomenclature for propositional formulae, we call $tw(H(Q))$ the primal treewidth of Q and denote it by $tw_p(Q)$.

We call a CQ Q an *acyclic conjunctive query* (ACQ) if the hypergraph $H(Q)$ is acyclic and a join tree/hypertree decomposition of $H(Q)$ also called a join tree/hypertree decomposition of Q. As each atom has a distinct set of variables, we can 1-1 associate the variable sets $\lambda(t)$ of a join tree $(T, \lambda), t \in V(T)$ with the atoms of Q. We will thus consider $\lambda(t)$ to be an atom and $Q\colon ans(X) \leftarrow \exists Y \bigwedge_{t \in V(T)} \lambda(t)$. Likewise, we define $hw(Q) = hw(H(Q))$ and, for hypertree decompositions (T, χ, λ) of Q, we treat $\lambda(t)$ to be a set of atoms of Q.

Example 3. The query

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

is acyclic, while the query

$$Q' : ans(x_1, x_2, x_3, x_4) \leftarrow R_1(x_1, x_2) \wedge R_2(x_2, x_3) \wedge R_3(x_1, x_3) \wedge R_4(x_3, x_4)$$

is cyclic due to the cycle within $R_1(x_1, x_2), R_2(x_2, x_3)$, and $R_3(x_1, x_3)$. The hypergraphs $H(Q)$ and $H(Q')$ can be seen in Figure 3.3. In Figure 3.4, a join tree of Q is depicted in addition to a width-optimal hypertree decomposition of Q'. The values $\lambda(t)$ are depicted as (sets of) atoms.

Analogously, a union of conjunctive queries $Q\colon ans(X) \leftarrow \bigvee_{j=1}^{l} \exists Y_j \psi_j(X, Y_j)$ is called a *union of acyclic conjunctive queries* (UACQs) if all $ans(X) \leftarrow \exists Y_j \psi_i(X, Y_j)$ are acyclic.

We will refer to the computational problem FOQ where we restrict ourselves to ACQs and UACQs also as ACQ and UACQs, respectively. Finding a solutions is doable in polynomial time for these problems due to Yannakakis' algorithm (Yannakakis, 1981). In contrast, parameterized by the primal treewidth $tw_p(Q)$ and hypertreewidth $hw(Q)$, respectively, for CQ$^\neg$ and CQ it is W[1]-hard and in XP to decide whether there exists a solutions over some database (Gottlob et al., 2002a,b). Moreover, the stronger parameter of primal treewidth is needed for CQ$^\neg$ as the problem remains NP-hard even on CQs$^\neg$ with acyclic hypergraphs (Samer and Szeider, 2010).

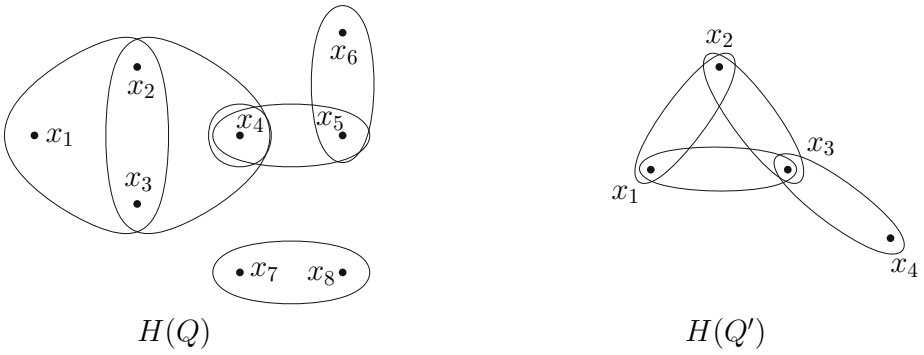


Figure 3.3: The hypergraphs $H(Q)$ and $H(Q')$.

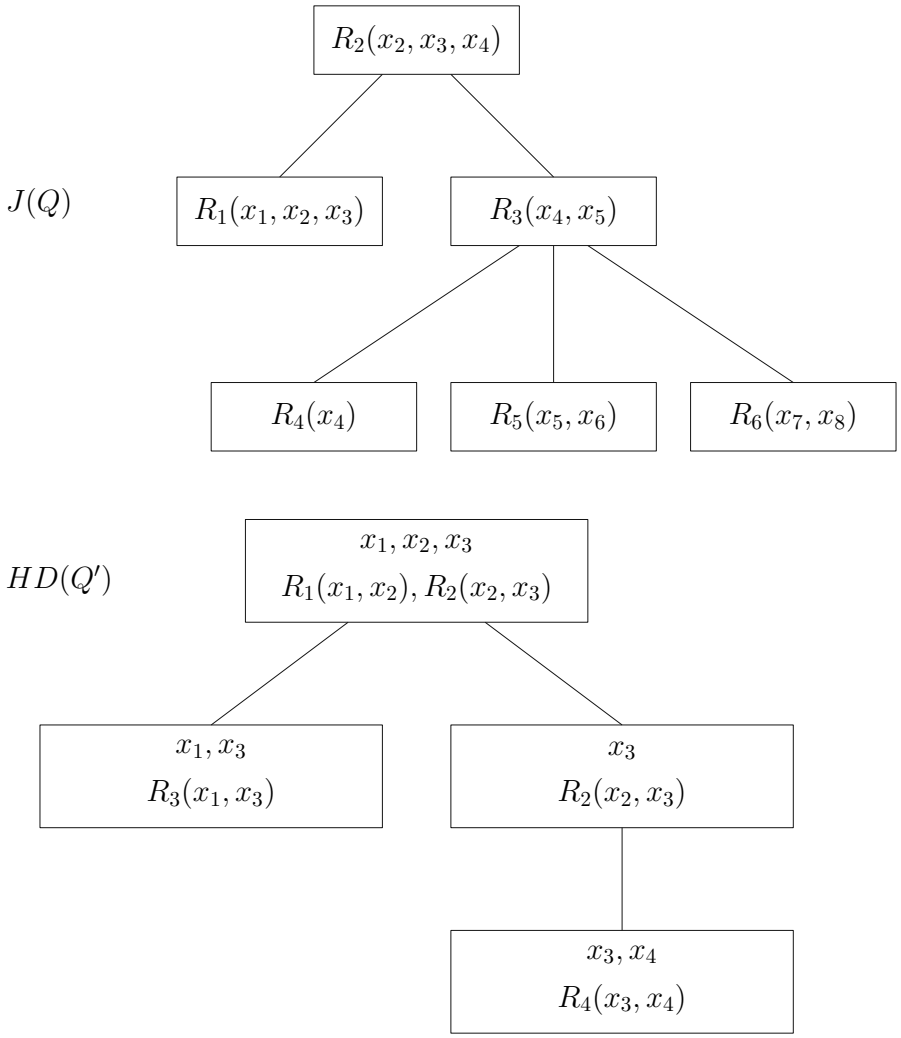


Figure 3.4: A join tree $J(Q)$ and a hypertree decomposition $HD(Q')$.

CHAPTER 4

Diversity and Problem Definitions

The approach to diversity used in this thesis loosely follows the *Diverse $\mathcal{X}$ Paradigm* published by Baste et al. (2019) (which, according to them, was proposed by Michael R. Fellows). In the Diverse $\mathcal{X}$ Paradigm, the $\mathcal{X}$ stands for a problem where the instances and solutions are defined, but not the question or task. A generic problem in this setting looks as follows:

$\mathcal{X}$
Input: An instance I of problem $\mathcal{X}$.
Solution: A solution γ of $\mathcal{X}$ for the instance I.

We denote in this generic setting the set of solutions of an instance I by $\mathcal{S}(I)$.

Now, one can ask many different questions about $\mathcal{X}$ and complexity results may not be translatable from one variant of the problem to the other. Classically, one would ask if there exists a solutions, i.e., the question "$\mathcal{S}(I) \neq \emptyset$?". However, there are many other natural questions to answer about $\mathcal{S}(I)$. This includes the questions: "How many different solutions are there in $\mathcal{S}(I)$?", "How does an example of a solution $\gamma \in \mathcal{S}(I)$ look like?", and "How does the whole set of solutions $\mathcal{S}(I)$ look like?". These correspond to different variants of the problem that originate from the same definition of instance and solution, namely a non-emptiness problem, a counting problem, a search problem, and an enumeration problem. Analyzing one and the same problem from these different perspectives leads to a deeper understanding of said problem.

Problems that are "easy to decide, but hard to count" (e.g. DNF-SAT) exemplify that results from one variant are not immediately transferrable to another variant. Thus, to ensure that no confusion arises, when talking about the complexity of the problem $\mathcal{X}$, we refer to the question whether there is a solution $\gamma \in S(I)$, i.e., "$\mathcal{S}(I) \neq \emptyset$?".

The Diverse $\mathcal{X}$ Paradigm tries to bridge the gap between asking for a single solution (a $\gamma \in \mathcal{S}(I)$) and asking for all solutions (the whole set $\mathcal{S}(I)$), by asking for a fixed number of solutions k (where k is usually small), which, as an additive property, are in some sense *diverse* to each other. This paradigm can thus be applied, similar as the questions above, to most computational problems and adds another perspective from which to look at a problem. It just requires an additional function δ (diversity measure) that expresses the diversity of a set of solutions. The *diverse* version of the problem X is defined as:

$\mathsf{Diverse}_\delta$-$\mathcal{X}$
Input: An instance I of problem $\mathcal{X}$ and positive integers k, d.
Question: Do there exist (pairwise distinct) solutions $\gamma_1, \ldots, \gamma_k \in \mathcal{S}(I)$ of I such that $\delta(\gamma_1, \ldots, \gamma_k) \geq d$?

We usually consider the number of sought-after solutions k as a parameter to the problem $\mathsf{Diverse}_\delta$-$\mathcal{X}$. Furthermore, we will differentiate between bag and set semantics, i.e, where two solutions $\gamma_i, \gamma_j, i \neq j$ are allowed to coincide and where they have to be distinct, respectively. The problem $\mathsf{Diverse}_\delta$-$\mathcal{X}$ can then be analyzed as a parameterized decision problem. In this thesis, we will mostly deal with two diversity measures: δ_{sum} and δ_{min}. These are, summed Hamming distance and minimal Hamming distance. For this purpose assume, as is the case for all problems considered in this thesis, that the solutions $\gamma_1, \ldots, \gamma_k$ are mappings $\gamma_i : X \to A$. The Hamming distance Δ of two solutions γ_i, γ_j is then defined as

$$\Delta(\gamma_i, \gamma_j) = \sum_{x \in X} \gamma_i(x) \neq \gamma_j(x).$$

The diversity measures are then, respectively, defined as

$$\delta_{\mathsf{sum}}(\gamma_1, \ldots, \gamma_k) = \sum_{1 \leq i < j \leq k} \Delta(\gamma_i, \gamma_j),$$
$$\delta_{\mathsf{min}}(\gamma_1, \ldots, \gamma_k) = \min_{1 \leq i < j \leq k} \Delta(\gamma_i, \gamma_j).$$

We will write $\mathsf{Diverse}_{\mathsf{sum}}$-$\mathcal{X}$ for $\mathsf{Diverse}_{\delta_{\mathsf{sum}}}$-$\mathcal{X}$, and $\mathsf{Diverse}_{\mathsf{min}}$-$\mathcal{X}$ for $\mathsf{Diverse}_{\delta_{\mathsf{min}}}$-$\mathcal{X}$. Furthermore, we will use $\mathsf{Diverse}$-$\mathcal{X}$ as a shorthand if a statement holds for $\mathsf{Diverse}_{\mathsf{sum}}$-$\mathcal{X}$ and $\mathsf{Diverse}_{\mathsf{min}}$-$\mathcal{X}$ both for set and bag semantic. Thus, for example, if we say $\mathsf{Diverse}$-$\mathcal{X}$ is solvable in polynomial time, we mean that all four problems are solvable in polynomial time, and if we say $\mathsf{Diverse}$-$\mathcal{X}$ is NP-hard, we mean that all four problems are NP-hard, independently.

In total, Figure 4.1 summarizes the diversity problems considered in this thesis and there connections. An arrow indicates that the problem at the tip is strictly more general than the problem at the shaft. The connection between $\mathsf{Diverse}$-SAT and $\mathsf{Diverse}$-$\mathrm{CQ}^\neg$ is due to a reduction and discussed in Chapter 7. Furthermore, Figure 4.1 also depicts which acyclicity measures are relevant for which problems. Thus, we usually consider

the diversity problem parameterized by k plus the depicted acyclicity measure (if any is depicted). How to associate an incidence treewidth to a propositional formula not in CNF will also be discussed in Chapter 7.

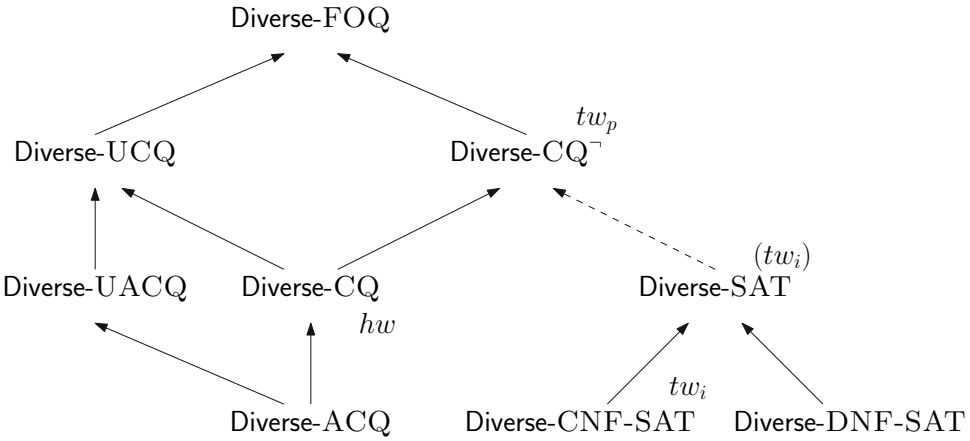


Figure 4.1: Relevant problems and their connections.

In the remainder of this section, we discuss some design decisions. Firstly, we allow solutions to appear multiple times in the list $\gamma_1, \ldots, \gamma_k$, i.e., we consider bag semantics, as our methods consider partial solutions on the way to compute the whole solutions. These partial solutions necessarily need to be allowed to coincide. Therefore, the central ideas can more easily be understood when we also allow the whole solutions to coincide. We will, however, whenever possible explain how to get from bag semantics to set semantics. Set semantics naturally makes more sense from a practical point of view, as distinct solutions are surely preferable for the user.

The Diverse $\mathcal{X}$ Paradigm allows us to consider a large variety of different diversity problems. This, however, is a rather simple way to express what diversity means and much more can be said about the structure of δ (see Ingmar et al. (2020)). Nevertheless, both $\mathsf{Diverse}_{\mathsf{sum}}$-$\mathcal{X}$ and $\mathsf{Diverse}_{\mathsf{min}}$-$\mathcal{X}$ are surly very basic and natural diversity problems. They are in particular often the problems under consideration when diversity is analyzed from a complexity theoretical point of view (Baste et al., 2019, 2022; Hanaka et al., 2021a,b). Furthermore, analyzing $\mathsf{Diverse}_\delta$-$\mathcal{X}$ for arbitrary δ may result in hardness results solely due to the fact that we allow "unnatural" diversity measure.

Dual to diversity problems are so called similarity problems (cf. Hebrard et al., 2005; Eiter et al., 2013). Similarity problems ask for a small subset of *similar* solutions, i.e, which for example have low δ_{sum}. However, although this may not be the intended use, the same semantic can be achieved in the Diverse $\mathcal{X}$ Paradigm by defining the measure of diversity such that high value express low diversity and low values express high diversity (for example we can define $\sigma = -\delta_{\mathsf{sum}}$). It is thus not necessary to introduce the a separate $\mathsf{Similar}_\sigma$-$\mathcal{X}$ problem. In fact, in Chapters 5 to 7 we will also discuss how to solve $\mathsf{Diverse}_\delta$-$\mathcal{X}$ for a class of diversity measures greater than just δ_{sum} and δ_{min}. This class

includes diversity measures which intuitively rather measure the similarity of a collection of solutions than its diversity.

CHAPTER 5

Diversity in Conjunctive Queries

In this chapter, we will tackle the problem Diverse-CQ, providing theoretical analysis and concrete algorithms. We will proceed in the following way: Firstly, we consider the case of acyclic conjunctive queries parameterized by the number of sought-after solutions k. To that end, an algorithm is presented in Section 5.1 that runs in polynomial time when k is bounded. The algorithm is inspired by Yannakakis' algorithm (Yannakakis, 1981) and can be seen as an extension thereof. Then, in Section 5.2, we explain how the result on ACQs can be used to solve the general problem Diverse-CQ and we discuss some modifications that are possible. Lastly, in Section 5.3, the W[1]-hardness of Diverse-ACQ is shown. There, we also look at what happens when either the database or the query is assumed to be fixed. Rather surprisingly, fixing the query turns out to be a weaker restriction than fixing the database for $\mathsf{Diverse}_{\mathsf{sum}}$-ACQ (bag semantics).

5.1 Algorithm for Acyclic Conjunctive Queries

In the following, we design a dynamic programming algorithm that solves Diverse-ACQ by using a join tree as a guide. First, we fix some notation. Let $\mathbf{R}$ be a relation scheme and I a matching database. Furthermore, let $Q\colon ans(X) \leftarrow \exists Y \varphi(X,Y)$ be an ACQ with $\varphi = \bigwedge_{i=1}^{n} A_i$ and each atom $A_i = R_i(Z_i)$ for some predicate $R_i \in \mathbf{R}$ and $Z_i \subseteq X \cup Y$. For an atom $R_i(Z_i)$, define $I(R_i(Z_i))$ as the set of mappings $I(ans(Z_i) \leftarrow R_i(Z_i))$, i.e., where $\alpha(Z_i) \in R^I$. For the sake of succinctness, we just write δ if the statements hold for both δ_{sum} and δ_{min}.

Moreover, we also assume a join tree (T, λ) of Q to be given. For subtrees T' of T we define $\varphi_{T'} = \bigwedge_{t \in V(T')} \lambda(t)$ and $Q_{T'} : ans((X \cup Y) \cap \mathbf{var}(\varphi_{T'})) \leftarrow \varphi_{T'}$. Notice that $Q_{T'}$ has no existentially bound variables and thus, Q_T is not necessarily equal to Q but $I(Q) = \{\gamma|_X : \gamma \in I(Q_T)\}$. Furthermore, we have

$$\max_{\gamma_1,\ldots,\gamma_k \in I(Q_T)} \delta(\gamma_1|_X, \ldots, \gamma_k|_X) = \max_{\gamma_1,\ldots,\gamma_k \in I(Q)} \delta(\gamma_1, \ldots, \gamma_k).$$

Thus, by defining $\delta_X(\gamma_1, \ldots, \gamma_k) = \delta(\gamma_1|_X, \ldots, \gamma_k|_X)$, solving Diverse-ACQ for Q is (almost) the same as solving Diverse$_{\delta_X}$-ACQ for Q_T. Notice that this works for summed and minimal Hamming distance. To that end, we also define $\Delta_X(\gamma_i, \gamma_j) = \Delta(\gamma_i|_X, \gamma_j|_X)$.

For two subtrees T_1 and T_2 of T where the respective roots are connected by an edge, we denote the subtree with the vertices $V(T_1) \cup V(T_2) \subseteq V(T)$ by $T_1 \cup T_2$. By abuse of notation, we denote the one vertex graph $(\{t\}, \emptyset)$ by t.

The algorithm will compute the following relationship for subtrees T' of T with root $t \in V(T')$:

$$D_{T'} \subseteq I(\lambda(t))^k \times [|X|]^{\frac{k(k-1)}{2}}$$

Intuitively, a tuple $(\alpha_1, \ldots, \alpha_k, d_{1,2}, \ldots, d_{k-1,k})$ shall appear in $D_{T'}$ if there exist answers $\gamma_1, \ldots, \gamma_k$ to the query encoded in T' which extend $\alpha_1, \ldots, \alpha_k$ and each pair $\gamma_i, \gamma_j, i < j$ achieves the diversity $d_{i,j}$. Put differently, $d_{1,2}, \ldots, d_{k-1,k}$ is a possible combination of pairwise diversities if answers to the subproblem locally need to look like $\alpha_1, \ldots, \alpha_k$. Formally, $D_{T'}$ is defined as

$$\begin{aligned} D_{T'} = \{(\alpha_1, \ldots, \alpha_k, d_{1,2}, \ldots, d_{k-1,k}) : \alpha_1, \ldots, \alpha_k, \in I(\lambda(t)), \\ \gamma_1, \ldots, \gamma_k \in I(Q_{T'}), \\ \gamma_1 \cong \alpha_1, \ldots, \gamma_k \cong \alpha_k, \\ d_{i,j} = \Delta_X(\gamma_i, \gamma_j), 1 \leq i < j \leq k\}. \end{aligned}$$

We note that $d_{i,j} = \Delta_X(\gamma_i, \gamma_j)$ is clearly never larger than $|X|$.

The algorithm will proceed as follows. First, the sets D_t are computed for all one vertex subtrees $t \in V(T)$ and then, by traversing the tree T bottom-up, the sets $D_{T'}$ are computed for increasingly large subtrees. For this, let t be an inner node with T_t being the subtree rooted in t. Furthermore, let $t_1, \ldots, t_{m_t}$ be its children and assume each $D_{T_{t_i}}$ to already be computed. We compute D_{T_t} by computing the sets

$$\begin{array}{ccccccc} D_t & \rightsquigarrow & D_{t \cup T_{t_1}} & \rightsquigarrow & D_{t \cup T_{t_1} \cup T_{t_2}} & \rightsquigarrow \cdots \rightsquigarrow & D_{t \cup \bigcup_{i=1}^{m_t} T_{t_i}} = D_{T_t} \\ & & \updownarrow & & \updownarrow & & \updownarrow \\ & & D_{T_{t_1}} & & D_{T_{t_2}} & & D_{T_{t_{m_t}}} \end{array} \tag{5.1}$$

in this order. These sets can be computed with the help of the subsequent lemmata. Lastly, the maximal achievable diversity can easily be read off from D_T.

Lemma 1. *Let t be a node in T. Then*

$$D_t = \{(\alpha_1, \ldots, \alpha_k, d_{1,2}, \ldots, d_{k-1,k}) : \alpha_l \in I(\lambda(t)), d_{i,j} = \Delta_X(\alpha_i, \alpha_j)\} \tag{5.2}$$

and D_t can be computed in time $\mathcal{O}(|I(\lambda(t))|^k \cdot k^2 \cdot |\boldsymbol{var}(\lambda(t))|)$.

Proof. Equation 5.2 follows directly, as the only extensions are $\alpha_1, \ldots, \alpha_k$ themselves. Furthermore, the time bound is achieved by iterating through all $(\alpha_1, \ldots, \alpha_k) \in I(\lambda)^k$ and naively computing $\Delta_X(\alpha_i, \alpha_j)$. □

Lemma 2. *Let t_1 and t_2 be two adjacent vertices in T and let t_1 be the parent of t_2. Furthermore, let T_1 and T_2 be two disjoint subtrees of T ($V(T_1) \cap V(T_2) = \emptyset$) with roots t_1 and t_2, respectively. Then, $D_{T_1 \cup T_2} =$*

$$\begin{aligned}\{(\alpha'_1, \ldots, \alpha'_k, d_{1,2}, \ldots, d_{k-1,k}) : \alpha'_l \in I(\lambda(t_1)), \alpha''_l \in I(\lambda(t_2)), \alpha'_l \cong \alpha''_l, l = 1, \ldots, k,\\ (\alpha'_1, \ldots, \alpha'_k, d'_{1,2}, \ldots, d'_{k-1,k}) \in D_{T_1},\\ (\alpha''_1, \ldots, \alpha''_k, d''_{1,2}, \ldots, d''_{k-1,k}) \in D_{T_2},\\ d_{i,j} = d'_{i,j} + d''_{i,j} - \Delta_X(\alpha'_i \cap \alpha''_i, \alpha'_j \cap \alpha''_j), 1 \leq i < j \leq k\}.\end{aligned} \tag{5.3}$$

Additionally, the set $D_{T_1 \cup T_2}$ can be computed in time $\mathcal{O}(|D|^2 \cdot k \cdot (k + |Z|))$ given D_{T_1} and D_{T_2}, where D is the larger of the two sets D_{T_1}, D_{T_2}, and Z is the larger set of variables from $\boldsymbol{var}(\lambda(t_1))$ and $\boldsymbol{var}(\lambda(t_2))$.

Proof. First assume $(\alpha'_1, \ldots, \alpha'_k, d_{1,2}, \ldots, d_{k-1,k})$ to be a tuple in the set on the right-hand side of Equation 5.3 and let $\alpha''_l, d'_{i,j}$ witness this. To that end, let $\gamma'_1, \ldots, \gamma'_k \in I(Q_{T_1})$, $\gamma''_1, \ldots, \gamma''_k \in I(Q_{T_2})$ witness that $(\alpha'_1, \ldots, \alpha'_k, d'_{1,2}, \ldots, d'_{k-1,k}) \in D_{T_1}$ and, analogously, $(\alpha''_1, \ldots, \alpha''_k, d''_{1,2}, \ldots, d''_{k-1,k}) \in D_{T_2}$ We conclude that $\gamma'_1 \cup \gamma''_1, \ldots, \gamma'_k \cup \gamma''_k$ are in $I(Q_{T_1 \cup T_2})$ and we compute

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

Hence, $(\alpha'_1, \ldots, \alpha'_k, d_{1,2}, \ldots, d_{k-1,k}) \in D_{T_1 \cup T_2}$.

For the reverse direction, consider a $(\alpha_1, \ldots, \alpha_k, d_{1,2}, \ldots, d_{k-1,k}) \in D_{T_1 \cup T_2}$ and let $\gamma_1, \ldots, \gamma_k \in I(Q_{T_1 \cup T_2})$ witness this. For $l = 1, \ldots, k$ we define $\gamma'_l = \gamma_l|_{\mathbf{var}(\varphi_{T_1})} \in I(Q_{T_1})$, $\gamma''_l = \gamma_l|_{\mathbf{var}(\varphi_{T_2})} \in I(Q_{T_2})$. Then, by definition of D_{T_1} and D_{T_2},

$$\begin{aligned}(\gamma'_1|_{\mathbf{var}(\lambda(t_1))}, \ldots, \gamma'_k|_{\mathbf{var}(\lambda(t_1))}, \Delta_X(\gamma'_1, \gamma'_2), \ldots, \Delta_X(\gamma'_{k-1}, \gamma'_k)) \in D_{T_1},\\ (\gamma''_1|_{\mathbf{var}(\lambda(t_2))}, \ldots, \gamma''_k|_{\mathbf{var}(\lambda(t_2))}, \Delta_X(\gamma''_1, \gamma''_2), \ldots, \Delta_X(\gamma''_{k-1}, \gamma''_k)) \in D_{T_1},\end{aligned}$$

and we compute

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

Hence, the tuple $(\alpha_1, \ldots, \alpha_k, d_{1,2}, \ldots, d_{k-1,k})$ is also in the set on the right-hand side of Equation 5.3.

Computing $D_{T_1 \cup T_2}$ can be done in the following way: First, project D_{T_2} onto the variables $Z = \mathbf{var}(\lambda(t_1)) \cap \mathbf{var}(\lambda(t_2))$, i.e, compute

$$D'_{T_2} = \{(\alpha''_1|_Z, \ldots, \alpha''_k|_Z, d''_{1,2}, \ldots, d''_{k-1,k}) : (\alpha''_1, \ldots, \alpha''_k, d''_{1,2}, \ldots, d''_{k-1,k}) \in D_{T_2}\}.$$

Then, join D_{T_1} and D'_{T_2} on the variables Z, i.e., compute

$$\begin{aligned} D'_{T_1 \cup T_2} = \{(\alpha'_1, \dots, \alpha'_k, d'_{1,2}, \dots, d'_{k-1,k}, d''_{1,2}, \dots, d''_{k-1,k}) : \\ (\alpha'_1, \dots, \alpha'_k, d'_{1,2}, \dots, d'_{k-1,k}) \in D_{T_1}, \\ (\alpha''_1, \dots, \alpha''_k, d''_{1,2}, \dots, d''_{k-1,k}) \in D'_{T_2}, \\ \alpha'_l|_Z = \alpha''_l, l = 1, \dots, k\}. \end{aligned}$$

Lastly, we just need to compute the values $d_{i,j}$ as described in Equation 5.3, i.e, we compute

$$\begin{aligned} D_{T_1 \cup T_2} = \{(\alpha'_1, \dots, \alpha'_k, d_{1,2}, \dots, d_{k-1,k}) : \\ (\alpha'_1, \dots, \alpha'_k, d'_{1,2}, \dots, d'_{k-1,k}, d''_{1,2}, \dots, d''_{k-1,k}) \in D'_{T_1 \cup T_2}, \\ d_{i,j} = d'_{i,j} + d''_{i,j} - \Delta_X(\alpha'_i|_Z, \alpha'_j|_Z), 1 < i \leq j < k\}. \end{aligned}$$

All of this can naively be done in time $\mathcal{O}\Big(|D_{T_1}| \cdot |D_{T_2}| \cdot k \cdot \big(k + |\mathbf{var}(\lambda(t_1))| + |\mathbf{var}(\lambda(t_2))|\big)\Big)$ by using nested loops. This completes the proof. □

With this we finalize the algorithm and bound its overall runtime.

Theorem 1. *Let $\boldsymbol{R}$ be a relation scheme, I a compatible database, and Q a compatible acyclic conjunctive query with free variables X. Then the problem Diverse-ACQ can be solved in time $\mathcal{O}(|R^I|^{2k} \cdot (|X|+1)^{k(k-1)} \cdot k^2 \cdot |\boldsymbol{var}(A)| \cdot |Q|)$, where R^I is the table from I with the most rows and A is the atom with the highest number of variables. In particular, the problem Diverse-ACQ is in XP when parameterized by the number of sought-after solutions k.*

Proof. Firstly, computing a join tree (T, λ) in the required time bound is no problem (Graham, 1979; Yu and Özsoyoğlu, 1979; Yannakakis, 1981). Then, using Lemmata 1 and 2, we can compute D_T in the required time bound. For this we proceed as described (see Equation 5.1) and apply Lemma 1 once for each node and Lemma 2 once for each edge, i.e., each lemma $\mathcal{O}(|Q|)$ many times. Furthermore, notice that $|D_{T'}| \leq |R^I|^k \cdot (|X|+1)^{\frac{k(k-1)}{2}}$ for each subtree T' of T by definition.

We can compute (gray brackets are for set semantics)

$$\begin{aligned} \max_{\substack{\gamma_1,\dots,\gamma_k \in I(Q) \\ (\gamma_i \neq \gamma_j \text{ for } i \neq j)}} \delta_{\mathsf{sum}}(\gamma_1, \dots, \gamma_k) &= \max_{\substack{\gamma_1,\dots,\gamma_k \in I(Q_T) \\ (\gamma_i|_X \neq \gamma_j|_X \text{ for } i \neq j)}} \delta_{\mathsf{sum}}(\gamma_1|_X, \dots, \gamma_k|_X) \\ &= \max_{\substack{\gamma_1,\dots,\gamma_k \in I(Q_T) \\ (\Delta_X(\gamma_i,\gamma_j) > 0 \text{ for } i \neq j)}} \sum_{1 \leq i < j \leq k} \Delta_X(\gamma_i, \gamma_j) \\ &= \max_{\substack{(\alpha_1,\dots,\alpha_k,d_{1,2},\dots,d_{k-1,k}) \in D_T \\ (d_{i,j} > 0 \text{ for } 1 \leq i < j \leq k)}} \sum_{1 \leq i < j \leq k} d_{i,j}, \end{aligned}$$

$$\max_{\substack{\gamma_1,\dots,\gamma_k \in I(Q) \\ (\gamma_i \neq \gamma_j \text{ for } i \neq j)}} \delta_{\mathsf{min}}(\gamma_1,\dots,\gamma_k) = \max_{\substack{\gamma_1,\dots,\gamma_k \in I(Q_T) \\ (\gamma_i|_X \neq \gamma_j|_X \text{ for } i \neq j)}} \delta_{\mathsf{min}}(\gamma_1|_X,\dots,\gamma_k|_X)$$
$$= \max_{\substack{\gamma_1,\dots,\gamma_k \in I(Q_T) \\ (\Delta_X(\gamma_i,\gamma_j)>0 \text{ for } i \neq j)}} \min_{1 \leq i < j \leq k} \Delta_X(\gamma_i,\gamma_j)$$
$$= \max_{\substack{(\alpha_1,\dots,\alpha_k,d_{1,2},\dots,d_{k-1,k}) \in D_T \\ (d_{i,j}>0 \text{ for } 1 \leq i < j \leq k)}} \min_{1 \leq i < j \leq k} d_{i,j}.$$

Thus, we can read off the maximal possible diversity from D_T for summed and minimal Hamming distance as well as for set and bag semantics in the required time bound. □

Example 4. An execution of the described algorithm on the query Q of Example 3 can be seen in Figure 5.1. For this, the database I in use is also depicted in the figure. For a node t, the set D_{T_t} is computed by considering the children subtrees 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 $(\alpha_1, \alpha_2, d'_{1,2}) \in D_{T'}$, i.e., $d_{1,2} < d'_{1,2}$.

5.2 Lifting the ACQ-Algorithm

In the following, we will consider some slight modification and extensions that allow the ACQ-Algorithm to solve $\mathsf{Diverse}_\delta$-ACQ for more general δ and the more general problem Diverse-CQ. Furthermore, we discuss how to achieve exponential speedups for the case of $\delta = \delta_{\mathsf{sum}}$ and how to reconstruct k witnessing answers $\gamma_1, \dots, \gamma_k$ with maximal diversity.

5.2.1 Reconstructing Witnesses

Reconstructing witnessing diverse answers is possible in the usual way by backtracking top-down. To see this, let t be an inner node and let $t_1, \dots, t_m$ be its children in the order they were considered by the algorithm. Now let $T_{t,t_s} = t \bigcup_{i=1}^{l} T_{t_s}$. The algorithm computes the tuples in $D_{T_{t,t_s}}$ by justifying them with tuples from $D_{T_{t,t_{s-1}}}$ and $D_{T_{t_s}}$. Concretely, for each tuple $(\alpha_1, \dots, \alpha_k, d_{1,2}, \dots, d_{k-1,k})$ in $D_{T_{t,t_s}}$, the algorithm "knows" about a tuple $(\alpha_1, \dots, \alpha_k, d'_{1,2}, \dots, d'_{k-1,k})$ in $D_{T_{t,t_{s-1}}}$ and about a tuple $(\alpha_1^{t_s}, \dots, \alpha_k^{t_s}, d_{1,2}^{t_s}, \dots, d_{k-1,k}^{t_s})$ in $D_{T_{t_s}}$ with $d_{i,j} = d'_{i,j} + d_{i,j}^{t_s} - \Delta_X(\alpha_i \cap \alpha_i^{t_s}, \alpha_j \cap \alpha_j^{t_s}), 1 \leq i < j \leq k$. In total, the algorithm can keep track of the tuples

$$(\alpha_1^{t_1}, \dots, \alpha_k^{t_1}, d_{1,2}^{t_1}, \dots, d_{k-1,k}^{t_1}) \in D_{T_{t_1}}, \dots, (\alpha_1^{t_m}, \dots, \alpha_k^{t_m}, d_{1,2}^{t_m}, \dots, d_{k-1,k}^{t_m}) \in D_{T_{t_m}}$$

that justify each $(\alpha_1^t, \dots, \alpha_k^t, d_{1,2}^t, \dots, d_{k-1,k}^t) \in D_{T_t}$. That means that for we have $\alpha_l^{t_1} \cong \alpha_l^t, \dots, \alpha_l^{t_m} \cong \alpha_l^t$ and

$$d_{i,j}^t = \Delta_X(\alpha_i^t, \alpha_j^t) + \sum_{s=1}^{m} d_{i,j}^{t_s} - \Delta_X(\alpha_i^t \cap \alpha_i^{t_s}, \alpha_j^t \cap \alpha_j^{t_s}).$$

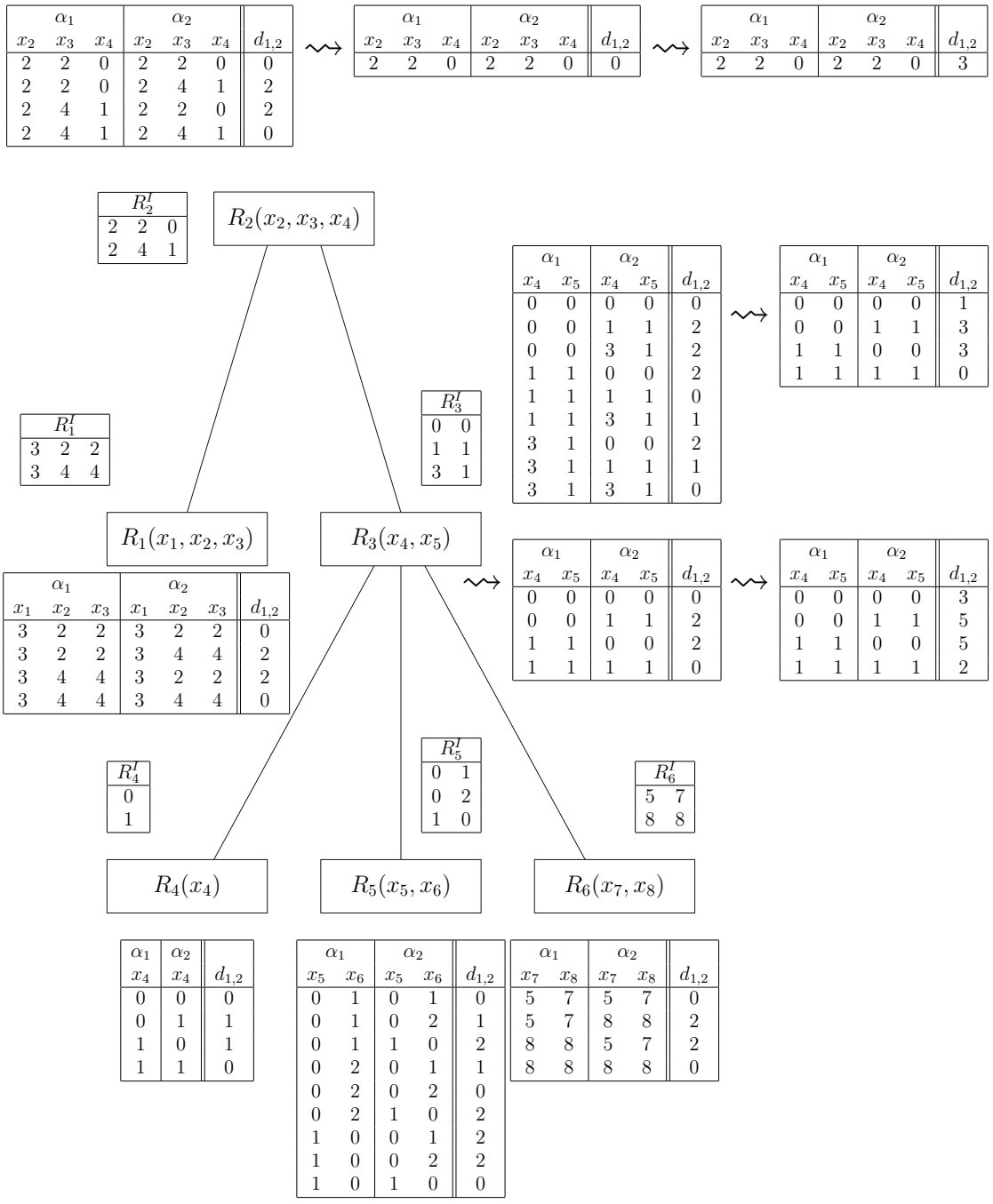


Figure 5.1: Solving Diverse-ACQ with the help of a join tree.

Now, if we start with a tuple $(\alpha_1^r, \ldots, \alpha_k^r, d_{1,2}^r, \ldots, d_{k-1,k}^r) \in D_{T_r}$ in the root and track down all justifications recursively, we get

$$d_{i,j}^r = \Delta_X(\alpha_i^r, \alpha_j^r) + \sum_{t \in V(T)} \Delta_X(\alpha_i^t, \alpha_j^t) - \Delta_X(\alpha_i^{p(t)} \cap \alpha_i^t, \alpha_j^{p(t)} \cap \alpha_j^t),$$

where $p(t)$ is the parent of $t \in V(T) \setminus \{r\}$. Thus, by the properties of a join tree,

$$d_{i,j}^r = \Delta_X(\bigcup_{t \in V(T)} \alpha_i^t, \bigcup_{t \in V(T)} \alpha_j^t).$$

Consequently, we can compute the witnesses $\bigcup_{t \in V(T)} \alpha_1^t, \ldots, \bigcup_{t \in V(T)} \alpha_k^t \in I(Q)$ by additional bookkeeping. This bookkeeping clearly does not increase the runtime of the algorithm.

5.2.2 Exponential Speedup

In the proofs above, the exponential dependency arises due to the possibly exponential size of the data structure $D_{T'}$. Thus, by reducing the size of the data structure, an exponential speedup is possible. This is important for $\mathsf{Diverse}_{\mathsf{sum}}$-ACQ (bag semantics) as it is possible to reduce the size of $D_{T'}$ to $|I(\lambda(t))|^k$. The idea is to only keep track of the maximal achievable overall diversity instead of all pairwise diversities, i.e., the data structure

$$\begin{aligned} D'_{T'} = \{(\alpha_1, \ldots, \alpha_k, d) : (\alpha_1, \ldots, \alpha_k, d_{1,2}, \ldots, d_{k-1,k}) \in D_{T'}, \\ d = \sum_{1 \le i < j \le k} d_{i,j}, \\ d \text{ is maximal for } (\alpha_1, \ldots, \alpha_k)\} \end{aligned}$$

is sufficient. We claim without formal proof that this data structure can also be maintained in the proven time bounds and thus, the problem $\mathsf{Diverse}_{\mathsf{sum}}$-ACQ (bag semantics) can be solved in time $\mathcal{O}(|R^I|^{2k} \cdot k^2 \cdot |\mathbf{var}(A)| \cdot |Q|)$. Most crucial for a formal proof is the fact that it is possible to compute the diversity of a collection of solution by combining the diversities achieved on each individual variable, i.e.,

$$\delta_{\mathsf{sum}}(\gamma_1, \ldots, \gamma_k) = \sum_{x \in X} \delta_{\mathsf{sum}}(\gamma_1|_{\{x\}}, \ldots, \gamma_k|_{\{x\}}). \tag{5.4}$$

Note that Equation 5.4 does not hold it we replace δ_{sum} with δ_{min}.

Similarly, for $\mathsf{Diverse}_{\mathsf{sum}}$-ACQ with set semantics only the summed diversity d and whether partial solutions are different from each other is relevant. Thus, following data structure suffices ($b_{i,j}$ are Boolean values indicating whether the extensions γ_i, γ_j of α_i, α_j need to be distinct from each other on the X variables or are allowed to coincide on

the X variables):

$$D''_{T'} = \{(\alpha_1, \ldots, \alpha_k, b_{1,2}, \ldots, b_{k-1,k}, d) : (\alpha_1, \ldots, \alpha_k, d_{1,2}, \ldots, d_{k-1,k}) \in D_{T'}, \\ d = \sum_{1 \le i < j \le k} d_{i,j}, \\ d \text{ is maximal for } (\alpha_1, \ldots, \alpha_k), \\ b_{i,j} = (d_{i,j} > 0), \text{ for } 1 \le i < j \le k\}.$$

The size of this data structure is never more than $|I(\lambda(t))|^k \cdot 2^{\frac{k(k-1)}{2}}$ and thus, we claim (again without proof) that $\mathsf{Diverse}_{\mathsf{sum}}$-ACQ (set semantics) can be solved in time $\mathcal{O}(|R^I|^{2k} \cdot 2^{k(k-1)} \cdot k^2 \cdot |\mathbf{var}(A)| \cdot |Q|)$.

5.2.3 Generalizing the Diversity Measure

Intuitively, to solve Diverse-ACQ, the presented algorithm computes all possible simultaneously achievable pairwise distances of solutions and then selects the collection with the highest overall diversity. Thus, how the overall diversity is computed is only relevant in the final step and, hence, it is possible to solve $\mathsf{Diverse}_\delta$-ACQ in the time bound of Theorem 1 for any "reasonable" diversity measure δ that stems from combining pairwise Hamming distances.

Corollary 1. *Let $f : \bigcup_{k \ge 1} \mathbb{N}^{\frac{k(k-1)}{2}} \to \mathbb{Q}$ be a function such that $f(d_{1,2}, \ldots, d_{k-1,k})$ is computable in time $\mathcal{O}(k^2)$ and define $\delta(\gamma_1, \ldots, \gamma_k) = f(\Delta(\gamma_1, \gamma_2), \ldots, \Delta(\gamma_{k-1}, \gamma_k))$. Then, Theorem 1 also holds for $\mathsf{Diverse}_\delta$-ACQ (set and bag semantics).*

Proof. We can copy the first part of the proof to Theorem 1 and then compute (gray brackets are for set semantics):

$$\begin{aligned} \max_{\substack{\gamma_1,\ldots,\gamma_k \in I(Q) \\ (\gamma_i \neq \gamma_j \text{ for } i \neq j)}} \delta(\gamma_1, \ldots, \gamma_k) &= \max_{\substack{\gamma_1,\ldots,\gamma_k \in I(Q_T) \\ (\gamma_i|_X \neq \gamma_j|_X \text{ for } i \neq j)}} \delta(\gamma_1|_X, \ldots, \gamma_k|_X) \\ &= \max_{\substack{\gamma_1,\ldots,\gamma_k \in I(Q_T) \\ (\Delta_X(\gamma_i,\gamma_j) > 0 \text{ for } i \neq j)}} f(\Delta_X(\gamma_1, \gamma_2), \ldots, \Delta_X(\gamma_{k-1}, \gamma_k)) \\ &= \max_{\substack{(\alpha_1,\ldots,\alpha_k,d_{1,2},\ldots,d_{k-1,k}) \in D_T \\ (d_{i,j} > 0 \text{ for } 1 \le i < j \le k)}} f(d_{1,2}, \ldots, d_{k-1,k}). \end{aligned}$$

Thus, analogously to before, we can read off the maximal possible diversity from D_T. □

Both δ_{sum} and δ_{min} fall into this category of diversity measures ($f = \sum$ and $f = \min$, respectively). But this also shows that we can solve some similarity problems in the time bound of Theorem 1. For example, following functions work:

$$\sigma_{\mathsf{sum}}(\gamma_1, \ldots, \gamma_k) = -\sum_{1 \le i < j \le k} \Delta(\gamma_i, \gamma_j),$$

$$\sigma_{\mathsf{max}}(\gamma_1, \ldots, \gamma_k) = -\max_{1 \le i < j \le k} \Delta(\gamma_i, \gamma_j).$$

5.2.4 From Diverse-ACQ to Diverse-CQ

We now turn our attention to the more general problem Diverse-CQ. The concepts of hypertree decompositions and hypertreewidth (Gottlob et al., 2002a) were precisely developed to bridge the graph between ACQs and CQs. In the following, we will outline the steps required to apply the developed algorithm from Section 5.1 to arbitrary CQs (cf. Gottlob et al., 2002a; Pichler and Skritek, 2013).

To that end, let I be a database instance, Q be a CQ, and (T, χ, λ) a hypertree decomposition thereof. Furthermore, let X be the free variables, Y the bound variables, and ω the width of (T, χ, λ). Due to Gottlob et al. (2002a), we can assume that each atom A of Q appears in some $\lambda(t), t \in V(T)$ and $\mathbf{var}(A) \subseteq \chi(t)$. With such a hypertree decomposition, we can compute a new query ACQ $\overline{Q}$ and new database $\overline{I}$ such that $I(Q) = \overline{I}(\overline{Q})$ (Gottlob et al., 2002a; Pichler and Skritek, 2013).

The new query is

$$\overline{Q} : ans(X) \leftarrow \exists Y \bigwedge_{t \in V(T)} R_t(\chi(t)),$$

where R_t are fresh relation symbols and the tables $R_t^{\overline{I}}$ are obtained by joining all tables corresponding to the atoms in $\lambda(t)$ and projecting the result onto $\chi(t)$. We have to perform at most ω joins to compute $R_t^{\overline{I}}$ and thus, $\overline{Q}$ and $\overline{I}$ are at most polynomially larger than Q and I, if we assume ω to be bounded by a constant. Lastly, note that all these steps, including computing a width optimal hypertree decomposition, can be done in polynomial time if $hw(Q)$ is assumed to be bounded by a constant (Gottlob et al., 2002a). This argumentation implies the corollary below.

Corollary 2. *The problem Diverse-CQ lies in XP parameterized by the number of sought-after solutions k plus the hypertreewidth of the query $hw(Q)$.*

Proof. This is a direct consequence of the above argumentation combined with Theorem 1. □

Example 5. The cyclic conjunctive query Q' considered in Example 3 together with a database I can be transformed into the acyclic conjunctive query

$$\overline{Q'} : ans(x_1, x_2, x_3, x_4) \leftarrow R_{t_1}(x_1, x_2, x_3) \wedge R_{t_2}(x_1, x_3) \wedge R_{t_3}(x_3) \wedge R_{t_4}(x_3, x_4)$$

with the help of the hypertree decomposition $HD(Q') = (T, \chi, \lambda)$ (also from Example 3). The accompanied tables are as follows: $R_{t_1}^{\overline{I}}$ is the join of R_1^I and R_2^I, $R_{t_2}^{\overline{I}} = R_3^I$, $R_{t_3}^{\overline{I}}$ is the projection of R_2^I onto the second column, and $R_{t_4}^{\overline{I}} = R_4^I$. The query $\overline{Q'}$ is clearly acyclic as (T, χ) is a join tree thereof.

5.3 Data-, Query-, and Combined Complexity

When analyzing database queries it is very important to specify which parts of the input are considered to be fixed. To that end, one usually considers the data-, query-, and combined complexity of database queries (Vardi, 1982):

Data Complexity: The query is assumed to be fixed and the impact of the database on the runtime is analyzed.

Query Complexity: The database is assumed to be fixed and the impact of the query on the runtime is analyzed.

Combined Complexity: Neither the database nor the query is assumed to be fixed and the overall runtime is analyzed.

The practical reason for the distinction is the fact that real world databases can easily have multiple terabyte of data while the queries are usually tiny in comparison. Thus, assuming the query to be fixed is reasonable in practice. Nevertheless, from a theoretical point of view and to more thoroughly understand the problems, all three cases are of interest.

Note that deciding the existence of an answering to general CQs is NP-hard query complexity (thus also for combined complexity), while deciding this for FO queries only requires logarithmic space data complexity (Chandra and Merlin, 1977; Vardi, 1982). However, recall also that answering ACQs is tractable combined complexity (Yannakakis, 1981).

In the following, we will consider the problem Diverse-ACQ more thoroughly from these three perspectives. To that end, Theorem 1 already showed the XP-membership in the combined complexity case. Moreover, the argumentation in Section 5.2.2 shows that $\mathsf{Diverse}_{\mathsf{sum}}$-ACQ (set and bag semantics) is FPT query complexity.

Subsequently, we will show the W[1]-hardness of Diverse-ACQ in the combined complexity case, thus establishing that the existence of an FPT algorithm is unlikely and the described algorithm is in fact optimal. Thereafter, we show that $\mathsf{Diverse}_{\mathsf{sum}}$-ACQ (only bag semantics) can be solved in polynomial time when the database is assumed to be fixed and k is given in unary. Lastly, if we assume the query to be fixed, we show that Diverse-ACQ (all cases) becomes FPT parameterized by k and NP-hard without the parameter. In fact, we show that Diverse-FOQ is already in FPT. Rather surprising, the problem $\mathsf{Diverse}_{\mathsf{sum}}$-ACQ does not follow the usual trend where fixing the query makes the problem easier than when fixing the database. These results for Diverse-ACQ are summarized in Table 5.1.

Complexity	Diversity	Semantics	Lower Bound	Upper Bound
combined	$\delta_{\mathsf{sum}}, \delta_{\mathsf{min}}$	bag/set	W[1]	XP
data	$\delta_{\mathsf{sum}}, \delta_{\mathsf{min}}$	bag/set	NP	FPT
query	δ_{sum}	bag		P
query	δ_{sum}	set		FPT
query	δ_{min}	bag/set		XP

Table 5.1: Results for Diverse-ACQ.

5.3.1 W[1]-Hardness - Combined complexity

We show that Diverse-ACQ is W[1]-hard, parameterized by the number of sought-after solutions k. This will be done by reducing the W[1]-hard Independent-Set problem to Diverse-ACQ, where the size of the sought-after independent set k' is the parameter.

Theorem 2. *The problem Diverse-ACQ is W[1]-hard parameterized by k, the number of sought-after solutions.*

The problem remains W[1]-hard even when restricted to relation symbols of arity at most two and queries Q without bound variables.

Proof. We proceed by first giving the reduction and then proving its correctness.

Reduction. Let (G, k') be an instance of Independent-Set with $V(G) = \{v_1, \ldots, v_n\}$ and $E(G) = \{e_1, \ldots, e_m\}$. We use the relation scheme $\mathbf{R}$ with relation symbols $R, R_1, \ldots, R_m$. The symbol R is of arity one while the symbols R_1 through R_m are of arity two. The query Q is defined as

$$Q\colon ans(v, x_1, \ldots, x_m) \leftarrow R(v) \wedge R_1(v, x_1) \wedge \cdots \wedge R_m(v, x_m)$$

and is clearly acyclic. Furthermore, Q contains no bound variables. For the database I, we define the domain as $\mathbf{dom}(I) = \{0, 1, \ldots, n\}$ and the tables as

$$\begin{aligned} R^I &= \{(i) : v_i \in V(G)\}, \\ R_j^I &= \{(i, i) : v_i \text{ is not incident to } e_j\} \\ &\quad \cup \{(i, 0) : v_i \text{ is incident to } e_j\}, j = 1, \ldots, m. \end{aligned}$$

The number of sought-after solution is $k = k'$ and the target diversity is

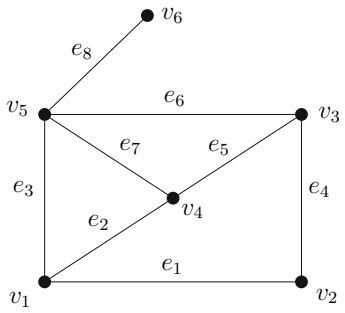
$$d_{\mathsf{sum}} = \frac{k(k-1)}{2}(m+1), \quad d_{\mathsf{min}} = m + 1,$$

respectively for δ_{sum} and δ_{min}. The Diverse$_{\mathsf{sum}}$-ACQ instance is $(\mathbf{R}, I, Q, k, d_{\mathsf{sum}})$ (set and bag semantics) while the Diverse$_{\mathsf{min}}$-ACQ instance is $(\mathbf{R}, I, Q, k, d_{\mathsf{min}})$ (set and bag semantics) and both can clearly be computed in polynomial time.

Correctness. First observe that $I(Q)$ consists of exactly n answers $\gamma_1, \ldots, \gamma_n$, as the value of $\gamma_i(v)$ already fixes the remaining variables. Let the answers be indexed such that $\gamma_i(v) = i$. In all four cases, the target diversity can only be achieved when all k answers differ pairwise on all variables. Hence, we do not need to differentiate between set and bag semantics. An answer γ_i differs from a different answer γ_j on x_l exactly when v_i and v_j are not both incident to e_l. Thus, γ_i and γ_j differ on all variables if and only if v_i and v_j are not adjacent. Therefore, a set $\{v_{i_1}, \ldots, v_{i_k}\}$ of k vertices is an independent set if and only if the tuple $(\gamma_{i_1}, \ldots, \gamma_{i_k})$ has diversity (at least) d_{sum} or d_{min}, in the respective cases. □

Example 6. An example of the reduction used in the proof can be seen in Figure 5.2. The corresponding query is

$$Q\colon ans(v, x_1, \ldots, x_8) \leftarrow R(v) \wedge R_1(v, x_1) \wedge R_2(v, x_2) \wedge R_3(v, x_3) \wedge R_4(v, x_4) \\ \wedge R_5(v, x_5) \wedge R_6(v, x_6) \wedge R_7(v, x_7) \wedge R_8(v, x_8).$$



R^I
1
2
3
4
5
6

R_1^I	
1	0
2	0
3	3
4	4
5	5
6	6

R_2^I	
1	0
2	2
3	3
4	0
5	5
6	6

R_3^I	
1	0
2	2
3	3
4	4
5	0
6	6

R_4^I	
1	1
2	0
3	0
4	4
5	5
6	6

R_5^I	
1	1
2	2
3	0
4	0
5	5
6	6

R_6^I	
1	1
2	2
3	0
4	4
5	0
6	6

R_7^I	
1	1
2	2
3	3
4	0
5	0
6	6

R_8^I	
1	1
2	2
3	3
4	4
5	0
6	0

Figure 5.2: An example illustrating the W[1]-hardness proof (combined complexity).

5.3.2 Polynomial Time Membership - Query Complexity

Note, that the algorithm presented in Section 5.1 computes some information multiple times. The reason being, that, for any permutation π, $(\alpha_1, \ldots, \alpha_k, d_{1,2}, \ldots, d_{k-1,k}) \in D_{T'}$ if and only if $(\alpha_{\pi(1)}, \ldots, \alpha_{\pi(k)}, d_{\pi(1),\pi(2)}, \ldots, d_{\pi(k-1),\pi(k)}) \in D_{T'}$ (assume $d_{i,j} = d_{j,i}$ for $i > j$). This redundancy makes the algorithm conceptually simpler and does not impact the runtime too much when k is small in comparison to the tables $R^I, R \in \mathbf{R}$.

However, when considering query complexity, the tables R^I are assumed to be constant. Hence, assuming them to be large in comparison to k is not justified. In fact, removing the redundancy combined with the argumentation after presented in Section 5.2.2 leads to a polynomial-time algorithm for Diverse-ACQ (bag semantics) query complexity, where k is not considered as a parameter but given in unary.

Theorem 3. *Let I be a database matching a relation scheme $\boldsymbol{R}$. Then the problem Diverse$_{\text{sum}}$-ACQ (bag semantics) is solvable in polynomial time, when the database is fixed to I and the relation scheme to $\boldsymbol{R}$. Furthermore, we assume k, the number of sought-after solutions, to be given in unary.*

Proof (sketch). We alter the algorithm from Section 5.1 by slightly redefining $D_{T'}$ to remove redundant rows. For this, let everything else be defined as before. We furthermore assume there to be an order $\preceq$ on the elements of $I(\lambda(t))$ for each $t \in V(T)$. We now only introduce a row into $D'_{T'}$ for each tuple $(\alpha_1, \dots, \alpha_k, d_{1,2}, \dots, d_{k-1,k}) \in D_{T'}$ where $\alpha_1 \preceq \dots \preceq \alpha_k$. Furthermore, as explained in Section 5.2.2, it suffices for $D'_{T'}$ to not contain $(d_{1,2}, \dots, d_{k-1,k})$ but instead the maximal possible $d = \sum_{1 \leq i < j \leq k} d_{i,j}$. That said, $D'_{T'}$ is defined as

$$\begin{aligned} D'_{T'} = \{(\alpha_1, \dots, \alpha_k, d) : \alpha_l, \in I(\lambda(t)), \gamma_l \in I(Q_{T'}), \gamma_l \cong \alpha_l, l = 1, \dots, k, \\ \alpha_1 \preceq \dots \preceq \alpha_k, \\ d = \sum_{1 \leq i < j \leq k} \Delta_X(\gamma_i, \gamma_j), \\ d \text{ is maximal for } (\alpha_1, \dots, \alpha_k)\}. \end{aligned}$$

Thus, each $D'_{T'}$ consists of

$$\binom{|I(\lambda(t))| + k - 1}{k} = \binom{|I(\lambda(t))| + k - 1}{|I(\lambda(t))| - 1} \leq (k + |I(\lambda(t))| - 1)^{|I(\lambda(t))|-1} = \mathcal{O}(k^{|I(\lambda(t))|-1})$$

rows due to basic combinatorics. Note that this is polynomial as $|I(\lambda(t))|$, i.e., the number of rows in a database table, is considered to be constant and $|k|$ is polynomial in k, where $|k|$ denotes the size of the representation of k.

Now, it is clear that the exponential dependency in Lemmata 1 and 2 arises from the exponential sizes of $D_{T'}$. As this is now no longer the case, both can be restated with polynomial runtimes and thus also Theorem 1. This completes the proof. □

Theorem 3 requires the assumption that k is given in unary as k appears in the runtime and integers can be exponentially larger than their representation. This assumption is insofar justified, as, otherwise, there would be no time for the algorithm to even consider k different solutions. Hence, the algorithm would have to reason about solutions without being allowed to materialize them.

Concluding, we note that we find it wise to perform these redundancy elimination in any implementation of the algorithm, even when the database cannot be assumed to be small and k to be large. As already noted, the redundancy is added for the sole reason on simplicity.

5.3.3 FPT-Membership - Data Complexity

For the analysis of the data complexity, it is actually not necessary to restrict the form of the database query. For any fixed first order database query, it is possible to evaluate the query in polynomial time (Vardi, 1982). Doing this results in a table of answer tuples. Throughout this section, we may therefore assume w.l.o.g. that the query is of the form $Q\colon ans(x_1,\dots,x_m) \leftarrow R(x_1,\dots,x_m)$ and the database I consists of a single relation R^I. In other words, R^I is the set of answer tuples of the original query over the original database.

Our goal is to prove FPT-membership of the diversity problem. To this end, we apply a kernelization that allows us to iteratively reduce the size of the database until it is bounded by a function of m and k, i.e., the query and the parameter. Let $X = \{x_1,\dots,x_m\}$. Moreover, for each assignment $\alpha\colon Z \to \mathbf{dom}(I)$ with $Z \subseteq X$ let $I(Q)_\alpha = \{\gamma \in I(Q)\colon \gamma \cong \alpha\}$, i.e., the set of answer tuples that coincide with α on Z. The key to our kernelization is the following reduction rule $\mathbf{Red}_t$ for $t \in \{1,\dots,m\}$:

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

The following lemma states the crucial properties of this reduction rule for its layerwise (i.e., for increasing t) exhaustive application:

Lemma 3. *Let $t \in \{1,\dots,m\}$ and suppose that all sets $I(Q)_{\alpha'}$ with $\alpha'\colon Z' \to \mathbf{dom}(I)$ 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 $\alpha : Z \to \mathbf{dom}(I)$ with $Z \in \binom{X}{m-t}$, the set $I(Q)_\alpha$ has more than $t!^2 \cdot k^t$ elements, then there indeed exist $t \cdot k$ solutions $\Gamma \subseteq I(Q)_\alpha$ 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 $\gamma_1,\dots,\gamma_k$ be solutions in $Q(I_{old})$. Then there exist solutions $\gamma'_1,\dots,\gamma'_k$ in $Q(I_{new})$ with $\delta_{\mathsf{sum}}(\gamma'_1,\dots,\gamma'_k) \geq \delta_{\mathsf{sum}}(\gamma_1,\dots,\gamma_k)$ and $\delta_{\mathsf{min}}(\gamma'_1,\dots,\gamma'_k) \geq \delta_{\mathsf{min}}(\gamma_1,\dots,\gamma_k)$, i.e., the diversity achievable before deleting tuples from the database can still be achieved after the deletion. Furthermore, if $\gamma_1,\dots,\gamma_k$ are pairwise distinct, it is also possible to pick $\gamma'_1,\dots,\gamma'_k$ pairwise distinct.

Proof. Let $t \in \{1, \ldots, m\}$ and suppose that all sets $I(Q)_{\alpha'}$ with $\alpha' \colon Z' \to \mathbf{dom}(I)$ and $Z' \in \binom{X}{m-(t-1)}$ have cardinality at most $(t-1)!^2 \cdot k^{t-1}$. Furthermore, the subsequent argumentation is equally valid for δ_{sum} and δ_{min}, hence, we will simply use δ.

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

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

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

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

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 $\gamma_i \in I(Q)_\alpha \setminus \bigcup_{j=1}^{i-1} C_{\gamma_j}$, i.e., arbitrarily choose $\gamma_1 \in I(Q)_\alpha$, then $\gamma_2 \in I(Q)_\alpha \setminus C_{\gamma_1}$, then $\gamma_3 \in I(Q)_\alpha \setminus (C_{\gamma_1} \cup C_{\gamma_2})$, etc.

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

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

Now set $\Gamma = \{\gamma_1, \ldots, \gamma_{t \cdot k}\} \subseteq I(Q)_\alpha$. 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.

"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 solutions $\gamma_1, \ldots, \gamma_k \in Q(I_{old})$. We have to show that there exist solutions $\gamma'_1, \ldots, \gamma'_k$ solutions in $Q(I_{new})$ with $\delta(\gamma'_1, \ldots, \gamma'_k) \geq \delta(\gamma_1, \ldots, \gamma_k)$ and which are distinct to each other if $\gamma_1, \ldots, \gamma_k$ are distinct to each other.

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 a solution γ'_i that was not removed, i.e., $\gamma'_i \in \Gamma \subseteq Q(I_{new})$, with the property $\delta(\gamma_1, \ldots, \gamma_{i-1}, \gamma'_i, \gamma_{i+1}, \ldots, \gamma_k) \geq \delta(\gamma_1, \ldots, \gamma_k)$ and γ'_i is distinct to $\gamma_1, \ldots, \gamma_{i-1}, \gamma_{i+1}, \ldots, \gamma_k$ if γ_i was distinct to them.

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 distinct to γ_j if γ_i was distinct to γ_j. By iterating this argument for every $i \in \{1, \ldots, k\}$, we may conclude that there exist solutions $\gamma_1', \ldots, \gamma_k' \in I_{new}$ with $\delta(\gamma_1', \ldots, \gamma_k') \geq \delta(\gamma_1, \ldots, \gamma_k)$ and $\gamma_1', \ldots, \gamma_k'$ are pairwise distinct if $\gamma_1, \ldots, \gamma_k$ are.

It only remains to show that $|\Gamma_j| \leq t$ really 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. □

With this lemma, we are now ready to prove that Diverse-ACQ and, more generally, Diverse-FOQ is in FPT data complexity.

Theorem 4. *The problem Diverse-FOQ parameterized by the number of sought-after solutions k is in FPT data complexity.*

Proof. For data complexity, the query is considered as fixed. Hence, we can evaluate the FOQ in polynomial time and we may restrict our attention to the case that the query is of the form $Q\colon ans(x_1, \ldots, x_m) \leftarrow R(x_1, \ldots, x_m)$ 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, for $Z \in \binom{X}{m}$, we have $Z = X$ and hence, for every $\alpha : Z \to \mathbf{dom}(I) \in I(Q)$, we have $I(Q)_\alpha = \{\alpha\}$. In particular, $|I(Q)_\alpha| = 1 \leq 0!^2 \cdot k^0$. Hence, if $\mathbf{Red}_1$ is applicable, then the preconditions of Lemma 3 are fulfilled and exhaustive application of $\mathbf{Red}_1$ does not alter the status of the Diverse-FOQ problem. After exhaustive application of $\mathbf{Red}_1$, if now $\mathbf{Red}_2$ is applicable, then the preconditions of Lemma 3 are fulfilled and exhaustive application of $\mathbf{Red}_2$ does not alter the status of the Diverse-FOQ 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 $\alpha : \emptyset \to \mathbf{dom}(I^*)$. In particular, this means that such an assignment 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 that also $|I^*| \leq m!^2 \cdot k^m$ holds. Diverse solutions can then be computed in FPT time by brute force.

It remains to show that the exhaustive application of $\mathbf{Red}_1$ through $\mathbf{Red}_m$ is in FPT. Note that the number of subsets $Z \in \binom{X}{m-t}$ is bounded by 2^m, where m is considered to be constant in case of data complexity. For every such set of variables $Z \subseteq X$, we iterate through all solutions $\gamma \in I(Q)$ for the current database instance I and group by $\alpha = \gamma|_Z$. We then count the size of $I(Q)_\alpha$ for each α and, if $\mathbf{Red}_t$ with $t = |Z|$ is applicable, we greedily compute Γ by computing $t \cdot k - 1$ sets C_{γ_j} as defined in the proof of Lemma 3. We then update the database such that $Q(I_{new}) = (I(Q) \setminus I(Q)_\alpha) \cup \Gamma$, where I_{new} is the new database. All this can clearly be done in FPT time. □

5.3.4 NP-Hardness - Data Complexity

We now study the data complexity of the Diverse-ACQ problem in the "unparameterized" case, i.e., the size k of the sought-after set of solutions is part of the input and no longer considered as parameter. It will turn out that this problem is NP-hard and, actually, NP-complete, provided that k is given in unary. 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 by Alimonti and Kann (1997) that this restricted problem remains NP-complete. The idea of Alimonti and Kann (1997) 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.

Theorem 5. *The problem Diverse-ACQ is NP-hard data complexity (considered without a parameter). It is NP-complete provided that the size of the sought-after set of solutions k is given in unary.*

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

For the NP-hardness, we define query Q independently of the instance of Independent-Set as

$$Q\colon ans(x_1, x_2, x_3, x_4, x_5) \leftarrow R(x_1, x_2, x_3, x_4, x_5).$$

Now let (G, k') be an instance of Independent-Set where each vertex of G has degree at most 3. Furthermore, let $V(G) = \{v_1, \ldots, v_n\}$ and $E(G) = \{e_1, \ldots, e_m\}$.

The database I consists of a single relation R^I with n tuples (= number of vertices in G) over the domain $\mathbf{dom}(I) = \{\mathbf{free}_1, \ldots, \mathbf{free}_n, \mathbf{taken}_1, \ldots, \mathbf{taken}_m\}$. 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 $e_{i,1}, \ldots, e_{i,5} \in \mathbf{dom}(I)$ 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 $\mathbf{taken}_j$.

Note that, 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 $\mathbf{free}_i$. By the analogous consideration for vertex $v_{i'}$ we conclude that, for at least three different values of $t \in \{1, \ldots, 5\}$, the value $e_{i',t}$ is still set to $\mathbf{free}_{i'}$. Hence, by the pigeon hole principle, there exists $t \in \{1, \ldots, 5\}$ such that $e_{i,t}$ and $e_{i',t}$ both still have the values $\mathbf{free}_i$ and $\mathbf{free}_{i'}$, respectively.

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 number of sought-after solutions is set to $k = k'$ and the target diversity is set to $d_{\mathsf{sum}} = 5 \cdot \frac{k(k-1)}{2}$ and $d_{\mathsf{min}} = 5$ in case of the $\mathsf{Diverse}_{\mathsf{sum}}$-ACQ and $\mathsf{Diverse}_{\mathsf{min}}$-ACQ problems, respectively. The resulting instances for $\mathsf{Diverse}_{\mathsf{sum}}$-ACQ and $\mathsf{Diverse}_{\mathsf{min}}$-ACQ are thus of the form $(I, Q, k, d_{\mathsf{sum}})$ and $(I, Q, k, d_{\mathsf{min}})$, respectively. Both can clearly be computed in polynomial time.

It remains to show the correctness of the reduction, i.e., the graph $G = (V(G), E(G))$ has an independent set of size k' if and only if there exists $S \subseteq I(Q)$ with $|S| = k$ and diversity $\geq d_{\mathsf{sum}}$ respectively $\geq d_{\mathsf{min}}$.

The answers $I(Q)$ are trivially $\{\gamma_1, \ldots, \gamma_n\}$ with $\gamma_i(x_t) = e_{i,t}$ for each $t \in \{1, \ldots, 5\}$. Furthermore, for both $\mathsf{Diverse}_{\mathsf{sum}}$-ACQ and $\mathsf{Diverse}_{\mathsf{min}}$-ACQ, the target diversity can only be achieved by k answers that pairwisely differ on all 5 variables $x_1, \ldots, x_5$. Hence, we do not need to distinguish between set and bag semantics.

Now suppose that graph G has an independent set of size k, say $\{v_{i_1}, \ldots, v_{i_k}\}$. We claim that then $\{\gamma_{i_1}, \ldots, \gamma_{i_k}\}$ is a subset of $I(Q)$ 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

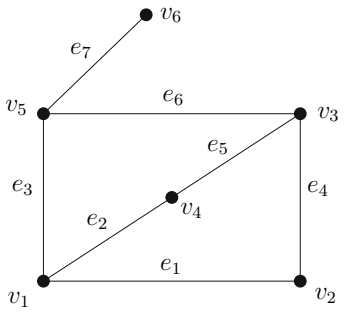
for some $t \in \{1, \ldots, 5\}$. By our construction of R^I, this can only happen if $\gamma_{i_r}(t) \neq \mathbf{free}_{i_r}$ and $\gamma_{i_s}(t) \neq \mathbf{free}_{i_s}$. Hence, $\gamma_{i_r}(t) = \gamma_{i_s}(t) = \mathbf{taken}_j$ for some $j \in \{1, \ldots, m\}$ holds. Again by our construction 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 $S \subseteq I(Q)$ of size k with the desired target diversity. Let $S = \{\gamma_{i_1}, \ldots, \gamma_{i_k}\}$. We claim that then $\{v_{i_1}, \ldots, v_{i_k}\}$ 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 $t \in \{1, \ldots, 5\}$ with $\gamma_{i_r}(t) = \gamma_{i_s}(t) = \mathbf{taken}_j$. This means that the target diversities d_{sum} and d_{min} cannot be reached by S, which is a contradiction. □

Example 7. An example of the reduction used in the proof above can be seen in Figure 5.3. Recall, the query is

$$Q\colon ans(x_1, x_2, x_3, x_4, x_5) \leftarrow R(x_1, x_2, x_3, x_4, x_5).$$

s



R^I				
$\mathbf{taken}_1$	$\mathbf{taken}_2$	$\mathbf{taken}_3$	$\mathbf{free}_1$	$\mathbf{free}_1$
$\mathbf{taken}_1$	$\mathbf{taken}_4$	$\mathbf{free}_2$	$\mathbf{free}_2$	$\mathbf{free}_2$
$\mathbf{taken}_5$	$\mathbf{taken}_4$	$\mathbf{free}_3$	$\mathbf{taken}_6$	$\mathbf{free}_3$
$\mathbf{taken}_5$	$\mathbf{taken}_2$	$\mathbf{free}_4$	$\mathbf{free}_4$	$\mathbf{free}_4$
$\mathbf{taken}_7$	$\mathbf{free}_5$	$\mathbf{taken}_3$	$\mathbf{taken}_6$	$\mathbf{free}_5$
$\mathbf{taken}_7$	$\mathbf{free}_6$	$\mathbf{free}_6$	$\mathbf{free}_6$	$\mathbf{free}_6$

Figure 5.3: An example illustrating the NP-hardness proof (data complexity).

CHAPTER 6

Introducing Unions and Negations

In this chapter, we allow slightly more general queries than in the previous chapter. To that end, we introduce unions in Section 6.1 and negated atoms in Sections 6.2 and 6.3.

6.1 The Case of Unions of Acyclic Conjunctive Queries

We turn our attention to Unions of Conjunctive Queries (UCQs) and, in particular, to Unions of Acyclic Conjunctive Queries (UACQs). Of course, all hardness results proved for CQs and ACQs carry over to UCQs and UACQs, respectively. Furthermore, the FPT-membership data complexity is already proven in Theorem 5.

It remains to study the query complexity and combined complexity of UCQs and, more interestingly, UACQs. It turns out that in this case, the union makes the problem significantly harder than for ACQs. We show next that Diverse-UACQ is NP-hard even in a very restricted setting, namely a union of two ACQs and with the number of sought-after solutions $k = 2$. 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 $C(v) \subseteq \{1, \ldots, l\}$, where $l \in \mathbb{N}$. 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 $uv \in E(G)$ 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 a C-admissible coloring $c : V(G) \to \{1, \ldots, l\}$ exist?

Clearly, List Coloring is a generalization of 3-Colorability and, hence, NP-complete. It was shown by Chlebík and Chlebíková (2006), 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 6. *The problem Diverse-UACQ is NP-hard query complexity (and, hence, also combined complexity). The problem remains NP-hard even if the number of sought-after solutions is bounded by 2 and the UACQ is restricted to at most two conjuncts and contains no existential variables. The problem is NP-complete if the number of sought-after solutions k is given in unary.*

Proof. The NP-membership in the case of k being 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 $\mathsf{Diverse}_{\mathsf{sum}}$-UACQ and $\mathsf{Diverse}_{\mathsf{min}}$-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 (G, l, C) 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 three 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 $Q\colon ans(x_1, \ldots, x_{3n}, y) \leftarrow \varphi \vee \psi$ 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 formulae 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).

The reduction is clearly feasible in polynomial time. For the correctness of the reduction, observe that diversity $d = 3n + 1$ can only be achieved by two answers γ, γ' that differ on all variables. Thus, we do not need to differentiate between set and bag semantics. 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 $(Q, I, 2, 3n + 1)$ is a positive instance of Diverse-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 $i \in \{1, \ldots, n\}$ 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 $i \in \{1, \ldots, n\}$.

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 $i \in \{1, \ldots, n\}$. 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 $r, s \in \{1, \ldots, n\}$ 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 $(Q, I, 2, 3n+1)$ is a positive instance of the problem Diverse-UACQ, i.e., there exist two solutions γ and γ' with diversity $3n+1$. 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 $i \in \{1, \ldots, n\}$.

We have to show that c is C-admissible. Consider an arbitrary edge e with endpoints v_r and v_s for $r, s \in \{1, \ldots, n\}$. 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 $3n+1$, 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. □

Example 8. Consider the graph depicted in Figure 6.1. The above reduction would construct the query

$$Q\colon ans(x_1, \ldots, x_{12}, y) \leftarrow \varphi \vee \psi$$

with

$$\varphi = R_0(y) \wedge R_1(x_1, x_2, x_3) \wedge R_2(x_4, x_5, x_6) \ \wedge R_3(x_7, x_8, x_9) \ \wedge R_4(x_{10}, x_{11}, x_{12}),$$
$$\psi = R'_0(y) \wedge R'_1(x_1, x_4, x_7) \wedge R'_2(x_2, x_5, x_{10}) \wedge R'_3(x_3, x_8, x_{11}) \wedge R'_4(x_6, x_9, x_{12}).$$

The results for Diverse-UACQ are summarized in Table 6.1.

Complexity	Diversity	Semantics	Lower Bound	Upper Bound
combined	$\delta_{\mathsf{sum}}, \delta_{\mathsf{min}}$	bag/set	NP for $k \leq 2$	NP
data	$\delta_{\mathsf{sum}}, \delta_{\mathsf{min}}$	bag/set	NP	FPT
query	$\delta_{\mathsf{sum}}, \delta_{\mathsf{min}}$	bag/set	NP for $k \leq 2$	NP

Table 6.1: Results for Diverse-UACQ.

6.2 Primal Treewidth Algorithm

In the remainder of this chapter, we will tackle the problem Diverse-CQ$^\neg$, i.e., we consider conjunctive queries with possibly negated atoms. As a first step, we describe a dynamic

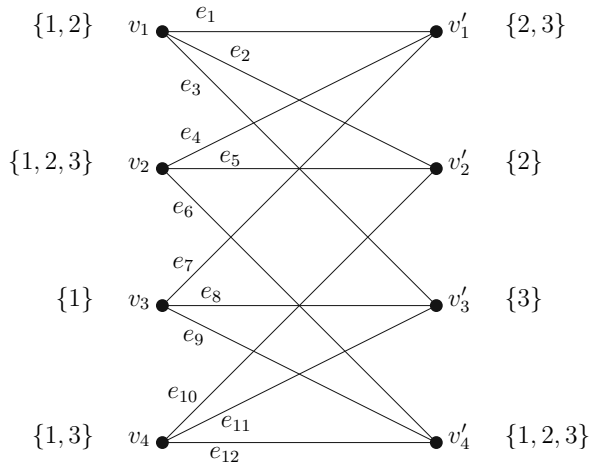


Figure 6.1: An example graph to illustrate the reduction (Diverse-UACQ).

programming algorithm that uses a tree decomposition of the primal graph as a guide to solve Diverse-CQ$^\neg$. This will be an XP algorithm combined complexity, where the number of sought-after solutions and the primal treewidth of the query are taken as parameters. Recall that CQ$^\neg$ remains NP-hard on queries with acyclic hypergraph (Samer and Szeider, 2010) and thus, an XP algorithm is unlikely to exist for the weaker parameter hypertreewidth.

To describe the algorithm, let $Q : ans(X) \leftarrow \exists Y \varphi(X, Y)$ be a CQ$^\neg$, I a database, and (T, χ) a nice tree decomposition of $H(Q)$ with root r. Furthermore, let $\varphi = \bigwedge_{i=1}^{n} L_i$ for literals L_i. We define for subtrees T' of T the formula

$$\varphi_{T'} = \bigwedge_{\substack{i=1,\dots,n \\ \mathbf{var}(L_i) \subseteq \chi(T')}} L_i$$

and query $Q_{T'} : ans(\chi(T')) \leftarrow \varphi_{T'}$. Although some variables of $\chi(T')$ may no longer be part of $\varphi_{T'}$, we assume that $\mathbf{var}(\varphi_{T'}) = \chi(T')$ always holds to spare ourselves from tedious but simple technical details. As before and by abuse of notation, we will denote the one vertex subtree $(\{t\}, \emptyset)$ of T just by t.

Notice that $Q_T = Q_{T_r}$ is the same as Q but where all variables are assumed to be unbound and thus, $I(Q) = \{\gamma|_X : \gamma \in I(Q_T)\}$. Furthermore, as before, for $\delta = \delta_{\mathsf{sum}}$ or $\delta = \delta_{\mathsf{min}}$ we have

$$\max_{\gamma_1,\dots,\gamma_k \in I(Q_T)} \delta(\gamma_1|_X, \dots, \gamma_k|_X) = \max_{\gamma_1,\dots,\gamma_k \in I(Q)} \delta(\gamma_1, \dots, \gamma_k).$$

Thus, by defining $\delta_X(\gamma_1, \dots, \gamma_k) = \delta(\gamma_1|_X, \dots, \gamma_k|_X)$, solving Diverse-CQ$^\neg$ for Q is (almost) the same as solving Diverse$_{\delta_X}$-CQ$^\neg$ for Q_T. Notice that this works for summed and minimal Hamming distance. Further, let us define $\Delta_X(\gamma_i, \gamma_j) = \Delta(\gamma_i|_X, \gamma_j|_X)$.

The dynamic programming algorithm will compute the set

$$D_t \subseteq (\chi(t) \to \mathbf{dom}(I))^k \times [|X|]^{\frac{k(k-1)}{2}}$$

bottom-up for each $t \in V(T)$. The intended meaning of D_t is such that a tuple $(\alpha_1, \ldots, \alpha_k, d_{1,2}, \ldots, d_{k-1,k})$ is in D_t if it is possible to extend the mappings $\alpha_1, \ldots, \alpha_k$ to mappings $\gamma_1, \ldots, \gamma_k$ which all satisfy the query corresponding to the whole subtree rooted in t and, simultaneously, each pair $\gamma_i, \gamma_j, i < j$ achieves the diversity $d_{i,j}$. Put differently, D_t keeps track of all simultaneously achievable pairwise diversities for solutions to a subproblem per local variable assignment. Formally, we define D_t as

$$\begin{aligned} D_t = \{(\alpha_1, \ldots, \alpha_k, d_{1,2}, \ldots, d_{k-1,k}) : \alpha_1, \ldots, \alpha_k \colon \chi(t) \to \mathbf{dom}(I), \\ \gamma_1, \ldots, \gamma_k \in I(Q_{T_t}), \\ \gamma_1 \cong \alpha_1, \ldots, \gamma_k \cong \alpha_k, \\ d_{i,j} = \Delta_X(\gamma_i, \gamma_j), 1 \leq i < j \leq k\}. \end{aligned}$$

The subsequent lemmata show how to compute D_t depending on the type of node t.

Lemma 4. *Let t be a leaf node of T. Then*

$$\begin{aligned} D_t = \{(\alpha_1, \ldots, \alpha_k, d_{1,2}, \ldots, d_{k-1,k}) : \alpha_1, \ldots, \alpha_k \colon \chi(t) \to \boldsymbol{dom}(I), \\ \alpha_1, \ldots, \alpha_k \in I(Q_t), \\ d_{i,j} = \Delta_X(\alpha_i, \alpha_j), 1 \leq i < j \leq k\}. \end{aligned} \tag{6.1}$$

Furthermore, the set D_t can be computed in time $\mathcal{O}(\boldsymbol{dom}(I)^{k \cdot |\chi(t)|}(|Q| \cdot |R^I| + k^2 \cdot \chi(t)))$ where $|R^I|$ is the size of the largest database table.

Proof. Observe that $\chi(t) = \chi(T_t)$ and $\alpha_1, \ldots, \alpha_k$ are the only extensions of $\alpha_1, \ldots, \alpha_k$. Hence, Equation 6.1 holds.

The set D_t can be computed by considering all $\mathbf{dom}(I)^{k \cdot |\chi(t)|}$ possibilities of $\alpha_1, \ldots, \alpha_k$ and using the Equation 6.1. Checking whether an α_i is an answer to Q_t can be done in time $\mathcal{O}(|Q| \cdot |R^I|)$ by naively checking whether α_i satisfies all literals whose variables are covered by $\chi(t)$, i.e., checking whether each instantiated atom appears (respectively does not appear) in the database. The overall runtime follows, as computing each $\Delta_X(\alpha_i, \alpha_j)$ only takes time $\mathcal{O}(k^2|\chi(t)|)$. □

Lemma 5. *Let p be an introduce node of T which introduces the variable z and let t be its child. Then*

$$\begin{aligned} D_p = \{(\alpha_1 \cup \beta_1, \ldots, \alpha_k \cup \beta_k, d'_{1,2}, \ldots, d'_{k-1,k}) : (\alpha_1, \ldots, \alpha_k, d_{1,2}, \ldots, d_{k-1,k}) \in D_t \\ \beta_1, \ldots, \beta_k \colon \{z\} \to \boldsymbol{dom}(I), \\ \alpha_1 \cup \beta_1, \ldots, \alpha_k \cup \beta_k \in I(Q_p) \\ d'_{i,j} = d_{i,j} + \Delta_X(\beta_i, \beta_j), 1 \leq i < j \leq k\}. \end{aligned} \tag{6.2}$$

Furthermore, the set D_p can be computed in time $\mathcal{O}(|D_t| \cdot \boldsymbol{dom}(I)^k \cdot (|Q| \cdot |R^I| + k^2))$ given the set D_t, where $|R^I|$ is the size of the largest database table.

Proof. We start by proving Equation 6.2. First notice that $\chi(T_t) \cup \{z\} = \chi(T_p)$ and thus, $\varphi_{T_t} \subseteq \varphi_{T_p}$. Furthermore, any variable of $\chi(T_t)$ connected to z in $G_p(\varphi)$, i.e., in a literal with z, has to appear in $\chi(p)$ due to the properties of a tree decomposition. We can therefore also conclude that $\varphi_{T_p} = \varphi_{T_t} \cup \varphi_p$.

Now, let $\alpha_l, \beta_l, \gamma_l, d_{i,j}, d'_{i,j}$ be such that the tuple $(\alpha_1 \cup \beta_1, \ldots, \alpha_k \cup \beta_k, d'_{1,2}, \ldots, d'_{k-1,k})$ is in the set on the right-hand side of Equation 6.2, and let γ_l be witnesses the fact that $(\alpha_1, \ldots, \alpha_k, d_{1,2}, \ldots, d_{k-1,k}) \in D_t$. Importantly, $\gamma_l \in I(Q_{T_t}), \alpha_l \cup \beta_l \in I(Q_p)$ and thus, $\gamma_l \cup \beta_l \in I(Q_{T_p})$. Furthermore, for $1 \leq i < j \leq k$, 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, $(\alpha_1 \cup \beta_1, \ldots, \alpha_k \cup \beta_k, d'_{1,2}, \ldots, d'_{k-1,k})$ is in D_p by definition.

For the reverse direction, let $(\alpha_1, \ldots, \alpha_k, d'_{1,2}, \ldots, d'_{k-1,k}) \in D_p$ and let $\gamma_1, \ldots, \gamma_k \in I(Q_{T_p})$ witness this. Thus, we can immediately conclude that $\gamma_1|_{\chi(p)}, \ldots, \gamma_k|_{\chi(p)} \in I(Q_p)$ while $\gamma_1|_{\chi(T_t)}, \ldots, \gamma_k|_{\chi(T_t)} \in I(Q_{T_t})$. Furthermore, for $1 \leq i < j \leq k$, 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, $(\gamma_1|_{\chi(t)}, \ldots, \gamma_k|_{\chi(t)}, d'_{1,2} - \Delta_X(\gamma_1|_{\{z\}}, \gamma_2|_{\{z\}}), \ldots, d'_{i,j} - \Delta_X(\gamma_{k-1}|_{\{z\}}, \gamma_k|_{\{z\}})) \in D_t$ and defining $\beta_l = \gamma_l|_{\{z\}}$ ensures that $(\alpha_1, \ldots, \alpha_k, d'_{1,2}, \ldots, d'_{k-1,k})$ is in the set on the right-hand side of Equation 6.2. The equation is consequently valid.

The set D_p can be computed by considering all elements $(\alpha_1, \ldots, \alpha_k, d_{1,2}, \ldots, d_{k-1,k}) \in D_t$ and using Equation 6.2. Per element we iterate through all $\mathbf{dom}(I)^k$ possibilities for $\beta_1, \ldots, \beta_k$ and check $\alpha_1 \cup \beta_1, \ldots, \alpha_k \cup \beta_k \in I(Q_p)$ as before in time $\mathcal{O}(|Q| \cdot |R^I|)$. Lastly, we have to compute $d'_{i,j}$ which takes constant time per (i, j) pair. Thus, everything put together, D_p can be computed in the required time bound. □

Lemma 6. *Let p be a forget node of T and t its child. Then*

$$D_p = \{(\alpha_1|_{\chi(p)}, \ldots, \alpha_k|_{\chi(p)}, d_{1,2}, \ldots, d_{k-1,k}) : (\alpha_1, \ldots, \alpha_k, d_{1,2}, \ldots, d_{k-1,k}) \in D_t\}. \quad (6.3)$$

Furthermore, the set D_p can be computed in time $\mathcal{O}(|D_t| \cdot (k \cdot \chi(t) + k^2))$ given the set D_t.

Proof. The equation directly follows from the fact that $Q_{T_p} = Q_{T_t}$ and $\chi(p) \subseteq \chi(t)$. Also, Equation 6.3 immediately tells us how to compute D_p in time $\mathcal{O}(|D_t| \cdot (k \cdot \chi(t) + k^2))$ from D_t. □

Lemma 7. *Let p be a join node of T with children t and t'. Then*

$$D_p = \{(\alpha_1, \ldots, \alpha_k, d''_{1,2}, \ldots, d''_{k-1,k}) : (\alpha_1, \ldots, \alpha_k, d_{1,2}, \ldots, d_{k-1,k}) \in D_t \\ (\alpha_1, \ldots, \alpha_k, d'_{1,2}, \ldots, d'_{k-1,k}) \in D_{t'} \\ d''_{i,j} = d_{i,j} + d'_{i,j} - \Delta_X(\alpha_i, \alpha_j), 1 \leq i < j \leq k\}. \tag{6.4}$$

Furthermore, the set D_p can be computed in time $\mathcal{O}(|D_t| \cdot |D_{t'}| \cdot k^2 \cdot |\chi(p)|)$ given the sets D_t and $D_{t'}$.

Proof. We start by proving Equation 6.4. First notice that $\chi(T_t) \cup \chi(T_{t'}) = \chi(T_p)$ and thus, $\varphi_{T_t} \cup \varphi_{T_{t'}} \subseteq \varphi_{T_p}$. But even more, if two variables appear in the same literal, they have to appear together in either T_t or $T_{t'}$ and therefore, we can observe that $\varphi_{T_t} \cup \varphi_{T_{t'}} = \varphi_{T_p}$.

We start with a $(\alpha_1, \ldots, \alpha_k, d_{1,2}, \ldots, d_{k-1,k}) \in D_t$ and $(\alpha_1, \ldots, \alpha_k, d'_{1,2}, \ldots, d'_{k-1,k}) \in D_{t'}$. Now let $\gamma_1, \ldots, \gamma_k \in I(Q_{T_t})$ and $\gamma'_1, \ldots, \gamma'_k \in I(Q_{T_{t'}})$ witness this, respectively. By the above observation, $\gamma_1 \cup \gamma'_1, \ldots, \gamma_k \cup \gamma'_k \in I(Q_{T_p})$ and, for $1 \leq i < j \leq k$, 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, $(\alpha_1, \ldots, \alpha_k, d_{1,2} + d'_{1,2} - \Delta_X(\alpha_1, \alpha_2), \ldots, d_{k-1,k} + d'_{k-1,k} - \Delta_X(\alpha_{k-1}, \alpha_k)) \in D_p$.

Conversely, let $(\alpha_1, \ldots, \alpha_k, d''_{1,2}, \ldots, d''_{k-1,k}) \in D_p$, witnessed by $\gamma_1, \ldots, \gamma_k \in I(Q_{T_p})$. Thus, we can immediately conclude that the restrictions $\gamma_1|_{\chi(T_t)}, \ldots, \gamma_k|_{\chi(T_t)}$ are in $I(Q_{T_t})$ while the restrictions $\gamma_1|_{\chi(T_{t'})}, \ldots, \gamma_k|_{\chi(T_{t'})}$ are in $I(Q_{T_{t'}})$. This implies that

$$(\gamma_1|_{\chi(p)}, \ldots, \gamma_k|_{\chi(p)}, \Delta_X(\gamma_1|_{\chi(T_t)}, \gamma_2|_{\chi(T_t)}), \ldots, \Delta_X(\gamma_{k-1}|_{\chi(T_t)}, \gamma_k|_{\chi(T_t)})) \in D_t,$$
$$(\gamma_1|_{\chi(p)}, \ldots, \gamma_k|_{\chi(p)}, \Delta_X(\gamma_1|_{\chi(T_{t'})}, \gamma_2|_{\chi(T_{t'})}), \ldots, \Delta_X(\gamma_{k-1}|_{\chi(T_{t'})}, \gamma_k|_{\chi(T_{t'})})) \in D_{t'}.$$

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

$$\begin{aligned} d''_{i,j} &= \Delta_X(\gamma_i, \gamma_j) \\ &= \Delta_X(\gamma_i|_{\chi(T_t)}, \gamma_j|_{\chi(T_t)}) + \Delta_X(\gamma_i|_{\chi(T_{t'})}, \gamma_j|_{\chi(T_{t'})}) - \Delta_X(\gamma_i|_{\chi(p)}, \gamma_j|_{\chi(p)}), \end{aligned}$$

implying that $(\alpha_1, \ldots, \alpha_k, d''_{1,2}, \ldots, d''_{k-1,k})$ is in the set on the right-hand side of Equation 6.4. The equation is consequently valid.

The bound on the runtime can be achieved by a naive implementation using nested loops with the help of Equation 6.4. □

These lemmata put together then imply the following theorem.

Theorem 7. *Let Q be a conjunctive query with negation and free variables X, I a database with largest table R^I, and (T, χ) a nice tree decomposition of $G_p(Q)$. Then* Diverse-$CQ^\neg$ *can be solved in time $\mathcal{O}(|V(T)| \cdot \mathbf{dom}(I)^{2 \cdot k \cdot (\omega+1)} \cdot (|X|+1)^{k(k-1)} \cdot (|Q| \cdot |R^I| + k^2 \cdot \omega))$, where k is the number of sought-after solutions and ω is the width of the tree decomposition. In particular, the problem* Diverse-$CQ^\neg$ *is in XP combined complexity when parameterized by the number of sought-after solutions k plus the primal treewidth $tw_p(Q)$ of the query.*

Proof. The algorithm computes the sets D_t for each $t \in V(T)$ by a bottom-up procedure. For this, we have to apply exactly one of the lemmata once for each $t \in V(T)$. This is clearly possible in the required time bound.

For the final step of the algorithm, let r be the root of T and we can compute (gray brackets are for set semantics)

$$\begin{aligned}
\max_{\substack{\gamma_1,\dots,\gamma_k \in I(Q) \\ (\gamma_i \neq \gamma_j \text{ for } i \neq j)}} \delta_{\mathsf{sum}}(\gamma_1, \dots, \gamma_k) &= \max_{\substack{\gamma_1,\dots,\gamma_k \in I(Q_T) \\ (\gamma_i|_X \neq \gamma_j|_X \text{ for } i \neq j)}} \delta_{\mathsf{sum}}(\gamma_1|_X, \dots, \gamma_k|_X) \\
&= \max_{\substack{\gamma_1,\dots,\gamma_k \in I(Q_T) \\ (\Delta_X(\gamma_i,\gamma_j) > 0 \text{ for } i \neq j)}} \sum_{1 \leq i < j \leq k} \Delta_X(\gamma_i, \gamma_j) \\
&= \max_{\substack{(\alpha_1,\dots,\alpha_k, d_{1,2},\dots,d_{k-1,k}) \in D_r \\ (d_{i,j} > 0 \text{ for } 1 \leq i < j \leq k)}} \sum_{1 \leq i < j \leq k} d_{i,j},
\end{aligned}$$

$$\begin{aligned}
\max_{\substack{\gamma_1,\dots,\gamma_k \in I(Q) \\ (\gamma_i \neq \gamma_j \text{ for } i \neq j)}} \delta_{\mathsf{min}}(\gamma_1, \dots, \gamma_k) &= \max_{\substack{\gamma_1,\dots,\gamma_k \in I(Q_T) \\ (\gamma_i|_X \neq \gamma_j|_X \text{ for } i \neq j)}} \delta_{\mathsf{min}}(\gamma_1|_X, \dots, \gamma_k|_X) \\
&= \max_{\substack{\gamma_1,\dots,\gamma_k \in I(Q_T) \\ (\Delta_X(\gamma_i,\gamma_j) > 0 \text{ for } i \neq j)}} \min_{1 \leq i < j \leq k} \Delta_X(\gamma_i, \gamma_j) \\
&= \max_{\substack{(\alpha_1,\dots,\alpha_k, d_{1,2},\dots,d_{k-1,k}) \in D_r \\ (d_{i,j} > 0 \text{ for } 1 \leq i < j \leq k)}} \min_{1 \leq i < j \leq k} d_{i,j}.
\end{aligned}$$

Thus, we can read off the maximal possible diversity from D_r for summed and minimal Hamming distance as well as for set and bag semantics in the required time bound.

Lastly, as computing a width optimal nice tree decomposition is even possible in FPT time when parameterized by the treewidth (Kloks, 1994; Bodlaender, 1996), XP-membership follows immediately. □

Example 9. An execution of the described algorithm can be seen in Figure 6.2. The query is

$$Q : ans(x, y, z, u, v) \leftarrow \neg R_1(x, y) \wedge \neg R_2(x, y) \wedge \neg R_3(y, z) \wedge \neg R_4(y, u) \wedge \neg R_5(u, v)$$

and the database is given by

$$R_1^I = \{0, 0\},\ R_2^I = \{1, 0\},\ R_3^I = \{0, 0\},\ R_4^I = \{1, 0\},\ R_5^I = \{0, 0\},\ \mathbf{dom}(I) = \{0, 1\}.$$

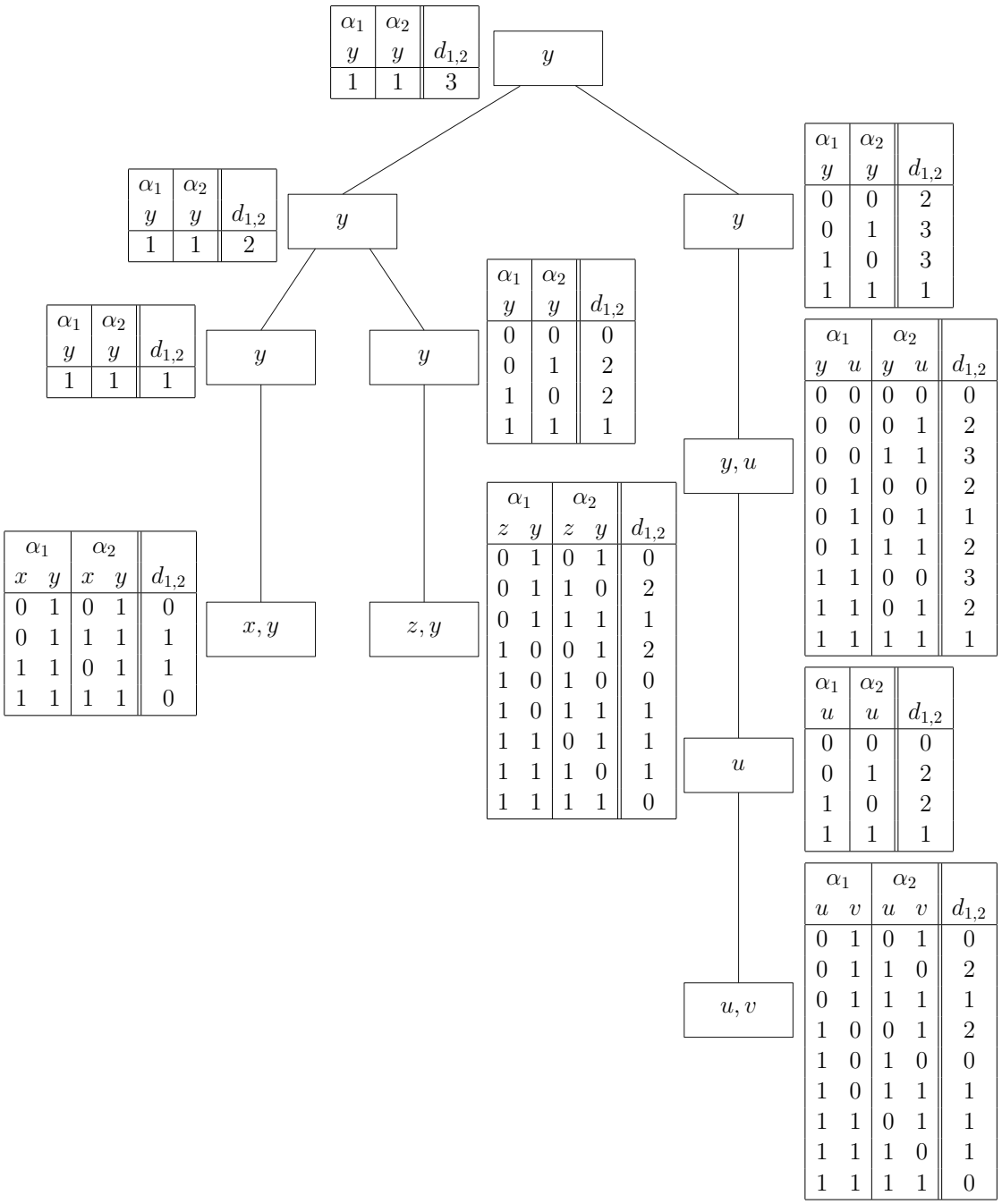


Figure 6.2: Solving Diverse-CQ$^\neg$ with the help the primal treewidth.

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 $(\alpha_1, \alpha_2, d'_{1,2}) \in D_t$, i.e., $d_{1,2} < d'_{1,2}$.

Concluding the description of the algorithm, we note that the ideas described in Sections 5.2.1 to 5.2.3 can also be applied to the primal treewidth algorithm (without full formal proofs). First of all, this means that we can construct k solutions that witness the maximal possible diversity by a bit of bookkeeping. For this the algorithm would again just have to keep track of the justifications for elements being in the sets D_t and then, combine the various local variable assignments that belong together.

Secondly, when we are interested in the problem $\mathsf{Diverse}_{\mathsf{sum}}$-CQ$^\neg$ (set or bag semantics), it is possible to combine the pairwise diversities $d_{i,j}$ into a singe value $d = \sum_{1 \leq i < j \leq k} d_{i,j}$. It is only necessary to keep the maximal possible diversity achievable per $\alpha_1, \ldots, \alpha_k$ combination. Thus, for bag semantics we can drop the factor $(|X| + 1)^{k(k-1)}$ in the runtime, and for set semantics we can replace it with a factor of $2^{k(k-1)}$. Consequently, when the number of domain elements $|\mathbf{dom}(I)|$ is bounded or part of the parameter, it is possible to solve $\mathsf{Diverse}_{\mathsf{sum}}$-CQ$^\neg$ in FPT time. This is for example the case when encoding a SAT instance into a CQ$^\neg$ (see Section 7.3 for more details).

Furthermore, for $\mathsf{Diverse}_{\mathsf{sum}}$-CQ$^\neg$ (bag semantics) another exponential speedup is possible. Notice that Lemma 7 is the reason why the factor 2 appears in the exponent of $\mathbf{dom}(I)^{2 \cdot k \cdot (\omega+1)}$. This is because we need to combine every tuple from D_t with every tuple from $D_{t'}$. However, if there is a single diversity value d per $\alpha_1, \ldots, \alpha_k$ combination, we only need to combine each element in D_t with a single element in $D_{t'}$. Thus, by assuming D_t and $D_{t'}$ to be sorted, this step is also possible in time $\mathbf{dom}(I)^{k \cdot (\omega+1)}$.

Lastly, we can solve $\mathsf{Diverse}_{\delta}$-CQ$^\neg$ in XP time for the more general diversity measures described in Section 5.2.3.

6.3 Fine Grained Analysis

Finally, we again turn our attention to the differences between data-, query-, and combined complexity for Diverse-CQ$^\neg$. It turns out that most of the ideas used in Chapter 5 apply here as well, and we just need to gather the arguments.

We start with combined complexity. Theorem 7 states that Diverse-CQ$^\neg$ is in XP combined complexity but, in fact, the proof of Theorem 2 already implies that Diverse-CQ$^\neg$ is W[1]-hard even if we restrict ourselves to queries with bounded primal treewidth.

Corollary 3. *The problem Diverse-CQ$^\neg$ is W[1]-hard combined complexity with the parameter being the number of sought-after solutions k plus the treewidth $tw_p(Q)$ of the query Q.*

In fact, the problems remain W[1]-hard for queries without negated atoms, without bound variables, and the treewidth being bounded by 1.

Proof. The proof of Theorem 2 actually already proves this corollary as the query used in the reduction is a CQ, has no bound variables, and has treewidth 1. □

We continue with query complexity. NP-hardness follows in the "unparameterized" case as deciding whether even a single solution of a CQ exists is already NP-hard (Chandra and Merlin, 1977).

Furthermore, XP-membership follows from the combined complexity case. But, as was for $\mathsf{Diverse}_{\mathsf{sum}}$-ACQ, in the case of $\mathsf{Diverse}_{\mathsf{sum}}$-CQ$^\neg$ we can do a bit better. The discussion at the end of the previous section ($\mathbf{dom}(I)$ is fixed) in fact implies that Diverse-CQ$^\neg$ is FPT query complexity.

Corollary 4. *The problem $\mathsf{Diverse}_{\mathsf{sum}}$-CQ$^\neg$ is FPT parameterized by the number of sought-after solutions k plus the primal treewidth $tw_p(Q)$ of the query Q.*

The case of data complexity is also easily taken care of. First notice that FPT-membership is already shown in Theorem 5. Furthermore, the NP-hardness of the "unparameterized" case follows already from the NP-hardness proof of Diverse-ACQ.

Corollary 5. *The problem Diverse-CQ$^\neg$ is NP-hard data complexity. In fact, the problem remains NP-hard for queries without bound variables, without negated atoms, and the treewidth being bounded by 4.*

Proof. Revisting the proof of Theorem 5, we notice that the query in use has the desired properties and thus, the proof also proves this corollary. □

The results for Diverse-CQ$^\neg$ are summarized in Table 6.2.

Complexity	Diversity	Semantics	Lower Bound	Upper Bound
combined	$\delta_{\mathsf{sum}}, \delta_{\mathsf{min}}$	bag/set	W[1]	XP
data	$\delta_{\mathsf{sum}}, \delta_{\mathsf{min}}$	bag/set	NP	FPT
query	δ_{sum}	bag/set		FPT
query	δ_{min}	bag/set		XP

Table 6.2: Results for Diverse-CQ$^\neg$.

CHAPTER 7

Diversity in Propositional Satisfiability

In this chapter, we tackle the problem Diverse-SAT. To that end, we will proceed as follows: First, we consider the formulae to be in CNF and analyze the problem parameterized by the number of sought-after solutions plus the incidence treewidth. The FPT-membership of $\text{Diverse}_{\text{sum}}$-CNF-SAT follows from an extension of Courcelle's Theorem (Courcelle, 1990) and is presented in Section 7.1. The technique used exemplifies how Courcelle's Theorem can generally be used to ensure FPT-membership of diversity problems.

Then in Section 7.2, a dynamic programming algorithms is presented that runs in time XP time and solves the more general problem Diverse-CNF-SAT. This algorithm can also be slightly modified to run in FPT time for the problem $\text{Diverse}_{\text{sum}}$-CNF-SAT.

In Section 7.3, we show how this carries over to Diverse-SAT. Moreover, in case of primal treewidth, we show how, by encoding the propositional formula into a $\text{CQ}^{\neg}$, we can use the algorithm from Section 6.2 to solving Diverse-SAT.

Lastly, we consider the case of Diverse-DNF-SAT in Section 7.4.

7.1 FPT-Membership

We turn our attention to the problem $\text{Diverse}_{\text{sum}}$-CNF-SAT parameterized by the incidence treewidth. As bounded primal treewidth implies bounded incidence treewidth (Kolaitis and Vardi, 2000), the achieved FPT result also implies FPT-membership of $\text{Diverse}_{\text{sum}}$-CNF-SAT parameterized by the primal treewidth.

The basic form of Courcelle's Theorem allows us to answer questions expressible in monadic second order logic (MSO) about graphs of bounded treewidth in linear time (Courcelle, 1990). To show that it is also possible to compute diverse solutions in the

same setting, we will use an extension introduced by Arnborg et al. (1991). They showed that certain optimization problems are also solvable in linear time on graphs of bounded treewidth. To understand the extension, the reader is first reminded of what MSO logic is.

MSO logic is an extension of FO logic where instead of only allowing basic quantification, one is allowed to quantify over *monadic* predicates (predicates of arity one). We will use set notation for monadic predicates and write $a \in A$ instead of $A(a)$. When quantifying over sets (monadic predicates), we will use upper case letters ($\exists X$) and when quantifying over domain elements, we will use lower case letter ($\exists x$).

We will use MSO logic to express graph properties and thus, only require predicates of arity one and two. Furthermore, we will allow our graphs in this section to have multiple kinds of edges and the vertices to possibly be colored. The semantics of $G \vDash \psi$ for a graph G and MSO formula ψ is then defined in the usual way. That is, the domain elements are the vertices of G, predicates of arity two correspond to the different kinds of edges of the graph, and predicates of arity one correspond to colors.

The optimization variant of Courcelle's Theorem can then be stated as follows:

Theorem 8 (Arnborg et al., 1991)**.** *Let G be a graph, $\psi(Y_1, \ldots, Y_l)$ an MSO formula with free (set) variables $Y_1, \ldots, Y_l$, and $g : \mathbb{N}^l \to \mathbb{N}$ a linear function. Then*

$$\max_{\substack{A_1 \ldots, A_l \subseteq V(G) \\ G \vDash \psi(A_1, \ldots, A_l)}} g(|A_1|, \ldots, |A_l|)$$

can be computed in time $f(tw(G), |\psi|, |g|) \cdot |G|$ for some computable function f.

We now want to apply Theorem 8 to $\mathsf{Diverse}_{\mathsf{sum}}$-CNF-SAT. To this end, let φ be a propositional formula in CNF. In this section, we differentiate the two kinds of vertices, i.e., clauses and variables, in $G_i(\varphi)$ by the colors C and X, respectively. Furthermore, we differentiate two kinds of edges in $G_i(\varphi)$. If the variable x appears positively in the clause c, then the edge xc corresponds to the atom $e_+(x, c)$ and otherwise to the atom $e_-(x, c)$.

Now consider the following MSO formula:

$$\begin{aligned} \zeta(B) : \ & \forall x : (x \in B \to x \in X) \\ & \wedge \forall c : \Big(C(c) \to \exists x : \big((x \in B \wedge e_+(x, c)) \vee (x \notin B \wedge e_-(x, c)) \big) \Big) \end{aligned}$$

The intended meaning of ζ is to interpret a set of variables $B \subseteq \mathbf{var}(\varphi)$ as the variable assignment $\gamma_B : \mathbf{var}(\varphi) \to \{0, 1\}$ defined by $\gamma_B(x) = 1$ for $x \in B$ and $\gamma_B(x) = 0$ for $x \notin B$. Then, the models of ζ exactly correspond to the models of φ, as γ_B satisfies φ if and only if $G_i(\varphi) \vDash \zeta(B)$.

To get from the formula ζ representing the solutions to a formula ψ and linear function g that represents the diversity problem, we first introduce a macro for the symmetric difference of two sets. We write $A = B\Delta C$ for

$$\forall v : v \in A \leftrightarrow (v \in B \leftrightarrow v \notin C).$$

Now consider the formula ψ and function g (gray part is needed for set semantics):

$$\psi(A_{1,2}, \ldots, A_{k-1,k}) : \exists B_1, \ldots, B_k \bigwedge_{i=1,\ldots,k} \zeta(B_k) \wedge \bigwedge_{1 \le i < j \le k} A_{ij} = B_i \Delta B_j \wedge \exists x : x \in A_{i,j},$$

$$g(y_1, \ldots, y_{\frac{k(k-1)}{2}}) = \sum_{i=1}^{\frac{k(k-1)}{2}} y_i.$$

In ψ, the sets $B_1, \ldots, B_k$ play the role of the k sought-after solutions of φ while $A_{1,2}, \ldots, A_{k-1,k}$ are all the symmetric differences. Thus, we can compute,

$$\begin{aligned}
\max_{\substack{A_{1,2},\ldots,A_{k-1,k} \subseteq V(G) \\ G \models \psi(A_{1,2},\ldots,A_{k-1,k})}} g(|A_{1,2}|, \ldots, |A_{k-1,k}|) &= \max_{\substack{B_1,\ldots,B_k \subseteq V(G) \\ G \models \zeta(B_1),\ldots,\zeta(B_k) \\ B_i \neq B_j \text{ for } i \neq j}} \sum_{1 \le i < j \le k} |B_i \Delta B_j| \\
&= \max_{\substack{B_1,\ldots,B_k \subseteq V(G) \\ G \models \zeta(B_1),\ldots,\zeta(B_k) \\ B_i \neq B_j \text{ for } i \neq j}} \sum_{1 \le i < j \le k} \Delta(\gamma_{B_i}, \gamma_{B_j}) \\
&= \max_{\substack{\gamma_1,\ldots,\gamma_k \in \mathcal{M}(\varphi) \\ \gamma_i \neq \gamma_j \text{ for } i \neq j}} \delta_{\text{sum}}(\gamma_1, \ldots, \gamma_k).
\end{aligned}$$

Combining these arguments with Theorem 8 shows that we can solve the problem $\mathsf{Diverse}_{\mathsf{sum}}$-CNF-SAT in linear time on formulae of bounded incidence treewidth and with a bound on k.

Corollary 6. *The problem $\mathsf{Diverse}_{\mathsf{sum}}$-CNF-SAT (set and bag semantics) can be solved in time $f(tw_i(\varphi), k) \cdot |\varphi|$, where φ is the input formula, k is the number of sought-after solutions, and f is a computable function.*

This method can clearly be used for many different $\mathsf{Diverse}_{\mathsf{sum}}$-$\mathcal{X}$ problems. The important ingredients are the following:

- We need to convert each instance I into a graph $G(I)$.
- We need to express the solutions $\mathcal{S}(I)$ via sets of vertices.
- We need to be able to identify solutions, i.e., the corresponding set of vertices, via an MSO formula ζ.
- The diversity in a collection of solutions has to be the same as the diversity in the corresponding vertex sets.

Corollary 7. *Let $\mathcal{X}$ be a problem, $G(I)$ a graph associated with an instance I of $\mathcal{X}$, $\iota : \mathcal{S}(I) \to 2^{V(G(I))}$ an injective function, and $\zeta(B)$ an MSO formula such that $G(I) \vDash \zeta(B)$ if and only if B is in the image of ι. If $\Delta(\gamma_i, \gamma_j) = |\iota(\gamma_i) \Delta \iota(\gamma_j)|$ and $G(I)$ can be efficiently computed, then $\mathsf{Diverse}_{\mathsf{sum}}$-$\mathcal{X}$ can be solved in time $f(tw(G(I)), k) \cdot |G(I)|$.*

For example, Corollary 7 implies FPT-membership of Diverse$_{\mathsf{sum}}$-Vertex-Cover and Diverse$_{\mathsf{sum}}$-Independent-Set. However, to be in alignment with our definition of Δ and δ_{sum} for that matter, we have to represent solutions $S \subseteq V(G)$ by their characteristic function $\mathbf{1}_S : V(G) \to \{0, 1\}$. Note that $\Delta(\mathbf{1}_S, \mathbf{1}_{S'}) = |S \Delta S'|$.

Concluding this section, the reader is reminded that the FPT algorithms constructed in the proof of Courcelle's Theorem or the extensions by Arnborg et al. (1991) for that matter are not well suited for practical applications. The problem being that the function f grows absurdly fast. Thus, the main benefit of these tools is in the ability to classify problems. Corollary 6 and 7 should be understood in the same way and to that end, the algorithm presented in the following is of independent interest.

7.2 Incidence Treewidth Algorithm

In this section we will design a dynamic programming algorithm which uses a tree decomposition of the incidence graph of the input formula as a guide to solve Diverse-CNF-SAT.

Henceforth, let φ be propositional formula in CNF, X the variables that appear in φ, and (T, χ) a nice tree decomposition of $G_i(\varphi)$ with root r. Each $\chi(t)$ consists of variables and clauses. Therefore, let $C(t) \subseteq \chi(t)$ be the clauses and $X(t) \subseteq \chi(t)$ the variables. Furthermore, let $C(T_t) = \bigcup C(V(T_t))$ and $X(T_t) = \bigcup X(V(T_t))$ for subtrees T_t of T rooted in t.

We note that a clause $C \in \varphi$ is satisfied if any of its literals is set to true (1). Thus, if a partial assignment $\alpha : Z \to \{0, 1\}, Z \subseteq \mathbf{var}(\varphi)$ sets a literal in C to true (1) we already know that every extension $\gamma : \mathbf{var}(\varphi) \to \{0, 1\}, \alpha \cong \gamma$ satisfies C. Thus, we will relax our definition of $\vDash$ in this section and write $\alpha \vDash C$ in such a case. Moreover, for a set of clauses $\mathcal{C}$ we write $\alpha \vDash \mathcal{C}$ when $\alpha \vDash C$ holds for every $C \in \mathcal{C}$.

The dynamic programming algorithm will compute the set

$$D_t \subseteq (X(t) \to \{0, 1\})^k \times (2^{C(t)})^k \times [|X(T_t)|]^{\frac{k(k-1)}{2}}$$

bottom-up for each $t \in V(T)$. The intended meaning of D_t is such that a tuple $(\alpha_1, \ldots, \alpha_k, S_1, \ldots, S_k, d_{1,2}, \ldots, d_{k-1,k})$ is in D_t if it is possible to extend the mappings $\alpha_1, \ldots, \alpha_k$ to mappings $\gamma_1, \ldots, \gamma_k$ which all satisfy all forgotten clauses plus the clauses in $S_1, \ldots, S_k$, and simultaneously each pair $\gamma_i, \gamma_j, i < j$ achieves the diversity $d_{i,j}$. Put differently, D_t keeps track of all simultaneously achievable pairwise diversities for solutions to a subproblem per local variable assignment. The concrete subproblem is determined

by t and $S_1, \ldots, S_k$. Formally, we define D_t as

$$D_t = \{(\alpha_1, \ldots, \alpha_k, S_1, \ldots, S_k, d_{1,2}, \ldots, d_{k-1,k}) : \alpha_l : X(t) \to \{0,1\}, l = 1, \ldots, k, \\ \gamma_l : X(T_t) \to \{0,1\}, \gamma_l \cong \alpha_l, \\ S_l \subseteq C(t), \gamma_l \vDash (C(T_t) \setminus C(t)) \cup S_l, \\ d_{i,j} = \Delta(\gamma_i, \gamma_j), 1 \leq i < j \leq k\}.$$

Note that the set D_t consists of at most $2^{k \cdot \chi(t)} \cdot (|X(T_t)| + 1)^{\frac{k(k-1)}{2}}$ elements, each of size at most $k \cdot \chi(t) + \frac{k(k-1)}{2}$.

Lemma 8. *Let t be a leaf node of T. Then*

$$D_t = \{(\alpha_1, \ldots, \alpha_k, S_1, \ldots, S_k, d_{1,2}, \ldots, d_{k-1,k}) : \alpha_l : X(t) \to \{0,1\}, l = 1, \ldots, k, \\ S_l \subseteq C(t), \alpha_l \vDash S_l, \\ d_{i,j} = \Delta(\alpha_i, \alpha_j), 1 \leq i < j \leq k\}. \quad (7.1)$$

Furthermore, the set D_t can be computed in time $\mathcal{O}(2^{k \cdot |\chi(t)|} k \cdot |\chi(t)| \cdot (|\boldsymbol{var}(C)| + k))$, where $C \in C(t)$ is the clause with the most variables.

Proof. Observe that $\alpha_1, \ldots, \alpha_k$ are the only extensions of $\alpha_1, \ldots, \alpha_k$ and, furthermore, $(C(T_t) \setminus C(t)) \cup S_i = S_i$. Thus, Equation 7.1 holds.

The set D_t can be computed by considering all possibilities of $(\alpha_1, \ldots, \alpha_k, S_1, \ldots, S_k)$ and using Equation 7.1. There are $2^{k \cdot |\chi(t)|}$ possibilities and checking whether an α_i sets a literal in every clause of S_i to true can be done naively in time $\mathcal{O}(|S_i| \cdot |\mathbf{var}(C)|)$. The overall runtime follows, as computing $\Delta(\alpha_i, \alpha_j)$ only takes time $\mathcal{O}(|X(t)|)$. □

Lemma 9. *Let p be an introduce node of T which introduces the variable x and t is its child. Let $C(p, x, 1)$ be the clauses from $C(p)$ in which x appears positively and let $C(p, x, 0)$ be the clauses in which x appears negatively. Then*

$$D_p = \{(\alpha'_1, \ldots, \alpha'_k, S'_1, \ldots, S'_k, d'_{1,2}, \ldots, d'_{k-1,k}) : \\ (\alpha_1, \ldots, \alpha_k, S_1, \ldots, S_k, d_{1,2}, \ldots, d_{k-1,k}) \in D_t \\ \beta_l : \{x\} \to \{0,1\}, l = 1, \ldots, k, \\ \alpha'_l = \alpha_l \cup \beta_l, \\ S'_l \setminus C(p, x, \beta_l(x)) = S_l, \\ d'_{i,j} = d_{i,j} + \Delta(\beta_i, \beta_j), 1 \leq i < j \leq k\}. \quad (7.2)$$

Furthermore, the set D_p can be computed in time $\mathcal{O}(2^{k \cdot |\chi(p)|} |X(T_p)|^{\frac{k(k-1)}{2}} \cdot k \cdot (k + |\chi(t)|))$ given the set D_t.

Proof. First let $(\alpha'_1, \ldots, \alpha'_k, S'_1, \ldots, S'_k, d'_{1,2}, \ldots, d'_{k-1,k})$ be a tuple in the set on the right-hand side of Equation 7.2 and let $\alpha_l, \beta_l, S_l, d_{i,j}$ witness this. Furthermore, let $\gamma_1, \ldots, \gamma_k$ be extensions which witness that $(\alpha_1, \ldots, \alpha_k, S_1, \ldots, S_k, d_{1,2}, \ldots, d_{k-1,k})$ is in D_t, i.e., each γ_l satisfies $(C(T_t) \setminus C(t)) \cup S_l$ and $\Delta(\gamma_i, \gamma_j) = d_{i,j}$. Thus, as $C(T_p) = C(T_t), C(p) = C(t)$, and β_l satisfies $C(p, x, \beta_l(x))$, we can conclude that the variable assignment $\gamma_l \cup \beta_l$ satisfies $(C(T_p) \setminus C(p)) \cup S_l \cup C(p, x, \beta_l(x)) \supseteq (C(T_p) \setminus C(p)) \cup S'_l$. We can also compute for each $1 \leq i < j \leq k$:

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

Hence, $(\alpha'_1, \ldots, \alpha'_k, S'_1, \ldots, S'_k, d'_{1,2}, \ldots, d'_{k-1,k})$ is in D_p as each $\gamma_l \cup \beta_l$ is an extension of α'_l.

For the reverse direction, let $(\alpha'_1, \ldots, \alpha'_k, S'_1, \ldots, S'_k, d'_{1,2}, \ldots, d'_{k-1,k}) \in D_t$ be an arbitrary tuple witnessed by $\gamma'_1, \ldots, \gamma'_k$. We define $\alpha_l = \alpha'_l|_{X(t)}, \beta_l = \alpha'_l|_{\{x\}} \gamma_l = \gamma'_l|_{X(T_t)}$, $d_{i,j} = \Delta(\gamma_i, \gamma_j)$, and $S_l = S'_l \setminus C(p, x, \gamma'_l(x))$. By the definition of $C(p, x, \gamma'_l(x))$ it is clear that γ_l must satisfy S_l. Furthermore, the variable x cannot appear in any clause of $C(T_t) \setminus C(t)$ due to the properties of a tree decomposition. Thus, γ_l additionally satisfies $C(T_t) \setminus C(t) = C(T_p) \setminus C(p)$. This implies that $(\alpha_1, \ldots, \alpha_k, S_1 \ldots, S_k, d_{1,2}, \ldots, d_{k-1,k})$ appears in D_t, witnessed by $\gamma_1, \ldots, \gamma_k$. Lastly, we note that

$$\begin{aligned}d'_{i,j} &= \Delta(\gamma'_i, \gamma'_j) \\ &= \Delta(\gamma_i, \gamma_j) + \Delta(\gamma'_i|_{\{x\}}, \gamma'_j|_{\{x\}}) \\ &= d_{i,j} + \Delta(\beta_i, \beta_j)\end{aligned}$$

and hence, $(\alpha'_1, \ldots, \alpha'_k, S'_1, \ldots, S'_k, d'_{1,2}, \ldots, d'_{k-1,k})$ appears in the set on the right-hand side of Equation 7.2.

We compute D_p as follows by using Equation 7.2: We first compute $C(p, x, 1)$ and $C(p, x, 0)$ in time $\mathcal{O}(|C(p)| \cdot |\mathbf{var}(C(p))|)$. We then iterate through all $2^{k \cdot |X(p)|} 2^{k \cdot |C(p)|} \cdot |X(T_p)|^{\frac{k(k-1)}{2}}$ possibilities for $(\alpha'_1, \ldots, \alpha'_k, S'_1, \ldots, S'_k, d'_{1,2}, \ldots, d'_{k-1,k})$. We then compute $\alpha_l, S_l, d_{i,j}$ as described by the equation. Then, by using appropriate data structure, we can check whether $(\alpha_1, \ldots, \alpha_k, S_1, \ldots, S_k, d_{1,2}, \ldots, d_{k-1,k})$ is in D_t or not. This all takes per tuple only time $\mathcal{O}(k \cdot |X(p)| + k \cdot |C(p)| + k^2)$. In total, computing D_p only requires time $\mathcal{O}(2^{k \cdot |\chi(p)|} |X(T_p)|^{\frac{k(k-1)}{2}} \cdot k \cdot (k + |\chi(t)|))$. □

Lemma 10. *Let p be an introduce node of T which introduces the clause C and t is its*

child. Then

$$D_p = \{(\alpha_1, \ldots, \alpha_k, S'_1, \ldots, S'_k, d_{1,2}, \ldots, d_{k-1,k}) : (\alpha_1, \ldots, \alpha_k, S_1, \ldots, S_k, d_{1,2}, \ldots, d_{k-1,k}) \in D_t, L = \{l \in \{1, \ldots, k\} : \alpha_l \vDash C\}, S_l \subseteq S'_l \subseteq S_l \cup \{C\}, l \in L, S'_l = S_l, l \in \{1, \ldots, k\} \setminus L\}. \quad (7.3)$$

Furthermore, the set D_p can be computed in time $\mathcal{O}(|D_t| \cdot (2^k \cdot (k \cdot \chi(t) + k^2) + k \cdot |\boldsymbol{var}(C)|))$ given the set D_t.

Proof. Due to the properties of a tree decomposition and the incidence graph, no variables that appear in C can already have been forgotten, i.e., they cannot be in $X(T_p) \setminus X(p)$. Therefore, any extension of α_i onto the variables $X(T_p)$ can only satisfy C if α_i already satisfies C. Thus, the tuples in D_p are the same as the tuples in D_t except that we may add C to a set of clauses S_l if α_l satisfies C. This is exactly what Equation 7.3 states.

Computing D_p is very simple as this just corresponds to duplicating each line of D_t up to 2^k times, adding C to the clause sets or not. Checking whether we are allowed to add C requires per element in D_t and per $l = 1, \ldots, k$ the check $\alpha_l \vDash C$, which can be done in time $\mathcal{O}(|\mathbf{var}(C)|)$. Thus, naively we require time $\mathcal{O}(|D_t| \cdot (2^k \cdot (k \cdot \chi(t) + k^2) + k \cdot |\mathbf{var}(C)|))$. □

Lemma 11. *Let p be a forget node of T which forgets the variable x and t is its child. Then*

$$D_p = \{(\alpha_1|_{X(p)}, \ldots, \alpha_k|_{X(p)}, S_1, \ldots, S_k, d_{1,2}, \ldots, d_{k-1,k}) : (\alpha_1, \ldots, \alpha_k, S_1, \ldots, S_k, d_{1,2}, \ldots, d_{k-1,k}) \in D_t\}.$$

Furthermore, the set D_p can be computed in time $\mathcal{O}(|D_t| \cdot (k \cdot \chi(t) + k^2))$ given the set D_t.

Proof. One just has to observe that extensions of $\alpha' : X(p) \to \{0, 1\}$ are also extensions of $\alpha : X(t) \to \{0, 1\}$. Computing D_p comes down to a projection, hence the time requirement. □

Lemma 12. *Let p be a forget node of T which forgets the clause C and t is its child. Then*

$$D_p = \{(\alpha_1, \ldots, \alpha_k, S_1, \ldots, S_k, d_{1,2}, \ldots, d_{k-1,k}) : (\alpha_1, \ldots, \alpha_k, S'_1, \ldots, S'_k, d_{1,2}, \ldots, d_{k-1,k}) \in D_t, S'_l = S_l \cup \{C\}, l = 1, \ldots, k\}.$$

Furthermore, the set D_p can be computed in time $\mathcal{O}(|D_t| \cdot (k \cdot \chi(t) + k^2))$ given the set D_t.

Proof. Here, one just has to observe that $(C(T_p) \setminus C(p)) \cup S_1 = (C(T_t) \setminus C(t)) \cup S_1 \cup C$. Also, computing D_p just corresponds to dropping elements and deleting columns from D_t. Hence, it is doable in time $\mathcal{O}(|D_t| \cdot (k \cdot \chi(t) + k^2))$. □

Lemma 13. *Let p be a join node of T with children t' and t''. Then*

$$\begin{aligned} D_p = \{(\alpha_1, \ldots, \alpha_k, S_1, \ldots, S_k, d_{1,2}, \ldots, d_{k-1,k}) : \\ (\alpha_1, \ldots, \alpha_k, S'_1, \ldots, S'_k, d'_{1,2}, \ldots, d'_{k-1,k}) \in D_{t'}, \\ (\alpha_1, \ldots, \alpha_k, S''_1, \ldots, S''_k, d''_{1,2}, \ldots, d''_{k-1,k}) \in D_{t''}, \\ S_l = S'_l \cup S''_l, l = 1, \ldots, k, \\ d_{i,j} = d'_{i,j} + d''_{i,j} - \Delta(\alpha_i, \alpha_j), 1 \leq i < j \leq k\}. \end{aligned} \quad (7.4)$$

Furthermore, the set D_p can be computed in time $\mathcal{O}(|D_{t'}| \cdot |D_{t''}| \cdot k \cdot (k + |\chi(p)|))$ given the sets $D_{t'}$ and $D_{t''}$.

Proof. Let $(\alpha_1, \ldots, \alpha_k, S_1, \ldots, S_k, d_{1,2}, \ldots, d_{k-1,k})$ be a tuple in the set on the right-hand side of Equation 7.4 and let $S'_l, S''_l, d'_{i,j}, d''_{i,j}$ witness this. Furthermore, let $\gamma'_1, \ldots, \gamma'_k$ witness that $(\alpha_1, \ldots, \alpha_k, S'_1, \ldots, S'_k, d'_{1,2}, \ldots, d'_{k-1,k})$ is in $D_{t'}$ and, analogously, let $\gamma''_1, \ldots, \gamma''_k$ witness that $(\alpha_1, \ldots, \alpha_k, S''_1, \ldots, S''_k, d''_{1,2}, \ldots, d''_{k-1,k})$ is in $D_{t''}$. Thus, each γ'_l satisfies $(C(T_{t'}) \setminus C(t')) \cup S'_l$ and each γ''_l satisfies $(C(T_{t''}) \setminus C(t'')) \cup S''_l$. Together, $\gamma'_l \cup \gamma''_l$ satisfy $(C(T_p) \setminus C(p)) \cup S_l$. Computing

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

ensures that $(\alpha_1, \ldots, \alpha_k, S_1, \ldots, S_k, d_{1,2}, \ldots, d_{k-1,k})$ is in D_p.

For the reverse direction, let $(\alpha_1, \ldots, \alpha_k, S_1, \ldots, S_k, d_{1,2}, \ldots, d_{k-1,k})$ be in D_p and let $\gamma_1, \ldots, \gamma_k$ witness this. We define $\gamma'_l = \gamma_l|_{X(T_{t'})}, \gamma''_l = \gamma_l|_{X(T_{t''})}, d'_{i,j} = \Delta(\gamma'_i, \gamma'_j)$, and $d''_{i,j} = \Delta(\gamma''_i, \gamma''_j)$. Notice that $d_{i,j} = d'_{i,j} + d''_{i,j} - \Delta(\alpha_i, \alpha_j)$. Furthermore, let S'_l be the clauses from S_l that are satisfies by γ'_l and analogously let S''_l be the clauses from S_l that are satisfies by γ''_l. As γ_l satisfies S_l, it has to be the case that $S'_l \cup S''_l = S_l$. Furthermore, each forgotten clause $C \in C(T_p) \setminus C(p)$ either has to be forgotten in the subtree $T_{t'}$ or $T_{t''}$. Thus, also all variables from C appear in the same subtree and hence, γ'_l satisfies the forgotten clauses $C(T_{t'}) \setminus C(t')$ while γ''_l satisfies the forgotten clauses $C(T_{t''}) \setminus C(t'')$. Consequently, $(\alpha_1, \ldots, \alpha_k, S'_1, \ldots, S'_k, d'_{1,2}, \ldots, d'_{k-1,k}) \in D_{t'}$ and $(\alpha_1, \ldots, \alpha_k, S''_1, \ldots, S''_k, d''_{1,2}, \ldots, d''_{k-1,k}) \in D_{t''}$. Put together, this ensures that $(\alpha_1, \ldots, \alpha_k, S_1, \ldots, S_k, d_{1,2}, \ldots, d_{k-1,k})$ also appears in the set on the right-hand side and hence, Equation 7.4 holds.

Computing D_p via Equation 7.4 requires us to go through all pairs of elements in $D_{t'}$ and $D_{t''}$. A naive implementation using nested loops achieves the required runtime. □

With these lemmata we can describe the whole algorithm and bound its runtime.

Theorem 9. *Let φ be a formula in CNF, $X = \boldsymbol{var}(\varphi)$, $C \in \varphi$ the clause with the most variables, and (T, χ) a nice tree decomposition of $G_i(\varphi)$. Then, Diverse-CNF-SAT can be solved in time $\mathcal{O}(4^{k\cdot(\omega_i+1)} \cdot (|X|+1)^{k(k-1)} \cdot k^2 \cdot \omega_i \cdot |\boldsymbol{var}(C)| \cdot |V(T)|)$, where k is the number of sought-after solutions and ω_i is the width of the tree decomposition. In particular, the problem Diverse-CNF-SAT is in XP when parameterized by the number of sought-after solutions k plus the treewidth of $G_i(\varphi)$.*

Proof. The Lemmata 8 through 13 ensure that we can compute the table D_t for each $t \in V(T)$ by a bottom-up procedure in the required time bound. We can then read of the maximal diversity achievable by k models of φ at the root $r \in V(T)$. This is true as $C(T_r) = \varphi$, $X(T_r) = \mathbf{var}(\varphi)$, and (gray brackets are for set semantics)

$$\max_{\substack{\gamma_1,\ldots,\gamma_k \in \mathcal{M}(\varphi) \\ (\gamma_i \neq \gamma_j \text{ for } i \neq j)}} \delta_{\mathsf{sum}}(\gamma_1, \ldots, \gamma_k) = \max_{\substack{\gamma_1,\ldots,\gamma_k \in \mathcal{M}(C(T_r)) \\ (\gamma_i \neq \gamma_j \text{ for } i \neq j)}} \sum_{1 \leq i < j \leq k} \Delta(\gamma_i, \gamma_j)$$

$$= \max_{\substack{(\alpha_1,\ldots,\alpha_k, C(r),\ldots,C(r), d_{1,2},\ldots,d_{k-1,k}) \in D_r \\ (d_{i,j} > 0 \text{ for } 1 \leq i < j \leq k)}} \sum_{1 \leq i < j \leq k} d_{i,j},$$

$$\max_{\substack{\gamma_1,\ldots,\gamma_k \in \mathcal{M}(\varphi) \\ (\gamma_i \neq \gamma_j \text{ for } i \neq j)}} \delta_{\mathsf{min}}(\gamma_1, \ldots, \gamma_k) = \max_{\substack{\gamma_1,\ldots,\gamma_k \in \mathcal{M}(C(T_r)) \\ (\gamma_i \neq \gamma_j \text{ for } i \neq j)}} \min_{1 \leq i < j \leq k} \Delta(\gamma_i, \gamma_j)$$

$$= \max_{\substack{(\alpha_1,\ldots,\alpha_k, C(r),\ldots,C(r), d_{1,2},\ldots,d_{k-1,k}) \in D_r \\ (d_{i,j} > 0 \text{ for } 1 \leq i < j \leq k)}} \min_{1 \leq i < j \leq k} d_{i,j}.$$

The right-hand side can clearly be obtained from D_r in the required time bound. This completes the proof. □

Example 10. An execution of the described algorithm can be seen in Figure 7.1, solving φ of Examples 1 and 2. For the sake of succinctness, tuples $(\alpha_1, \alpha_2, S_1, S_2, d_{1,2}) \in D_{T'}$ are omitted if there is a strictly better tuple $(\alpha_1, \alpha_2, S'_1, S'_2, d'_{1,2}) \in D_{T'}$, i.e., $S_1 \subseteq S'_1$, $S_2 \subseteq S'_2, d_{1,2} \leq d'_{1,2}$, and the tuples are distinct to each other.

Concluding the description of the algorithm, we once again note that the ideas described in the Sections 5.2.1 to 5.2.3 can also be applied to the incidence treewidth algorithm (without full formal proofs). Reconstructing solutions works in the same way as before. The algorithm needs to keep track of why it is justified to add a tuple to D_t and then, at the end of the algorithm, one needs to follow the justification of the tuple in D_r which has the highest diversity (and satisfies $C(r)$).

Furthermore, when we are interested in the problem $\mathsf{Diverse}_{\mathsf{sum}}$-CNF-SAT (set or bag semantics), it is possible to combine the pairwise diversities $d_{i,j}$ into a single value $d = \sum_{1 \leq i < j \leq k} d_{i,j}$. It is also only necessary to keep the maximal possible diversity achievable per $\alpha_1, \ldots, \alpha_k, S_1, \ldots, S_k$ combination. Thus, for bag semantics we can drop the factor $(|X|+1)^{k(k-1)}$ in the runtime, and for set semantics we can replace it with a factor of $2^{k(k-1)}$. This gives us for both cases an FPT algorithm.

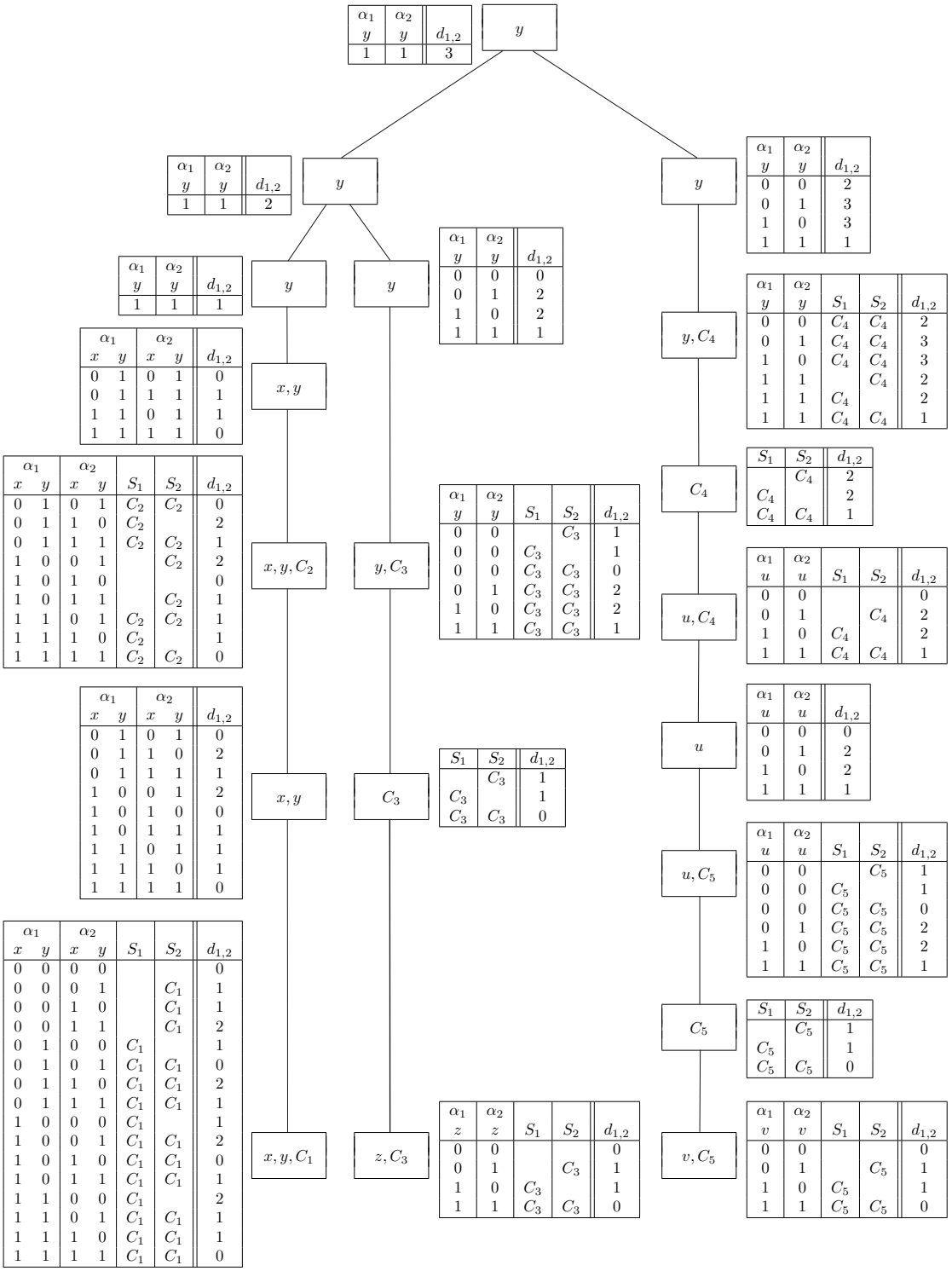


Figure 7.1: Solving Diverse-CNF-SAT with the help of the incidence treewidth.

Lastly, we can solve $\mathsf{Diverse}_\delta$-CNF-SAT in the time bound given in Theorem 9 for more general diversity measures δ as described in Section 5.2.3.

7.3 From Diverse-SAT over Diverse-CNF-SAT to Diverse-CQ¬

In classical complexity theory, it does not make much sense to distinguish between the problems SAT and CNF-SAT. The reason being that an arbitrary propositional formula φ can be transformed in polynomial time into a CNF formula ψ which is satisfiable if and only if φ is satisfiable. To that end, most SAT-solvers require the input formula to be in CNF, as only such formulae are expressable in the most prevalent input format *DIMACS CNF*.

The most widely known method to perform this transformation is Tseitin transformation, named after Grigori Tseitin (Tseitin, 1983). This is an iterative process that introduces new variables Y such that finally

$$\varphi(X) \equiv \exists Y \psi(X, Y),$$

where X are the variables used in φ and X, Y are the ones used in ψ. Thus, it is clear that satisfiability is preserved, but k maximally diverse models $\gamma_1, \ldots, \gamma_k \in \mathcal{M}(\psi)$ of ψ need not necessarily correspond to k maximally diverse models $\gamma_1|_X, \ldots, \gamma_k|_X \in \mathcal{M}(\varphi)$ of φ. However, this problem can be eradicated by computing the diversity of a pair $\gamma_i, \gamma_j \in \mathcal{M}(\psi)$ with a different function Δ_X.

Let δ_X, Δ_X again be defined as

$$\delta_X(\gamma_1, \ldots, \gamma_k) = \delta(\gamma_1|_X, \ldots, \gamma_j|_X), \ \Delta_X(\gamma_i, \gamma_j) = \Delta(\gamma_i|_X, \gamma_j|_X),$$

where δ is either δ_{sum} or δ_{min}. Then it is clear that (gray brackets are for set semantics)

$$\begin{aligned}
\max_{\substack{\gamma_1,\ldots,\gamma_k \in \mathcal{M}(\varphi) \\ (\gamma_i \neq \gamma_j \text{ for } i \neq j)}} \delta_{\mathsf{sum}}(\gamma_1, \ldots, \gamma_k) &= \max_{\substack{\gamma_1,\ldots,\gamma_k \in \mathcal{M}(\psi) \\ (\gamma_i|_X \neq \gamma_j|_X \text{ for } i \neq j)}} \sum_{1 \leq i < j \leq k} \Delta_X(\gamma_i, \gamma_j), \\
&= \max_{\substack{\gamma_1,\ldots,\gamma_k \in \mathcal{M}(\psi) \\ (\Delta_X(\gamma_i,\gamma_j) > 0 \text{ for } i \neq j)}} \delta_X(\gamma_1, \ldots, \gamma_k), \\
\max_{\substack{\gamma_1,\ldots,\gamma_k \in \mathcal{M}(\varphi) \\ (\gamma_i \neq \gamma_j \text{ for } i \neq j)}} \delta_{\mathsf{min}}(\gamma_1, \ldots, \gamma_k) &= \max_{\substack{\gamma_1,\ldots,\gamma_k \in \mathcal{M}(\psi) \\ (\gamma_i|_X \neq \gamma_j|_X \text{ for } i \neq j)}} \min_{1 \leq i < j \leq k} \Delta_X(\gamma_i, \gamma_j) \\
&= \max_{\substack{\gamma_1,\ldots,\gamma_k \in \mathcal{M}(\psi) \\ (\Delta_X(\gamma_i,\gamma_j) > 0 \text{ for } i \neq j)}} \delta_X(\gamma_1, \ldots, \gamma_k).
\end{aligned}$$

Therefore, to solve the problem Diverse-SAT, it suffices to use the incidence treewidth algorithm when we swap δ for δ_X and Δ for Δ_X.

Corollary 8. *Let φ be a propositional formula, ψ the satisfiability-equivalent formula obtained by Tseitin transformation, $X = \boldsymbol{var}(\psi)$, $C \in \psi$ the clause with the most variables,*

and (T, χ) a nice tree decomposition of $G_i(\psi)$. Then the problem Diverse-SAT can be solved in time $\mathcal{O}(4^{k\cdot(\omega_i+1)} \cdot (|X|+1)^{k(k-1)} \cdot k^2 \cdot \omega_i \cdot |\boldsymbol{var}(C)| \cdot |V(T)|)$, where k is the number of sought-after solutions and ω_i is the width of the tree decomposition. In particular, the problem Diverse-SAT is in XP when parameterized by the number of sought-after solutions k plus the treewidth of $G_i(\psi)$.

Proof (sketch). The Lemmata 8 through 13 remain true when we swap Δ for Δ_X. Thus, Theorem 9 remains true when we consider the diversity measure δ_X instead of δ and, for set semantics, we consider two models $\gamma_i, \gamma_j \in \mathcal{M}(\psi)$ to be distinct only if $\Delta_X(\gamma_i, \gamma_j) > 0$. □

We now turn our attention to the step from Diverse-CNF-SAT to Diverse-CQ$^\neg$. To that end, it is well know that every CNF formula φ can be encoded into a CQ$^\neg$ (see for example Lanzinger, 2021). The idea is that each clause C in φ is satisfied by all but one assignment $\alpha_C : \mathbf{var}(C) \to \{0, 1\}$. The assignment α_C that does no satisfy C has to set each literal in C to false (0). This assignment can thus also be excluded by the single negative literal $\neg R_C(\mathbf{var}(C))$ in a CQ$^\neg$ and the corresponding database I also only needs a single tuple per clause, i.e., $R_C^I = \{\alpha_C(\mathbf{var}(C))\}$ suffices. In total, the query is $Q : ans(\mathbf{var}(\varphi)) \leftarrow \bigwedge_{C\in\varphi} \neg R_C(\mathbf{var}(C))$ and the database I consists of the tables R_C^I. This encoding is such that $I(Q) = \mathcal{M}(\varphi)$ and hence, solving Diverse-CNF-SAT for φ is the same as solving Diverse-CQ$^\neg$ for I, Q. We can therefore use the primal treewidth algorithm for Diverse-CQ$^\neg$ to solve Diverse-CNF-SAT and, even more generally, Diverse-SAT.

Corollary 9. *Let φ be a propositional formula, ψ the satisfiability-equivalent formula obtained by Tseitin transformation, $X = \boldsymbol{var}(\psi)$, and (T, χ) a nice tree decomposition of the primal graph $G_p(\psi)$. Then the problem Diverse-SAT can be solved in time $\mathcal{O}(4^{k\cdot(\omega_p+1)} \cdot (|X|+1)^{k(k-1)} \cdot (|\psi| + k^2 \cdot \omega_p) \cdot |V(T)|)$, where k is the number of sought-after solutions, ω_p is the width of the tree decomposition, and $|\psi|$ is the number of symbols in the formula ψ. In particular, the problem Diverse-SAT is in XP when parameterized by the number of sought-after solutions k plus the treewidth of $G_p(\psi)$.*

Proof. We can obtain Q and I from ψ as detailed before. Furthermore, let Y be the variables introduced by Tseitin transformation, i.e., $\varphi = \exists Y \psi$. We define the query $Q_Y : ans(\mathbf{var}(\varphi)) \leftarrow \exists Y \bigwedge_{C\in\varphi} \neg R_C(\mathbf{var}(C))$ and thus, as $I(Q) = \mathcal{M}(\psi)$, it is clear that $I(Q_Y) = \mathcal{M}(\exists Y \psi) = \mathcal{M}(\varphi)$. Furthermore, notice that $G_p(\psi)$ and $H(Q_Y)$ have the same tree decompositions. Therefore, applying Theorem 7 to the instance Q_Y, I and tree decomposition (T, χ) proves the corollary. □

Example 11. The query and database used in Example 9 is obtained by transforming the formula φ of Example 1 into a CQ$^\neg$. Thus, also the example execution of the primal treewidth algorithm for Diverse-CQ$^\neg$ depicted in Figure 6.2 can be seen as an example execution of the primal treewidth algorithm for Diverse-SAT.

We now have two algorithms to choose from to solve Diverse-SAT. Recall, however, that every class of graphs with bounded primal treewidth also has bounded incidence treewidth

while the reverse statement is not necessarily true (Kolaitis and Vardi, 2000). Same applies to the parameter dual treewidth which is often also considered in this setting (see for example Samer and Szeider, 2010). Thus, the incidence treewidth algorithm can be seen as a bit more general.

But, in the case of $\mathsf{Diverse}_{\mathsf{sum}}$-SAT (bag semantics), the primal treewidth algorithm has a significant advantage. We argued at the end of Section 6.2 that we can decrease the exponential asymptotic runtime of $4^{k\cdot(w_p+1)}$ to $2^{k\cdot(w_p+1)}$ in this case. Note that the same idea does not work for the incidence treewidth algorithm and thus, in this case, we are still left with $4^{k\cdot(w_i+1)}$.

Consequently, it is not always clear which of the two algorithm should be preferred. A comparison of both algorithm on benchmarks would therefore be helpful but this is outside the scope of this thesis.

7.4 The Case of Diverse-DNF-SAT

We now turn our attention to Diverse-DNF-SAT. In contrast to the fragment CNF-SAT, the fragment DNF-SAT of SAT is tractable without any additional parameter like incidence treewidth. Thus, we can also hope for Diverse-DNF-SAT to be tractable parameterized only by the number of sought-after solutions k. We show that this is at least the case for $\mathsf{Diverse}_{\mathsf{sum}}$-DNF-SAT (bag semantics) when k is bounded. However, we also show that $\mathsf{Diverse}_{\mathsf{sum}}$-DNF-SAT (bag semantics) and, even more generally, that Diverse-DNF-SAT is W[1]-hard. Thus, although the algorithm described in the following section to some extend brute forces the problem, there likely does not exist a significantly better method from a theoretical point of view – in particular no FPT algorithm.

7.4.1 Brute Force Approach

In the following, let $\varphi = \bigvee_{i=1}^{n} D_i$ be a propositional formula in DNF, i.e., each D_i is of the form $D_i = \bigwedge_{j=1}^{m_i} l_{i,j}$, where each $l_{i,j}$ is a literals. We assume that each variable appears at most once in each conjunct D_i as multiple occurrences are either redundant or make D_i unsatisfiable. We define the variable assignment $\alpha_{D_i} : \mathbf{var}(D_i) \to \{0,1\}$ by

$$\alpha_{D_i}(x) = \begin{cases} 1 & x \text{ appears in } D_i, \\ 0 & \neg x \text{ appears in } D_i. \end{cases}$$

Thus, by definition $\alpha_{D_i} \vDash D_i$ and every extension $\gamma : \mathbf{var}(\varphi) \to \{0,1\}$ of α_{D_i} satisfies φ.

Conversely, every model $\gamma : \mathbf{var}(\varphi) \to \{0,1\}$ of φ must satisfy at least one conjunct D_i and hence, must be an extension of α_{D_i}. Thus, the maximal possible diversity can be computed with the help of

$$\max_{\gamma_1,\dots,\gamma_k \in \mathcal{M}(\varphi)} \delta(\gamma_1,\dots,\gamma_k) = \max_{\alpha_1,\dots,\alpha_k \in \{\alpha_{D_1},\dots,\alpha_{D_n}\}} \max_{\substack{\gamma_1,\dots,\gamma_k : \mathbf{var}(\varphi)\to\{0,1\} \\ \gamma_1 \cong \alpha_1,\dots,\gamma_k \cong \alpha_k}} \delta(\gamma_1,\dots,\gamma_k). \quad (7.5)$$

Furthermore, the inner maximum can be directly computed without needing to go through all extensions.

Lemma 14. *Let $\alpha_1, \ldots, \alpha_k \in \{\alpha_{D_1}, \ldots, \alpha_{D_n}\}$ be variable assignments. Furthermore, for each $x \in \boldsymbol{var}(\varphi)$ let*

$$d(\alpha_1, \ldots, \alpha_k, x) = \begin{cases} l(k-l) & l > \frac{k}{2} \text{ of the variable assignments set } x \text{ to } 1, \\ l(k-l) & l > \frac{k}{2} \text{ of the variable assignments set } x \text{ to } 0, \\ \lfloor \frac{k}{2} \rfloor \lceil \frac{k}{2} \rceil & \text{otherwise.} \end{cases}$$

Then

$$\max_{\substack{\gamma_1,\ldots,\gamma_k:\boldsymbol{var}(\varphi)\to\{0,1\} \\ \gamma_1 \cong \alpha_1,\ldots,\gamma_k \cong \alpha_k}} \delta(\gamma_1, \ldots, \gamma_k) = \sum_{x \in \boldsymbol{var}(\varphi)} d(\alpha_1, \ldots, \alpha_k, x). \tag{7.6}$$

Proof. For $j = 1, \ldots, k$ let $\alpha_{D_{i_j}}$ be α_j and let $\gamma_j : \mathbf{var}(\varphi) \to \{0,1\}$ be an extension of $\alpha_j : \mathbf{var}(D_{i_j}) \to \{0,1\}$. The important observation is the fact that the values of $\gamma_j(x)$ for $x \in \mathbf{var}(\varphi) \setminus \mathbf{var}(D_{i_j})$ do not impact that γ_j satisfies φ. Thus, they only impact the achieved diversity and are in the best case picked to maximize the diversity of the values $\gamma_1(x), \ldots, \gamma_k(x)$. Put differently, we can maximize independently for each $x \in \mathbf{var}(\varphi)$, i.e.,

$$\max_{\substack{\gamma_1,\ldots,\gamma_k:\mathbf{var}(\varphi)\to\{0,1\} \\ \gamma_1 \cong \alpha_1,\ldots,\gamma_k \cong \alpha_k}} \delta(\gamma_1, \ldots, \gamma_k) = \sum_{x \in \mathbf{var}(\varphi)} \max_{\substack{\gamma_1,\ldots,\gamma_k:\{x\}\to\{0,1\} \\ \gamma_1 \cong \alpha_1,\ldots,\gamma_k \cong \alpha_k}} \delta(\gamma_1, \ldots, \gamma_k).$$

We now show that

$$\max_{\substack{\gamma_1,\ldots,\gamma_k:\{x\}\to\{0,1\} \\ \gamma_1 \cong \alpha_1,\ldots,\gamma_k \cong \alpha_k}} \delta(\gamma_1, \ldots, \gamma_k) = d(\alpha_1, \ldots, \alpha_k, x).$$

If $l > \frac{k}{2}$ variables assignments α_j set x to 1, the best we can do is set x to 0 in the remaining $k-l$ variable assignment. Doing this, we get diversity $l(k-l) = d(\alpha_1, \ldots, \alpha_k, x)$, i.e., $d(\alpha_1, \ldots, \alpha_k, x)$ is exactly the maximal possible diversity on the variable x. The case that $l > \frac{k}{2}$ variable assignments set x to 0 follows analogously. Lastly, if neither more than half of the variable assignments fix x to 0 nor to 1, the best we can do is to set x to 0 in half the cases and to 1 in the other half. If k is odd, we have to set x to either 0 or 1 one more time than the other. This gives us a diversity of $\lfloor \frac{k}{2} \rfloor \lceil \frac{k}{2} \rceil = d(\alpha_1, \ldots, \alpha_k, x)$, i.e., $d(\alpha_1, \ldots, \alpha_k, x)$ is exactly the maximal possible diversity on the variable x.

Consequently Equation 7.6 is valid. □

Equations 7.5 and 7.6 give a straightforward way to determine the maximal achievable diversity and thus, a procedure based on this solves $\mathsf{Diverse}_{\mathsf{sum}}$-DNF-SAT (bag semantics).

Theorem 10. *Let φ be a formula in DNF. Then, $\mathsf{Diverse}_{\mathsf{sum}}$-DNF-SAT (bag semantics) can be solved in time $\mathcal{O}(|\varphi|^{k+1} \cdot k)$, where k is the number of sought-after solutions. In particular, $\mathsf{Diverse}_{\mathsf{sum}}$-DNF-SAT (bag semantics) is in XP.*

Proof. Let n be the number of conjuncts in φ. This runtime is achieved by a naive implementation that iterates trough the n^k possibilities for $\alpha_1, \ldots, \alpha_k \in \{\alpha_{D_1}, \ldots, \alpha_{D_n}\}$ and computes the maximal possible diversity as Equations 7.5 and 7.6 suggest. Computing $d(\alpha_1, \ldots, \alpha_k, x)$ can clearly be done in time $\mathcal{O}(k)$ and has to be performed once for each variable in $\mathbf{var}(\varphi)$. Correctness of this procedure is guaranteed by Lemma 14. □

Concluding, we note that also determining k witnessing solutions can be done, straightforwardly, in the given time bound.

7.4.2 W[1]-Hardness

We now show the W[1]-hardness of Diverse-DNF-SAT with the number of sought-after solutions k being the parameter. For this we give a reduction from the Independent-Set problem, where the parameter is the size of the sought-after independent set k'.

Theorem 11. *The problem Diverse-DNF-SAT is W[1]-hard parameterized by the number of sought-after solutions k.*

Proof. We first give a reduction form Independent-Set to Diverse-DNF-SAT and then prove its correctness.

Reduction. Let (G, k') be an instance of Independent-Set with $V(G) = \{v_1, \ldots, v_n\}$ and $E(G) = \{e_1, \ldots, e_m\}$. We will use the variables

$$e_1^1, \ldots, e_m^1, \ldots, e_1^n, \ldots, e_m^n,$$
$$e_1^*, \ldots, e_m^*$$

and the conjuncts

$$D_1, \ldots, D_n$$

for the definition of our Diverse-DNF-SAT instance. We note that each variable will appear in each conjunct. Let $t = 1, \ldots, m$ and $i, j = 1, \ldots, n$. Each variable e_t^j shall only appear positively in the conjunct D_i if $i = j$ and v_i is not incident to e_t, and otherwise negatively. Each variable e_t^* appears positively in D_i if v_i is incident to e_t and otherwise negatively.

The number of solutions one is allowed to pick is $k = k'$ and the minimum target diversity is

$$d_{\mathsf{sum}} = k(k-1)m, \quad d_{\mathsf{min}} = 2m,$$

respectively for $\mathsf{Diverse}_{\mathsf{sum}}$-DNF-SAT and $\mathsf{Diverse}_{\mathsf{min}}$-DNF-SAT. The instances are $(\bigvee_{i=1}^n D_i, k, d_{\mathsf{sum}})$ and $(\bigvee_{i=1}^n D_i, k, d_{\mathsf{min}})$, respectively. Both can clearly be computed in polynomial time.

Correctness. First notice that $\varphi = \bigvee_{i=1}^n D_i$ has exactly n models, which are $\alpha_{D_1}, \ldots, \alpha_{D_n}$. Recall that we defined

$$\alpha_{D_i}(x) = \begin{cases} 1 & x \text{ appears in } D_i, \\ 0 & \neg x \text{ appears in } D_i, \end{cases}$$

for variables $x \in \mathbf{var}(D_i) = \mathbf{var}(\varphi)$. The idea of the reduction is that picking the vertices $v_{i_1}, \ldots, v_{i_k}$ corresponds to picking the models $\alpha_{D_{i_1}}, \ldots, \alpha_{D_{i_k}}$ and vice versa. This is done as the conjuncts are constructed such that each α_{D_i} sets exactly m variables to true, one for each edge. Furthermore, the target diversity is so high, that k models can only achieve this diversity if each model sets m different variables to true. Thus, we also do not have to differentiate between set and bag semantics. The variables $e^i_1, \ldots, e^i_m$ are "non competitive" for a model α_{D_i} as, if any, only this model can set these variables to true. However, if v_i is incident to e_t, the variable e^i_t is set to false while the variable e^*_t is set to true. But, the model α_{D_i} has to "compete" with the model corresponding to the other vertex v_j incident to e_t for e^*_t. Thus, only either α_{D_i} or α_{D_j} can be picked when meeting the target diversity. It is therefore clear that $v_{i_1}, \ldots, v_{i_k}$ is an independent set of size k if and only if $(\alpha_{D_{i_1}}, \ldots, \alpha_{D_{i_k}})$ is a tuple of k distinct models with diversity (at least) d_{sum} or d_{min}, respectively. □

Example 12. Consider the instance (G, k') of Independent-Set with $k' = 2$ and with graph G' depicted in Figure 7.2. Our problem reduction yields the following formula φ in DNF:

$$\begin{aligned}\varphi : \quad & (\neg e^1_1 \wedge \neg e^1_2 \wedge \neg e^2_1 \wedge \neg e^2_2 \wedge \neg e^3_1 \wedge \neg e^3_2 \wedge \quad e^*_1 \wedge \quad e^*_2) \\ \vee\ & (\neg e^1_1 \wedge \neg e^1_2 \wedge \neg e^2_1 \wedge \quad e^2_2 \wedge \neg e^3_1 \wedge \neg e^3_2 \wedge \quad e^*_1 \wedge \neg e^*_2) \\ \vee\ & (\neg e^1_1 \wedge \neg e^1_2 \wedge \neg e^2_1 \wedge \neg e^2_2 \wedge \quad e^3_1 \wedge \neg e^3_2 \wedge \neg e^*_1 \wedge \quad e^*_2).\end{aligned}$$

One can see that models corresponding to rows two and three maximize diversity for $k = 2$, and vertices v_2, v_3 form an independent set of size $k' = 2$.

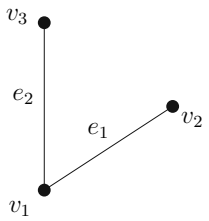


Figure 7.2: A graph used for the DNF reduction.

CHAPTER 8

Conclusion and Future Work

Baste et al. (2019) initiated the study of diversity problems from the perspective of parameterized complexity. In this thesis, we followed in their footsteps and took a look at database querying problems as well as SAT. To that end, we showed how to solve Diverse-CQ, Diverse-CQ$^\neg$, and Diverse-SAT by leveraging the acyclicity measures treewidth and hypertreewidth. The developed dynamic programming algorithms only require polynomial time if we assume the parameter to be bounded. Our algorithms can be broadly applied to diversity problems and even to the dual similarity problem.

Furthermore, for the database problems, we showed that an FPT algorithm is unlikely due to W[1]-hardness. In fact, Diverse-ACQ is already W[1]-hard and adding unions to the query language increases the complexity of the diversity problem considerable. For this, we showed that the very restrictive case of Diverse-UACQ is already NP-hard even if we bound the number of sought-after solutions by two.

Nevertheless, for fixed first order queries, we also presented an FPT kernelization process. This algorithm may prove to work well in practice as assuming the query to be fixed seems to be reasonable since the size of a real world query usually is only tiny in comparison to the size of the database.

Lastly, we showed that the problem Diverse-DNF-SAT is already W[1]-hard while $\text{Diverse}_{\text{sum}}$-DNF-SAT (bag semantics) is in XP.

Hence, especially, we were able to classify the diverse variant of many natural problems solvable in polynomial time. This is of interest as the classification is only known for a handful of problems and it exemplifies that the complexity of these problems is a priori not at all clear. Our results show that these problems can remain solvable in polynomial time (the best case), be in FPT or XP, or it can be NP-hard to find even a constant number of diverse solutions (usually the worst case).

Future Work. As far as the problems considered in this thesis are concerned, it would be interesting to consider further structural restrictions and parameters. For $\mathrm{CQ}^\neg$, the acyclicity notion β-acyclicity (Brault-Baron, 2012; Brault-Baron et al., 2015) and the novel complementary acyclicity measure nest-set-width (Lanzinger, 2021) seem to be a promising starting point. For SAT, the parameters considered by Sæther et al. (2015) could be analyzed first as they have already been used to establish tractable fragments when interested in counting the number of models.

On a different note, it would also be of interest to formally prove the intuitive difference in hardness between diversity measures. In this thesis, in particular when considering query complexity and SAT, it seems as if $\mathsf{Diverse}_{\mathsf{min}}$-$\mathcal{X}$ is slightly harder than $\mathsf{Diverse}_{\mathsf{sum}}$-$\mathcal{X}$. To that end, recall that we showed that $\mathsf{Diverse}_{\mathsf{sum}}$-CNF-SAT is FPT while we were only able to prove XP-membership for $\mathsf{Diverse}_{\mathsf{min}}$-CNF-SAT. However, we could not rule out the existence of an FPT algorithm for $\mathsf{Diverse}_{\mathsf{min}}$-CNF-SAT.

From a more high level point of view, analyzing the diverse variant of problems from the perspective of parameterized complexity is quite new and thus, still wide open to further research. As the diversity paradigm can be applied to almost any computational problem, the possible new directions are likewise almost countless.

List of Figures

List of Tables

Bibliography

Adler, I., Gottlob, G., and Grohe, M. (2007). Hypertree width and related hypergraph invariants. *Eur. J. Comb.*, 28(8):2167–2181.

Alimonti, P. and Kann, V. (1997). Hardness of approximating problems on cubic graphs. In Bongiovanni, G. C., Bovet, D. P., and Battista, G. D., 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.

Angelsmark, O. and Thapper, J. (2004). Algorithms for the maximum hamming distance problem. In Faltings, B., Petcu, A., Fages, F., and Rossi, F., editors, *Recent Advances in Constraints, Joint ERCIM/CoLogNet International Workshop on Constraint Solving and Constraint Logic Programming, CSCLP 2004, Lausanne, Switzerland, June 23-25, 2004, Revised Selected and Invited Papers*, volume 3419 of *Lecture Notes in Computer Science*, pages 128–141. Springer.

Arnborg, S., Lagergren, J., and Seese, D. (1991). Easy problems for tree-decomposable graphs. *J. Algorithms*, 12(2):308–340.

Bailleux, O. and Marquis, P. (1999). DISTANCE-SAT: Complexity and algorithms. In Hendler, J. and Subramanian, D., editors, *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence, July 18-22, 1999, Orlando, Florida, USA*, pages 642–647. AAAI Press / The MIT Press.

Baste, J., Fellows, M. R., Jaffke, L., Masarík, T., de Oliveira Oliveira, M., Philip, G., and Rosamond, F. A. (2022). Diversity of solutions: An exploration through the lens of fixed-parameter tractability theory. *Artif. Intell.*, 303:103644.

Baste, J., Jaffke, L., Masarík, T., Philip, G., and Rote, G. (2019). FPT algorithms for diverse collections of hitting sets. *Algorithms*, 12(12):254.

Bodlaender, H. L. (1996). A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317.

Brault-Baron, J. (2012). A negative conjunctive query is easy if and only if it is beta-acyclic. In Cégielski, P. and Durand, A., editors, *Computer Science Logic (CSL'12) - 26th International Workshop/21st Annual Conference of the EACSL, CSL 2012, September 3-6, 2012, Fontainebleau, France*, volume 16 of *LIPIcs*, pages 137–151. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Brault-Baron, J. (2016). Hypergraph acyclicity revisited. *ACM Comput. Surv.*, 49(3):54:1–54:26.

Brault-Baron, J., Capelli, F., and Mengel, S. (2015). Understanding model counting for beta-acyclic cnf-formulas. In Mayr, E. W. and Ollinger, N., editors, *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany*, volume 30 of *LIPIcs*, pages 143–156. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Chandra, A. K. and Merlin, P. M. (1977). Optimal implementation of conjunctive queries in relational data bases. In Hopcroft, J. E., Friedman, E. P., and Harrison, M. A., editors, *Proceedings of the 9th Annual ACM Symposium on Theory of Computing, May 4-6, 1977, Boulder, Colorado, USA*, pages 77–90. ACM.

Chlebík, M. and Chlebíková, J. (2006). Hard coloring problems in low degree planar bipartite graphs. *Discret. Appl. Math.*, 154(14):1960–1965.

Cook, S. A. (1971). The complexity of theorem-proving procedures. In Harrison, M. A., Banerji, R. B., and Ullman, J. D., editors, *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, pages 151–158. ACM.

Courcelle, B. (1990). Graph rewriting: An algebraic and logic approach. In van Leeuwen, J., editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pages 193–242. Elsevier and MIT Press.

Crescenzi, P. and Rossi, G. (2002). On the hamming distance of constraint satisfaction problems. *Theor. Comput. Sci.*, 288(1):85–100.

Cygan, M., Fomin, F. V., Kowalik, L., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., and Saurabh, S. (2015). *Parameterized Algorithms*. Springer.

Danna, E. and Woodruff, D. L. (2009). How to select a small set of diverse solutions to mixed integer programming problems. *Oper. Res. Lett.*, 37(4):255–260.

Deng, T. and Fan, W. (2014). On the complexity of query result diversification. *ACM Trans. Database Syst.*, 39(2):15:1–15:46.

Downey, R. G. and Fellows, M. R. (1999). *Parameterized Complexity*. Monographs in Computer Science. Springer.

Downey, R. G. and Fellows, M. R. (2013). *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer.

Drosou, M. and Pitoura, E. (2010). Search result diversification. *SIGMOD Rec.*, 39(1):41–47.

Eiter, T., Erdem, E., Erdogan, H., and Fink, M. (2013). Finding similar/diverse solutions in answer set programming. *Theory Pract. Log. Program.*, 13(3):303–359.

Flum, J. and Grohe, M. (2006). *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer.

Fomin, F. V., Golovach, P. A., Jaffke, L., Philip, G., and Sagunov, D. (2020). Diverse pairs of matchings. In Cao, Y., Cheng, S., and Li, M., editors, *31st International Symposium on Algorithms and Computation, ISAAC 2020, December 14-18, 2020, Hong Kong, China (Virtual Conference)*, volume 181 of *LIPIcs*, pages 26:1–26:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

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

Gottlob, G., Leone, N., and Scarcello, F. (2002a). Hypertree decompositions and tractable queries. *J. Comput. Syst. Sci.*, 64(3):579–627.

Gottlob, G., Miklós, Z., and Schwentick, T. (2009). Generalized hypertree decompositions: Np-hardness and tractable variants. *J. ACM*, 56(6):30:1–30:32.

Gottlob, G., Scarcello, F., and Sideri, M. (2002b). Fixed-parameter complexity in AI and nonmonotonic reasoning. *Artif. Intell.*, 138(1-2):55–86.

Graham, M. H. (1979). On The Universal Relation. Technical report, University of Toronto.

Hanaka, T., Kobayashi, Y., Kurita, K., Lee, S. W., and Otachi, Y. (2021a). Computing diverse shortest paths efficiently: A theoretical and experimental study. *CoRR*, abs/2112.05403.

Hanaka, T., Kobayashi, Y., Kurita, K., and Otachi, Y. (2021b). Finding diverse trees, paths, and more. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 3778–3786. AAAI Press.

Hebrard, E., Hnich, B., O'Sullivan, B., and Walsh, T. (2005). Finding diverse and similar solutions in constraint programming. In Veloso, M. M. and Kambhampati, S., 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.

Hebrard, E., O'Sullivan, B., and Walsh, T. (2007). Distance constraints in constraint satisfaction. In Veloso, M. M., editor, *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 106–111.

Holyer, I. (1981). The NP-completeness of edge-coloring. *SIAM J. Comput.*, 10(4):718–720.

Ingmar, L., de la Banda, M. G., Stuckey, P. J., and Tack, G. (2020). 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.

Karp, R. M. (1972). Reducibility among combinatorial problems. In Miller, R. E. and Thatcher, J. W., editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York.

Kloks, T. (1994). *Treewidth, Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer.

Kolaitis, P. G. and Vardi, M. Y. (2000). Conjunctive-query containment and constraint satisfaction. *J. Comput. Syst. Sci.*, 61(2):302–332.

Lanzinger, M. (2021). Tractability beyond ß-acyclicity for conjunctive queries with negation. In Libkin, L., Pichler, R., and Guagliardo, P., editors, *PODS'21: Proceedings of the 40th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Virtual Event, China, June 20-25, 2021*, pages 355–369. ACM.

Levin, L. A. (1973). Universal sequential search problems. *Problemy peredachi informatsii*, 9(3):115–116.

Nadel, A. (2011). Generating diverse solutions in SAT. In Sakallah, K. A. and Simon, L., 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.

Petit, T. and Trapp, A. C. (2015). Finding diverse solutions of high quality to constraint optimization problems. In Yang, Q. and Wooldridge, M. J., 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.

Pichler, R. and Skritek, S. (2013). Tractable counting of the answers to conjunctive queries. *J. Comput. Syst. Sci.*, 79(6):984–1001.

Robertson, N. and Seymour, P. D. (1984). Graph minors. III. Planar tree-width. *J. Comb. Theory, Ser. B*, 36(1):49–64.

Sæther, S. H., Telle, J. A., and Vatshelle, M. (2015). Solving #SAT and MAXSAT by dynamic programming. *J. Artif. Intell. Res.*, 54:59–82.

Samer, M. and Szeider, S. (2010). Algorithms for propositional model counting. *J. Discrete Algorithms*, 8(1):50–64.

Schaefer, T. J. (1978). The complexity of satisfiability problems. In Lipton, R. J., Burkhard, W. A., Savitch, W. J., Friedman, E. P., and Aho, A. V., editors, *Proceedings of the 10th Annual ACM Symposium on Theory of Computing, May 1-3, 1978, San Diego, California, USA*, pages 216–226. ACM.

Tarjan, R. E. and Yannakakis, M. (1984). Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM J. Comput.*, 13(3):566–579.

Tseitin, G. S. (1983). On the complexity of derivation in propositional calculus. In *Automation of reasoning*, pages 466–483. Springer.

Vardi, M. Y. (1982). The complexity of relational query languages (extended abstract). In Lewis, H. R., Simons, B. B., Burkhard, W. A., and Landweber, L. H., editors, *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*, pages 137–146. ACM.

Vieira, M. R., Razente, H. L., Barioni, M. C. N., Hadjieleftheriou, M., Srivastava, D., Jr., C. T., and Tsotras, V. J. (2011). On query result diversification. In Abiteboul, S., Böhm, K., Koch, C., and Tan, K., editors, *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany*, pages 1163–1174. IEEE Computer Society.

Yannakakis, M. (1981). 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.

Yu, C. T. and Özsoyoğlu, M. Z. (1979). 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.

Ziegler, C., McNee, S. M., Konstan, J. A., and Lausen, G. (2005). Improving recommendation lists through topic diversification. In Ellis, A. and Hagino, T., editors, *Proceedings of the 14th international conference on World Wide Web, WWW 2005, Chiba, Japan, May 10-14, 2005*, pages 22–32. ACM.