UNIVERSITY OF CALIFORNIA SAN DIEGO

The Fine-Grained Complexity of Problems Expressible by First-Order Logic and Its Extensions

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Computer Science

by

Jiawei Gao

Committee in charge:

Professor Russell Impagliazzo, Chair
Professor Sam Buss
Professor Jiawang Nie
Professor Ramamohan Paturi
Professor Victor Vianu

2019

The dissertation of Jiawei Gao is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

_____

_____

Chair

University of California San Diego

2019

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

First of all I would like to thank my advisor Prof. Russell Impagliazzo for his invaluable help all these years. I would also like to thank Prof. Ramamohan Paturi for his help on all our projects. I would also like to acknowledge our co-authors Antonina Kolokolova, Ryan Williams, Marco Carmosino, Ivan Mihajlin, Stefan Schneider. I am glad that I took Prof. Victor Vianu's database theory class, which raised my interest in first-order logic and model checking. I would also like to thank theory group students Anant Dhayal, Sasank Mouli, Jessica Sorrell and Jiapeng Zhang for helpful discussions about different problems. I am also thankful to my husband Jian Yang who gave me lots of support through this program. Finally I would like to thank coffee and diet coke.

Chapter 2 contains material from "Nondeterministic Extensions of the Strong Exponential Time Hypothesis and Consequences for Non-Reducibility", by Marco L Carmosino, Jiawei Gao, Russell Impagliazzo, Ivan Mihajlin, Ramamohan Paturi, and Stefan Schneider, which appeared in the proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science (ITCS 2016). The author of this dissertation was a principal author of this publication. The material in this chapter is copyright ©2016 by Association for Computing Machinery. We would like to thank Amir Abboud, Karl Bringmann, Bart Jansen, Sebastian Krinninger, Virginia Vassilevska Williams, Ryan Williams and the anonymous reviewers for many helpful comments on an earlier draft.

Chapter 3 contains material from "Completeness for First-Order Properties on Sparse Structures with Algorithmic Applications", by Jiawei Gao, Russell Impagliazzo, Antonina Kolokolova, and Ryan Williams, which appeared in the proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2017). The author of this dissertation was a principal author of this publication. The material in this chapter is copyright ©2017 by Association for Computing Machinery and Society for Industrial and Applied Mathematics. We would thank Virginia Vassilevska Williams for her inspiring ideas. We would like to thank Marco Carmosino, Anant Dhayal, Ivan Mihajlin and Victor Vianu for proofreading and suggestions on this paper. We

also thank Valentine Kabanets, Ramamohan Paturi, Ramyaa Ramyaa and Stefan Schneider for many useful discussions. Finally, we really appreciate the suggestions from the reviewers about the writing and expression.

Chapter 4 contains material from "The Fine-Grained Complexity of Strengthenings of First-Order Logic", by Jiawei Gao and Russell Impagliazzo, which is currently in submission. The author of this dissertation was a principal author of this work. The authors sincerely thank Marco Carmosino and Antonina Kolokolova for comments on improving this paper.

Chapter 5 contains material from "On the Fine-grained Complexity of Least Weight Subsequence in Multitrees and Bounded Treewidth DAGs", by Jiawei Gao, to appear in International Symposium on Parameterized and Exact Computation (IPEC 2019). The author of this dissertation was a principal author of this publication. The author would like to thank Russell Impagliazzo for his guidance and advice on this paper, and thank Marco Carmosino, Anant Dhayal and Jessica Sorrell for helpful comments.

VITA

2013	B. E. in Software Engineering, Fudan University, Shanghai, China
2019	Ph. D. in Computer Science, University of California, San Diego

ABSTRACT OF THE DISSERTATION

The Fine-Grained Complexity of Problems Expressible by First-Order Logic and Its Extensions

by

Jiawei Gao

Doctor of Philosophy in Computer Science

University of California San Diego, 2019

Professor Russell Impagliazzo, Chair

This dissertation studies the fine-grained complexity of model checking problems for fixed logical formulas on sparse input structures.

The Orthogonal Vectors problem is an important and well-studied problem in fine-grained complexity: its hardness is implied by the Strong Exponential Time Hypothesis, and its hardness implies the hardness of many other interesting problems. We show that the Orthogonal Vectors problem is complete in the class of first-order model checking on sparse structures, under fine-grained reductions. In other words, the hardness of Orthogonal Vectors and the hardness of first-order model checking imply each other. This also gives us an improved algorithm for first-order model checking problems.

Among all first-order logic formulas in prenex normal form, we have reasons to believe that quantifier structures $\exists \ldots \exists \forall$ and $\forall \ldots \forall \exists$ may be the hardest in computational complexity: If the Nondeterministic version of the Strong Exponential Time Hypothesis is true, formulas of these forms are the only hard ones under the Strong Exponential Time Hypothesis.

We can add extensions to first-order logic to strengthen its expressive power. This work also studies the fine-grained complexity of first-order formulas with comparison on structures with total order, first-order formulas with transitive closure operations, first-order formulas of fixed quantifier rank, and first-order formulas of fixed variable complexity.

We also introduce a technique that can be used to reduce from sequential problems on graphs to parallel problems on sets, which can be applied to extending the Least Weight Subsequence problems from linear structures to some special classes of graphs.

Chapter 1

Introduction

1.1 Fine-Grained Complexity of Model Checking Problems

This dissertation presents results about the fine-grained complexity and algorithms about problems in polynomial time, that are describable by first-order logic or extensions of first-order logic. *Fine-grained complexity* is a relatively new sub-area within theoretical computer science that not only qualitatively classifies problems as "easy" or "hard", but (to the extent possible) pin-points their exact complexities. It aims to make complexity theory more relevant to algorithm design (and vice versa) by giving reductions that better preserve the times required for solving problems, and connecting algorithmic progress with complexity theory. While some of the key ideas can be traced back to parameterized algorithms and complexity ([FJ56, DF92]), studies of the exact complexity of NP-complete problems ([SHI90, IPZ98, JS99, IP99]), and algorithmic consequences of circuit lower bounds ([AW85, Yao82, LMN93, NW94, BFNW93, IKW02, KI04]), the full power of this approach has emerged only recently. This approach has given us new circuit lower bounds ([Wil13, Wil14b]), surprising algorithmic improvements using circuit lower bound techniques ([AWY15, Wil05, CW16, CIKK16]), and many new insights into the relative difficulty of substantially improving known algorithms for a variety of problems both within and beyond polynomial time.

There are now a wide variety of standard algorithmic problems where no significant improvements in algorithmic running time can be made without refuting one of a few conjectures about well-studied problems, such as the k-SUM problem [GO95], All Pairs Shortest Paths [WW10, AGW14, LWW18], SAT, or Orthogonal Vectors [AWW14, BI15, BCH16, Bri14, ABW15, BI15, BK15, AHWW16, MPS16, KPS17, AR18, ABDN18, BRS$^+$18]. As the field has grown, many fundamental relationships between problems have been discovered, making the graph of known results a somewhat tangled web of reductions.

Traditionally, complexity theory has been used to distinguish very hard problems, such as NP-complete problems, from relatively easy problems, such as those in P. However, over the past few decades, there has been progress in understanding the exact complexities of problems, both for very hard problems and those within P, under plausible assumptions.

Unfortunately, as our understanding of the relationship between the exact complexities of problems grows, so does the complexity of the web of known reductions and the number of distinct conjectures these results are based on. Ideally, we would like to show that many of these conjectures are in fact equivalent, or that all follow from some basic unifying hypothesis, thereby improving our understanding and simplifying the state of knowledge. For example, it would be nice to show that the 3-SUM conjecture or the APSP conjecture follows from SETH. A result like that would reduce the number of conjectures we rely on to explain the complexity of problems.

At the same time, many problems seem to be hard, but their hardness is not explained by any of the three most popular conjectures in fine-grained complexity, SETH, the 3-SUM conjecture and the APSP conjecture. Among these questions is if HITTINGSET can be solved in subquadratic time or if MAXFLOW has a linear time algorithm. For neither of these problems can we answer the question positively with an algorithm nor negatively with a conditional lower bound.

In traditional complexity, classes of problems are related to each other, and individual problems understood by identifying classes for which they are complete. In contrast, most of the results in fine-grained complexity were obtained on a problem-by-problem basis. One reason for this is that results in fine-grained complexity cut across traditional classes, with NP-complete

problems reducing to problems within P or even smaller classes. This raises the questions: is it possible to give a fine-grained complexity of classes of problems? Is the notion of completeness useful in fine-grained complexity?

We study the class of problems where each problem asks whether the input structure satisfies a fixed first-order formula. This class is natural both in terms of computational complexity, where it is the uniform version of AC_0, and in database theory, because these are the queries expressible in basic SQL [AHV95]. First-order logic can also express many polynomial time computable problems: ORTHOGONALVECTORS, k-ORTHOGONALVECTORS, k-CLIQUE, k-INDEPENDENTSET, k-DOMINATINGSET, etc. Not only were the likely complexities of the hardest problems (as a function of number of quantifiers) given, but in the second paper, a natural complete problem was identified, the orthogonal vectors problem (discussed below in more detail). The conclusion was that there were substantial improvements possible in the worst-case complexity of model checking for first-order properties if and only if the known Orthogonal Vectors algorithms can be substantially improved. Using a recent sub-polynomial improvement in OV algorithms by [AWY15], they obtained a similar improvement in model checking every first-order property.

In CNF-SAT problem, given a Boolean formula F in CNF form (conjunction of disjunctions of (possibly negated) variables), the goal is to determine whether there is an assignment of Boolean values to variables of F which makes F true. In k-CNF-SAT, every clause (disjunction) can have at most k literals. We refer to the following conjecture about complexity of solving CNF-SAT:

Strong Exponential Time Hypothesis (SETH)[1]: For every $\varepsilon > 0$, there exists a $k \geq 2$ so that k-CNF-SAT cannot be solved in time $O(2^{n(1-\varepsilon)})$.

The problem of deciding whether a structure satisfies a logical formula is called the model checking problem. It is well-studied in finite model theory. In relational databases, first-order model checking plays an important role, as first-order queries capture the expressibility of relational algebra. In contrast to the *combined complexity*, where the database and query are both given

[1] Some define SETH over randomized algorithms instead of deterministic ones

as input, the *data complexity* measures the running time when the query is fixed. The combined complexity of first-order queries is PSPACE-complete, but the data complexity is in LOGSPACE [Var82]. Moreover, these problems are also major topics in parameterized complexity theory. In [FG06], Flum and Grohe organize parameterized first-order model-checking problems (many of which are graph problems) into hierarchical classes based on their quantifier structures. Here, we study model checking from the fine-grained complexity perspective.

This dissertation talks about model checking problem where the logical formula (i.e., the query) is fixed by the problem. We use the term "first-order property" to refer to these problems.

First-order properties are also extensively studied in complexity, logic (especially finite model theory and theory of databases) and combinatorics. For example, the first zero-one law for random graphs was proved for first-order properties on finite models ([Fag76]), and Ajtai's lower bound for AC^0 [Ajt83] (proved independently by Furst, Saxe, and Sipser ([FSS84])) was motivated and stated as a result about inexpressibility in first-order logic.

There are many problems within P that are known to be SETH-hard, but few of them are first order property problems. And of the ones that are, they tend to have similar logical forms. For instance, k-DOMINATINGSET [BCH16] is definable by a $\forall^k\exists$ quantified formula; the GRAPHDIAMETER-2 problem and the BIPARTITEGRAPHDOMINATEDVERTEX problem [BCH16] are definable by $\forall\forall\exists$ quantified formulas. Here we study the relations between SETH-hardness and the logical structures of model checking problems. A result by Ryan Williams [Wil14a] explored the first-order graph properties on dense graphs, while here we look into sparse graphs whose input is a list of edges.

We define first-order properties on hypergraphs. The input is a many-sorted universe that we view as sets of vertices, together with a number of unary relations (node colors), and binary relations, viewed as different categories or colors of edges. The binary relations are not symmetric in general. We specify the problem to be solved by a first order sentence. Let φ be a first order

sentence in prenex normal form, with k quantifiers:

$$\varphi = Q_1 x_1 \in X_1, Q_2 x_2 \in X_2, \ldots Q_k x_k \in X_k \psi \tag{1.1}$$

or shortened as

$$\varphi = Q_1 x_1 Q_2 x_2 \ldots Q_k x_k \psi \tag{1.2}$$

where φ is a quantifier-free formula whose atoms are unary or binary predicates on $x_1, \ldots, x_k$.

An instance of the model checking problem of a formula φ with $k \geq 3$ quantifiers specifies sets $X_1, \ldots X_k$, where variable x_i is an element of set X_i, as well as all the unary and binary relations that occur in φ. We assume without loss of generality that the sets X_i are disjoint and that the domain of any predicate is restricted to one pair (X_i, X_j). We can always duplicate elements and adjust the corresponding relations accordingly. We also assume equality is one of the relations, so we can tell when $x_i = x_j$. To reformulate the problem as a graph problem, we view the sets $X_1, \ldots, X_k$ as the sets of nodes in a k-partite graph, and the binary predicates as (colored) edges, i.e. for some predicate P, if $P(x_i, x_j)$ is true then there is an edge between the nodes x_i and x_j. We refer to the k-partite graph with edges defined by predicate P as G_P, and the colored union of graphs defined on all predicates as G.

We assume that the input is given as follows: For each unary relation, we are given a Boolean vector indexed by the vertices saying whether the relation holds, and for each binary predicate, the list representation of the corresponding directed graph. We want to decide if φ is true for the input model.

Examples of this problem include k-CLIQUE, which is defined by

$$\varphi = \exists x_1 \ldots \exists x_k \bigwedge_{i,j \in \{1,\ldots,k\}, i \neq j} E(x_i, x_j) \tag{1.3}$$

k-DOMINATINGSET, defined by

$$\varphi = \exists x_1 \ldots \exists x_k \forall x_{k+1} \left(E(x_1, x_{k+1}) \vee \cdots \vee E(x_k, x_{k+1}) \right) \tag{1.4}$$

and GRAPHRADIUS2, defined by

$$\varphi = \exists x_1 \forall x_2 \exists x_3 \left(E(x_1, x_3) \wedge E(x_3, x_2) \right) \tag{1.5}$$

We let $n = \max_i |X_i|$ be the maximum size of the node parts, and m be the number of edges in the union of the graphs. The size is $n+m$, but for convenience, we will assume $m > n$ and use m as the input size.

1.2 Overview of the Dissertation

Section 1.3 gives the definitions of problems and classes used in this dissertation. Section 1.5 introduces the basic concepts and techniques of fine-grained complexity.

In Chapter 2, we introduce a new technique that provides reasons to believe that some problems may be strictly harder than some other problems. We show that under the Nondeterministic Strong Exponential Hypothesis, the hardest first-order property problems all have similar quantifier structures: either $\exists \ldots \exists \forall$, or $\forall \ldots \forall \exists$.

In Chapter 3, we prove that the well-studied Orthogonal Vectors problem is complete in the class of first-order property problems. In other words, improved algorithms for OV will imply better algorithms for all first-order property problems. This result shows that Orthogonal Vectors is a relatively hard problem. Even if the Strong Exponential Time Hypothesis is false, OV may remain hard if there exists a hard first-order property.

We give algorithms for every first-order property problem that improves this upper bound to $m^k/2^{\Theta(\sqrt{\log n})}$, i.e., an improvement by a factor more than any poly-log, but less than the polynomial required to refute SETH. Moreover, we show that further improvement is *equivalent* to improving

algorithms for sparse instances of the well-studied Orthogonal Vectors problem. Surprisingly, both results are obtained by showing completeness of the Sparse Orthogonal Vectors problem for the class of first-order properties under fine-grained reductions. To obtain improved algorithms, we apply the fast Orthogonal Vectors algorithm of [AWY15, CW16].

While fine-grained reductions (reductions that closely preserve the conjectured complexities of problems) have been used to relate the hardness of disparate specific problems both within P and beyond, this is the first such completeness result for a standard complexity class.

Chapter 4 studies extensions of the class of first-order model checking problems, and studies this class with more lenient parameterizations. We consider classes obtained by allowing function symbols; first-order on ordered structures; adding various notions of transitive closure operations; and stratifications of first-order properties by quantifier depth and variable complexity, rather than number of quantifiers. For some of these classes, OV is still a complete problem, in that significant improvement for the entire class is equivalent to significant improvement for OV algorithms. For these classes, we can also use the improved OV algorithm of [AWY15, CW16] to get moderate improvements on algorithms for the entire class. For other classes, we show that model checking becomes harder than for first-order, under well-studied conjectures such as SETH. For these classes, we show hardness follows from weaker assumptions than SETH.

Surprisingly, whether an extension increases the complexity of model checking seems independent of whether it increases the expressive power of the logic. For example, adding function symbols does not change which problems are expressible by first-order, but does increase the time for model checking under SETH. On the other hand, adding an ordering does not change the fine-grained complexity of model checking, although it increases the logic's expressive power.

Chapter 5 introduces a new technique that generalizes previously known subquadratic time reductions from linear structures to graphs. Least Weight Subsequence (LWS) is a class of highly sequential optimization problems with form $F(j) = \min_{i<j}[F(i) + c_{i,j}]$ [HL87]. They can be solved in quadratic time using dynamic programming, but it is not known whether these problems can be solved faster than $n^{2-o(1)}$ time. Surprisingly, each such problem is subquadratic time reducible to a

highly parallel, non-dynamic programming problem [KPS17]. In other words, if a "static" problem is faster than quadratic time, so is an LWS problem. For many instances of LWS, the sequential versions are equivalent to their static versions by subquadratic time reductions. The previous result applies to LWS on linear structures, and this chapter extends this result to LWS on paths in sparse graphs, the Least Weight Subpath (LWSP) problems. When the graph is a multitree (i.e. a DAG where any pair of vertices can have at most one path) or when the graph is a DAG whose underlying undirected graph has constant treewidth, we show that LWS on this graph is still subquadratically reducible to their corresponding static problems. For many instances, the graph versions are still equivalent to their static versions.

Moreover, this chapter shows that on these graphs, if we can decide a first-order property $\exists x \exists y P(x,y)$ in subquadratic time, where P is a quickly checkable property, then we can also in subquadratic time decide whether there are pairs x, y in the transitive closure of a DAG of the above types that satisfy $P(x,y)$, which is a considerably more expressive class of problems.

Appendix A lists some example problems in the classes of problems studied in this dissertation.

1.3 Definitions of Model Checking Problems and Classes

1.4 First-Order Property Problems

Next we will give the definitions and notations regarding the model checking problems.

Let φ be a fixed formula and let G be an input structure, the model checking problem for φ is to decide whether G satisfies φ. When φ is a first-order formula, we also call it a first-order property problem.

φ is a fixed formula without free variables (i.e. all variables are quantified by either $\exists$ or $\forall$). We use MC_φ to denote the model checking problem for formula φ. In this dissertation we usually use letter φ for formulas containing quantifiers, and use letter ψ for quantifier-free subformulas.

Letter Q is used to represent a quantifier, either $\exists$ or $\forall$.

The input structure has multiple variable domains and multiple relations. It can be considered as a hypergraph (represented by adjacency list): the elements of the structure correspond to vertices, and the relation tuples of the structure correspond to edges.

The *domain of a variable* is a fixed set of vertices so that a variable in φ can be assigned to any one of the vertices. The total number of vertices is n.

A *relation* is a fixed set of edges (or hyperedges) so that a binary (or t-ary) predicate in φ can correspond to one of the edges. The total number of edges is m. We only consider the case that $m \geq n$.

Because the number of edges is important in describing problem size, in the rest of this dissertation we define the *degree* of a vertex v to stand for the total number of edges the vertex is in.

let φ be a fixed first-order sentence containing free predicates of arbitrary constant arity (and no other free variables). For example, the k-Orthogonal Vectors (k-OV) problem can be expressed by a $(k+1)$-quantifier formula $\varphi = (\exists v_1 \in A_1)\ldots(\exists v_k \in A_k)(\forall i)\left[\bigvee_{j=1}^{k} \neg(v_j[i]=1)\right]$. The model-checking problem for φ, denoted by MC_φ, is deciding whether φ is true on a given input structure interpreting predicates in φ (e.g., given k sets of vectors, decide k-OV). We sometimes refer to structures as "hypergraphs" ("graphs" when all relations are unary or binary), and relations as edges or hyperedges. We use n to denote size of the universe of the structure and m the total number of tuples in all its relations (size of the structure). Many graph properties such as k-clique have natural first-order representations, and set problems such as Hitting Set are representable in first-order logic using a relation $R(u,S) \equiv (u \in S)$.

We propose the following conjecture on the hardness of model checking of first-order properties.

First-order property conjecture (FOPC): There is an integer $k \geq 2$, so that there is a $(k+1)$-quantifier first-order property that cannot be decided in $O(m^{k-\varepsilon})$ time, for any $\varepsilon > 0$.

Assumptions

The complexity is measured in the word RAM model with $O(\log n)$ bit words. The notation

$\tilde{O}$ notation is generally used for time complexity hiding sub-polynomial time factors. But in this dissertation we usually consider savings factors in running time that grow faster than polylogarithmic, so we still use the big-O notation but let it hide polylogarithmic factors.

In this dissertation, without loss of generality we make the following assumptions.

- Assume $m = n^{1+o(1)}$, because otherwise the $O(mn^{k-2})$ time baseline algorithm (Lemma 4.9.1) is better than the conjectured time $m^{k-o(1)}$.
- Assume that different variables are in different domains. (For instance, the universe of x is X, the universe of y is Y, etc.) In other words, a structure for a k-quantifier formula can be considered as a k-partite graph. However, in the case where transitive closure operations can be taken on a relation, we will assume both variables of the relation are in the same universe.
- We assume that for any tuple of elements, the value of a predicate on this tuple can be queried in constant time. Also, assume that the neighbors of any element v can be enumerated in time linear to the degree of v.

Let $R_1, \ldots, R_r$ be predicates of constant arities $a_1, \ldots, a_r$ (a vocabulary). A finite *structure* over the vocabulary $R_1, \ldots, R_r$ consists of a *universe* U of size n together with r lists, one for every R_i, of m_i tuples of elements from U on which R_i holds. Let $m = \sum_{i=1}^{r} m_i$; viewing the structure as a database, m is the total number of records in all tables (relations).

We loosely use the term *hypergraph* to denote an arbitrary structure; in this case, we refer to its universe as a set of vertices $V = \{v_1, \ldots, v_n\}$ and call tuples $(v_1, \ldots, v_{a_i})$ such that $R_i(v_1, \ldots, v_{a_i})$ holds hyperedges (labeled R_i). A set of all R_i-labeled hyperedges in a given hypergraph is denoted by E_{R_i} or just E_i; the structure is denoted by $G = (V, E_1, \ldots, E_r)$. Similarly, we use the term *graph* for structures with only unary and binary relations (edges); here, we mean edge-labeled vertex-labeled directed graphs with possible self-loops, as we allow multiple binary and unary relations and relations do not have to be symmetric. This allows us to use graph terminology such as a *degree* (the number of (hyper)edges containing a given vertex) or a neighbourhood of a vertex.

Let φ be a first-order sentence (i.e. formula without free first-order variables) containing predicates $R_1, \ldots, R_r$. Let k be the number of quantifiers in φ. Without changing k, we can write φ

in prenex form. The *model-checking problem* for a *first-order property* φ, MC_φ, is: given a structure (hypergraph) G, determine whether φ holds on G (denoted by $G \models \varphi$). We use notation FOP_k for the class of model checking for k-quantifier first-order formulas in prenex form, and $MC(Q_1 \dots Q_k)$ for the model checking for first-order prenex formulas with quantifier prefix $Q_1 \dots Q_k$, with a shortcut Q_i^c denoting c consecutive occurrences of Q (e.g. $MC(\exists^k \forall)$).

We assume that (hyper)graphs are given as a list of m (hyper)edges, with each hyperedge encoded by listing its elements. In the Word RAM model with $O(\log n)$ bit words, the size of an encoding of a hypergraph is $O(n+m)$ words, and an algorithm can access a hyperedge in constant time. With additional $O(m)$ time preprocessing, we can compute degrees and lists of incident edges for each vertex, and store them in a hash table for a constant-time look-up; edges incident to a vertex can then be listed in time proportional to its degree. We also assume that $m \geq n$, with every vertex incident to some edge, because the interesting instances are in this case. Moreover, we assume the (hyper)graph is k-partite where k is the number of variables in φ, so that each variable is selected from a distinct vertex set. From any (hyper)graph, the construction of this k-partite graph needs a linear time, linear space blowup preprocessing which creates at most k duplicates of the vertices and k^2 duplicates of the edges. Finally, we treat domains of quantifiers as disjoint sets forming a partition of the universe; any structure can be converted into this form with constant increase of the universe size. We also view predicates on different variable sets (e.g., $R(x_1,x_2)$ vs. $R(x_2,x_4)$ vs. $R(x_4,x_4)$) as different predicates, and partition corresponding edge sets appropriately.

The focus of this dissertation is on *sparse* structures, that is, the case when $m \leq O(n^{1+\gamma})$ for some γ such that $0 \leq \gamma < 1$. In particular, all E_i are sparse relations; we use the term *co-sparse* to refer to complements of sparse relations. We will usually measure complexity as a function of m. From the following baseline algorithm which will be proved in Section 3.9.1, the sparse assumption is without loss of generality.

Claim 1.4.1 (Baseline algorithm)**.** *For $k \geq 1$, FOP_{k+1} is solvable in time $O(n^{k-1}m)$.*

1.4.1 Types of First-Order Property Problems with Different Complexity Measures

***k*-Quantifier Problems:** Here φ has form $Q_1x_1 \ldots Q_kx_k\psi(x_1,\ldots,x_k)$. Without loss of generality, we assume φ is in prenex normal form. For example, the sparse OV problem can be represented by

$$\varphi_{OV} = \exists x \exists y \forall z(\neg One(x,z) \vee \neg One(y,z)),$$

where x, y are vectors and z is a coordinate of the vectors. $One(x,z)$ is true iff a vector x has a one on its z-th coordinate. The sparse k-OV problem is equivalent to

$$\varphi_{k-OV} = \exists x_1 \ldots \exists x_k \forall z \bigvee_{i=1}^{k}(\neg One(x_k,z)).$$

We use the notation FOP_k for the class of MC_φ where φ is a first-order formula with k quantifiers. This is the class of problems studied in Chapter 3.

Quantifier Rank *k* Problems: The *quantifier rank* of a formula is the maximum depth of nesting of its quantifiers. When we can reuse the same variable name in different scopes, for example,

$$\varphi = \exists x(\exists y(\exists z\psi_1 \wedge \forall z\psi_2) \wedge \forall y(\exists z\psi_3 \vee \forall z\psi_4))$$

has only three variable names x, y, z, but they represent different variables in different scopes of the formula. The above formula is of quantifier rank 3. The property that graph vertices s, t has a path of length ℓ can be represented by a formula with $\ell - 1$ existential quantifiers, but using the technique of Savitch's Theorem, the quantifier rank can be only $\log \ell$.

We use the notation $\mathsf{FOP}_{\mathrm{qr}=k}$ for the class of MC_φ where φ is a FO formula with quantifier rank k. $\mathsf{FOP}_{\mathrm{qr}=k}$ contains FOP_k. It can be solved in time $O(mn^{k-2})$ for any $k \geq 2$ (Lemma 4.9.1).

This dissertation shows that, even though a formula with quantifier rank k may have more than k variables when converted to prenex normal form, but the following theorem shows that even if it seems more powerful, it is reducible to quantifier number k problems.

This theorem will be proved in Appendix 4.3.

Variable Complexity k Problems: If we do not bound the quantifier rank of φ, it will have even more expressive power, for example, a formula of form

$$\varphi = \exists x \exists y (R(x,y) \wedge \exists x (R(y,x) \wedge \exists y \exists x (R(x,y) \wedge \dots)))$$

can represent a path of any constant length using only 2 variables.

A formula with k different variable names is referred to as of *variable complexity* k. A formula is of variable complexity k iff it can be computed by a straightline program, each line has at most k distinct variables (even if written in prenex form). That is, it's equivalent to the result of a sequence of first-order queries of form $R_i = \{(x_1,\dots,x_a) \mid \varphi_i(x_1,\dots,x_a)\}$ where each R_i is an intermediate relation of arity a ($0 \leq a \leq k$), and each φ_i is a first-order formula with at most k variables, including $x_1,\dots,x_a$, which appear as free variables in φ_i.

When the variable complexity is 3, if in each line of the corresponding straightline program φ, there is at most one occurrence of an intermediate binary relation computed in a previous line, then φ can be solved in $O(mn)$ time for sparse graphs, which will be shown in Appendix 4.10. In this case we call this variable complexity 3 problem *weakly succinct*. Appendix 4.10 gives another example which is in matrix multiplication time but not known to have $O(mn)$ algorithms.

We use the notation $\mathsf{FOP}_{\mathrm{vc}=k}$ for the class of MC_φ where φ is a FO formula with variable complexity k. The class of weakly succinct problems in $\mathsf{FOP}_{\mathrm{vc}=k}$ contains $\mathsf{FOP}_{\mathrm{qr}=k}$ (Each line of the straightline program creates a new intermediate relation on x,y by an intermediate relation on x,y and some original relations on some z. We will not elaborate the details here).

The following theorem shows that the SETH of constant depth circuit implies the quadratic-time hardness of $\mathsf{FOP}_{\mathrm{vc}=3}$.

1.4.2 Types of Problems Definable by Extensions of First-Order Logic

***k*-Quantifier with Function Symbols:** In first-order formulas, sometimes we allow a list of functions symbols $f_1,\dots,f_c$. Here we only consider unary function symbols, because the description

of higher arity functions takes n^2 space in the input. To simulate higher arities, we could increase the universe to a Cartesian power and then have unary functions on this product space. Each function symbol f_i maps an element to another element. Predicates can be taken on function symbols, e.g. $P(f_1(x), f_2(y))$.

For example, the problem "checking if a graph coloring satisfies the condition that no pairs of vertices of distance 2 have the same coloring" can be written as $\forall x \forall y(\exists z(E(x,z) \wedge E(z,y)) \rightarrow \mathit{Diff}(c(x), c(y)))$, where function $c(x)$ maps vertex x to a color c, and predicate *Diff* means two colors are different.

We use the notation FOPF_k for the class of MC_φ where φ is a k-quantifier FO formula with function symbols.

While we can simulate any function by a relation coding the graph of the function, $R_f(x,y) \iff f(x) = y$, to express, for example that $f(x_1) = f(x_2)$ we would need to write $\exists y, R_f(x_1,y) \wedge R_f(x_2,y)$. So the number of quantifiers in the translated forumlas would increase, possibly up to the number of function symbols appearing in the original. However, there is still a trivial $O(n^k)$ algorithm for model checking a k-quantifier formula with functions. So, assuming the OV Conjecture, the complexity grows by at most a linear amount over that for first-order without functions.

We show that this increase is necessary. Compared to the linear time baseline algorithm for the model checking of 2 quantifier formulas, when we introduce function symbols, a 2 quantifier formula may require quadratic time to solve.

First-Order on Ordered Structures: If all the elements in the domain of some variable have a total order so that for any two elements a, b it may be $a > b$, $a < b$ or $a = b$, then the comparison relation on two elements is a dense relation. But unlike general dense relations, it can be represented succinctly in the input by $O(n)$ space.

An example is that we have a log of communications in a network with events where a processor receives or sends messages, and we want to verify that every message sent is later

received. This can be expressed by $\forall e \exists e'((e < e') \wedge \mathit{SameSender}(e,e') \wedge \mathit{SameReceiver}(e,e') \wedge \mathit{SameMessage}(e,e'))$, where e and e' are two log entries.

We use the notation $\mathsf{FOP}_k(\leq)$ for the class of MC_φ where φ is a k-quantifier FO formula with comparison predicates. $\mathsf{FOP}_k(\leq)$ contains FOP_k. It can be solved in time $O(mn^{k-2})$ for any $k \geq 2$ (Lemma 4.9.1).

We will consider the case where elements are given a total pre-ordering, and there are three predicates expressing that an element is greater than, less than, or equivalent to another element in the ordering. The comparison relation is an implicit dense relation but can be represented in $O(n)$ space in the input, by giving a table indexed by element, giving the element's rank within the ordering, with equivalent elements given the same rank. (If we were not given this table, we could use any sorting algorithm to construct it in $O(n \log n)$ time.) Using this table, we can list the elements by this ordering in time $O(n)$, and given any two elements, we can compare them in time $O(1)$. The following theorem shows that adding comparison to first-order logic does not increase the fine-grained complexity because it is equivalent to first-order properties without comparison.

First-order with Transitive Closure: The transitive closure of a sparse relation may be a dense relation. The comparison relation is a special case, which is the transitive closure of the successor relation. We use TC_R to denote the transitive closure of relation R.

OV can be expressed by a 2-quantifier formula with transitive closure operation: we connect each vector u to each coordinate i iff $\mathit{One}(u,i)$, and connect each coordinate i to each vector v iff $\mathit{One}(v,i)$. Then u is reachable to v iff u is not orthogonal to v.

We use the notation $\mathsf{FOP}_k(\mathrm{TC})$ for the class of MC_φ where φ is a k-quantifier FO formula allowing transitive closure operations. $\mathsf{FOP}_k(\mathrm{TC})$ contains $\mathsf{FOP}_k(\leq)$.

For the problems of $\mathsf{FOP}_k(\mathrm{TC})$ where the transitive closure operations are only allowed on symmetric input relations, we use the notation $\mathsf{FOP}_k(\mathrm{TC}_{\mathrm{sym}})$ to donate the class of these problems.

Consider the model checking of a first-order formula with transitive closures, where the transitive closure operation can only be taken on symmetric input relations. In this case $TC_R(x,y)$

is true iff x and y are in the same connected component by edges of undirected edge set R. Thus the formula can have binary predicates about whether two variables are in the same connected component or not. Note that there can be more than one symmetric relations that the transitive closure operation can be taken on. We we use the notation $\mathsf{FOP}_k(\mathrm{TC}_{\mathrm{sym}})$ to donate the class of these problems.

1.5 Fine-Grained Complexity Preliminaries

1.5.1 Fine-Grained Reductions

To establish the relationship between complexities of different problems, we use the notion of *fine-grained reductions* as defined in [WW10]. Fine-grained reductions are defined with the motivation to control the exact complexity of the reducibility. For this purpose, we consider languages together with their *presumed* or *conjectured* complexities. These reductions establish conditional hardness results of the form "If one problem has substantially faster algorithms, so does another problem". We will also use *exact complexity reductions* (see definition 1.5.2), which strengthen the above claim to "if one problem has algorithms improved by a factor $s(m)$, then another problem can be improved by a factor $s^c(m)$" for some constant c. (Note that some fine-grained reductions already have this property.) The underlying computational model is the Word RAM with $O(\log n)$ bit words.

Thus, if L_2 has an algorithm substantially faster than T_2, L_1 can be solved substantially faster than T_1. [2]

Below we give the formal definition of Fine-Grained Turing Reduction.

[2] In almost all fine-grained reductions, $T_1 \geq T_2$, that is, we usually reduce from harder problems to easier problems, which may seem counter-intuitive. A harder problem L_1 can be reduced to a easier problem L_2 with $T_1 > T_2$ in two ways: by making multiple calls to an algorithm solving L_2 and/or by blowing up the size of the L_2 instance (e.g., the reduction from CNF-SAT to OV [Wil05]). All reductions from higher complexity to lower complexity problems in this dissertation belong to the first type.

Actually, it is harder to fine-grained reduce from a problem with lower time complexity to a problem with higher time complexity (e.g., prove that $(MC(k), m^{k-1}) \leq_{FGR} (MC(k+1), m^k)$), because this direction often needs creating instances with size much smaller than the original instance size.

We use the pair (L,T) to denote a language together with its time complexity T. Intuitively, if (L_1,T_1) fine-grained reduces to (L_2,T_2), then any constant savings in the exponent of the time complexity of L_2 implies some constant savings in the exponent of the time complexity of L_1.

Definition 1.5.1 (Fine-Grained Reductions ($\leq_{FGR}$))**.** Let L_1 and L_2 be languages, and let T_1 and T_2 be time bounds. We say that (L_1,T_1) *fine-grained reduces* to (L_2,T_2) (denoted $(L_1,T_1) \leq_{FGR} (L_2,T_2)$) if for all $\varepsilon > 0$, there is a $\delta > 0$ and a deterministic Turing reduction $\mathcal{M}^{L_2}$ from L_1 to L_2 satisfying the following conditions.

(a) The time complexity of the Turing reduction without counting the oracle calls is bounded by $T_1^{1-\delta}$.

$$\mathsf{TIME}[\mathcal{M}] \leq T_1^{1-\delta} \tag{1.6}$$

(b) Let $\tilde{Q}(\mathcal{M},x)$ denote the set of queries made by $\mathcal{M}$ to the oracle on an input x of length n. The query lengths obey the following time bound.

$$\sum_{q \in \tilde{Q}(\mathcal{M},x)} (T_2(|q|))^{1-\varepsilon} \leq (T_1(n))^{1-\delta}$$

If a fine-grained reduction exists from (L_1,T_1) to (L_2,T_2), algorithmic savings for L_2 can be transferred to L_1. The definition gives us exactly what is needed to establish savings for L_1 by simulating the machine $\mathcal{M}^{L_2}$ using the faster algorithm for L_2. The role of each parameter in the definition of fine-grained reducibility makes this clear.

T_1: The presumed time to decide L_1, usually given by a trivial algorithm.

T_2: The presumed time to decide L_2.

ε: Any savings (assumed or real) on computing L_2.

δ: The savings (as a function of ε) that can be obtained over T_1 when deciding L_1 by reducing to L_2.

It is easy to verify that fine-grained reductions, just like polynomial time reductions, can be composed.

Lemma 1.5.1 (Fine-grained reductions are closed under composition)**.** *Let* $(A,T_A) \leq_{FGR} (B,T_B)$ *and* $(B,T_B) \leq_{FGR} (C,T_C)$*. It then follows* $(A,T_A) \leq_{FGR} (C,T_C)$*.*

To simplify transferring algorithmic results, we define a stricter variant of fine-grained reductions, which we call *exact reductions*. These reductions satisfy a stronger reducibility notion.

Definition 1.5.2. (Exact complexity reduction ($\leq_{EC}$))
Let L_1 and L_2 be languages and let T_1, T_2 denote time bounds. Then $(L_1,T_1) \leq_{EC} (L_2,T_2)$ if there exists an algorithm A_{L_1} for L_1 running in time $T_1(n)$ on inputs of length n, making calls to oracle of L_2 with query lengths $n_1,\ldots,n_q$, where q is the number of calls and $\sum_{i=1}^{q} T_2(n_i) \leq T_1(n)$.

That is, if L_2 is solvable in time $T_2(n)$, then A_{L_1} solves L_1 in time $T_1(n)$.

The same fine-grained reductions also transfer nondeterministic and co-nondeterministic savings. In particular, the existence of both a fast nondeterministic and co-nondeterministic algorithm for L_2 implies that there are also fast nondeterministic and co-nondeterministic algorithms for L_1.

Lemma 1.5.2 (Fine-grained reductions transfer savings for $(\mathsf{N}\cap\mathsf{coN})\mathsf{TIME}$)**.** *Let* $(L_1,T_1) \leq_{FGR} (L_2,T_2)$*, and* $L_2 \in (\mathsf{N}\cap\mathsf{coN})\mathsf{TIME}[T_2(n)^{1-\varepsilon}]$ *for some* $\varepsilon > 0$*. Then there exists a* $\delta > 0$ *such that*

$$L_1 \in (\mathsf{N}\cap\mathsf{coN})\mathsf{TIME}[T_1(n)^{1-\delta}]$$

Proof. Both the nondeterministic algorithm for L_1 and $\neg L_1$ follow the same outline. We simulate the deterministic Turing reduction $\mathcal{M}^{L_2}$. For the oracle calls, we nondeterministically guess for instances $q \in \tilde{Q}(\mathcal{M},x)$ if $q \in L_2$ or not and simulate either the nondeterministic machine for L_2 or $\neg L_2$. We can therefore simulate each oracle call q in $\mathsf{NTIME}[T_2(|q|)^{1-\varepsilon}]$ for some $\varepsilon > 0$. By the properties of the fine-grained reduction we have $\mathsf{TIME}[\mathcal{M}] \leq T_1^{1-\delta}$ and $\sum_{q\in\tilde{Q}(\mathcal{M},x)}(T_2(|q|))^{1-\varepsilon} \leq (T_1(n))^{1-\delta}$,

and hence $L_1 \in \mathsf{NTIME}[T_1(n)^{1-\delta}]$. Similarly we have $L_1 \in \mathsf{coNTIME}[T_1(n)^{1-\delta}]$ as we can negate the output of the fine-grained reduction $\mathcal{M}^{L_2}$. □

We define nondeterministic fine-grained reductions as follows.

Definition 1.5.3 (Nondeterministic Fine-Grained Reductions)**.** Let L_1 and L_2 be languages, and let T_1 and T_2 be time bounds. We say that (L_1, T_1) *nondeterministically fine-grained reduces* to (L_2, T_2) if for all $\varepsilon > 0$, there is a $\delta > 0$ and two nondeterministic Turing reductions $\mathcal{M}_1^{L_2}$ and $\mathcal{M}_2^{L_2}$ satisfying the following conditions.

(a) For all $x \in L_1$, then there is a y with $|y| \leq T_1^{1-\delta}$ such that $\mathcal{M}_1(x,y)$ accepts.

(b) For all $x \notin L_1$, then there is a y with $|y| \leq T_1^{1-\delta}$ such that $\mathcal{M}_2(x,y)$ accepts.

(c) The time complexity of both Turing reduction without counting the oracle calls is bounded by $T_1^{1-\delta}$, that is, for $c \in \{1,2\}$

$$\mathsf{TIME}[\mathcal{M}_c] \leq T_1^{1-\delta} \tag{1.7}$$

(d) Let $\tilde{Q}(\mathcal{M},x)$ denote the set of queries made by $\mathcal{M}$ to the oracle on an input x of length n. The query lengths obey the following time bound for $c \in \{1,2\}$.

$$\sum_{q \in \tilde{Q}(\mathcal{M}_c,x)} (T_2(|q|))^{1-\varepsilon} \leq (T_1(n))^{1-\delta}$$

To prove Lemma 1.5.2 under nondeterministic reductions we can nondeterministically guess y and simulate $\mathcal{M}_1$ and $\mathcal{M}_2$ similar to the deterministic case to get nondeterministic Turing machines for L_1 and $\neq L_1$.

1.5.2 Conjectures

The Orthogonal Vectors (OV) problem is a well-studied problem in the field of fine-grained complexity. Its hardness is implied by the hardness of CNF-SAT, and implies the hardness of many

problems (A list of hardness results under OV conjecture is compiled in [Wil18]). It is defined as follows: Given a set A of n Boolean vectors of dimension d, and must decide if there are $u, v \in A$ such that u and v are orthogonal, i.e., $u[i] \cdot v[i] = 0$ for all indices $i \in \{1, \ldots, d\}$. Another (equivalent) version is to decide with two sets A and B of Boolean vectors whether there are $u \in A$ and $v \in B$ so that u and v are orthogonal. A naïve algorithm for OV runs in time $O(n^2 d)$, and the best known algorithm runs in $n^{2-\Omega(1/\log(d/\log n))}$ [AWY15, CW16].

In this dissertation we introduce a version of OV we call the *Sparse Orthogonal Vectors (Sparse OV) problem*, where the input is a list of m vector-index pairs (v, i) for each $v[i] = 1$ (corresponding to the adjacency list representation of graphs) and complexity is measured in terms of m; we usually consider $m = O(n^{1+\gamma})$ for some $0 \leq \gamma < 1$. The popular hardness conjectures on OV restrict the dimension d to be between $\omega(\log n)$ (low dimension) and $n^{o(1)}$ (moderate dimension); however in Sparse OV we do not restrict d.

We thus identify three versions of Orthogonal Vector Conjecture, based on the size of the dimension d. In all three conjectures the complexity is measured in the word RAM model with $O(\log n)$ bit words.

Low-dimension OVC (LDOVC): For all $\varepsilon > 0$, there is no $O(n^{2-\varepsilon})$ time algorithm for OV with dimension $d = \omega(\log n)$.

Moderate-dimension OVC (MDOVC): For all $\varepsilon > 0$, there is no $O(n^{2-\varepsilon}\mathsf{poly}(d))$ time algorithm that solves OV with dimension d.

Sparse OVC (SOVC): For all $\varepsilon > 0$, there is no $O(m^{2-\varepsilon})$ time algorithm for Sparse OV where m is the total Hamming weight of input vectors.

OV can be extended to the k-OV problem for any integer $k \geq 2$: given k sets $A_1, \ldots, A_k$ of Boolean vectors, determine if there are k different vectors $v_1 \in A_1, \ldots, v_k \in A_k$ so that for all indices i, $\prod_{j=1}^{k} v_j[i] = 0$ (that is, their inner product is 0). We naturally define a sparse version of k-OV similar to Sparse OV, where all ones in the vectors are given in a list.

Strong Exponential Time Hypothesis(SETH) for CNF-SAT For all $\varepsilon > 0$, there exists a k so

that k-CNF-SAT cannot be solved in time $O(2^{n(1-\varepsilon)})$. [IPZ98]

Strong Exponential Time Hypothesis(SETH) for circuit class C For all $\varepsilon > 0$, the satisfiability of C cannot be solved in time $O(2^{n(1-\varepsilon)})$. [AHWW16] For each depth d, the SETH for depth-d circuit forms a hierarchy of hardness. It is analogous to the W-hierarchy in parameterized complexity theory [DF95].

Nondeterministic Strong Exponential Time Hypotheses For all $\varepsilon > 0$, there exists a k so that k-SAT is not in $\mathsf{coNTIME}[2^{n(1-\varepsilon)}]$. [CGI+16]

Low-dimension OVC (LDOVC): For all $\varepsilon > 0$, there is no $O(n^{2-\varepsilon})$ time algorithm for OV with dimension $d = \omega(\log n)$.

Moderate-dimension OVC (MDOVC): For all $\varepsilon > 0$, there is no $O(n^{2-\varepsilon}\mathsf{poly}(d))$ time algorithm that solves OV with dimension d. [GIKW17]

Sparse OVC (SOVC): For all $\varepsilon > 0$, there is no $O(m^{2-\varepsilon})$ time algorithm for Sparse OV where m is the total Hamming weight of input vectors. [GIKW17]

First-order property conjecture (FOPC): There is an integer $k \geq 2$, so that there is a $(k+1)$-quantifier first-order property that cannot be decided in $O(m^{k-\varepsilon})$ time, for any $\varepsilon > 0$. [GIKW17]

Hitting Set Conjecture $\forall \varepsilon > 0$ there is no $O(n^{2-\varepsilon})$ time algorithm for the Hitting Set problem with set sizes bounded by $d = \omega(\log n)$. It implies LDOVC; subquadratic approximation algorithms for Diameter-2 and Radius-2 would respectively refute the LDOVC and the Hitting Set Conjecture. [AWW16]

It is known that SETH implies LDOVC[Wil05]. Because MDOVC is a weakening of LDOVC, it follows from the latter.[3] Like LDOVC, MDOVC also implies the hardness of problems

[3] Although dimension d is not restricted, we call it "moderate dimension" because such an algorithm only improves on the naive algorithm if $d = n^{O(\varepsilon)}$.

including Edit Distance, LCS, etc. Here we further show that MDOVC and SOVC are equivalent (see Lemma 3.1.1).

The SETH of CNF-SAT implies LDOVC [Wil05], and LDOVC implies MDOVC because low-dimension OV is a special case of moderate-dimension OV. MDOVC, SOVC and FOPC are equivalent [GI19] (Will be shown in Chapter 3). The SETH of depth d circuits, where d is a constant greater than 2, is weaker than the SETH of CNF-SAT.

1.5.3 Basic Reduction Techniques

Quantifier-Eliminating Self reduction

A useful reduction is that $(\mathsf{FOP}_{k+1}, n \cdot T(m,n)) \leq_{\mathrm{EC}} (\mathsf{FOP}_k, T(m,n))$, where $T(m,n)$ is the running time on m edges and n vertices. This is because we can exhaustively search the outermost quantified variable, and use the value as a constant in the rest of the formula, thus reducing it to n instances of the model checking for k-quantifier formulas. The same technique can also be applied in the reductions for formulas with comparisons, and formulas of quantifier rank k.

Split and List

The "Split and List" technique was used in the reduction from CNF-SAT to Orthogonal Vectors.

Let φ be a CNF with n variables and m clauses. We partition the variables into two sets, each of size $n/2$. For each set, there are $2^{n/2}$ possible assignments to the $n/2$ variables. Create two sets A, B, each containing $2^{n/2}$ boolean vectors of length m. Each boolean vector corresponds to an assignment to $n/2$ variables. If an assignment does not satisfy the i-th clause, we let the i-th bit of the corresponding variable be 1, otherwise be 0. Thus, there exist assignment α for the first half of the variables and β for the second half of the variables that together satisfying φ iff for all clauses, either α satisfies it or β satisfies it, iff either the vector $v_\alpha \in A$ or $v_\beta \in B$ has a zero on all coordinates, iff v_α and v_β are a pair of orthogonal vectors.

When $m = cn$, if Orthogonal Vectors of dimension $d = cn$ is in time $O(n^{2-\varepsilon})$, then CNF-SAT

is in time $O(2^{n/2^{2-\varepsilon}}) = O(2^{n(1-\varepsilon/2)})$.

This technique will be used in reducing the satisfiability of constant depth circuits to model checking for formulas of variable complexity 3, in Chapter 2.

High-Degree Low-Degree Trick

The High-Degree Low Degree Trick is a commonly used trick. For example, it was used in proving Triangle Detection in sparse graphs is in time $O(m^{3/2})$.

We split the set of vertices into two parts: those with degree greater than $\sqrt{m}$, and those with degree less than $\sqrt{m}$.

For the vertices of degree greater than $\sqrt{m}$, there can be at most $O(\sqrt{(}m))$ of them, for otherwise the total number of edges would exceed m. So in $O(\sqrt{m})$ time we can enumerate the high degree vertices, and then using $O(m)$ time to enumerating the opposite edge, we can find any triangle containing a large degree vertex.

For the small degree vertices, we use $O(m)$ time to enumerate all edges containing such a vertice, and then use $O(\sqrt{m})$ time to enumerate all the neighbors of the current vertex to find the third point of the triangle, if there is one.

This technique will be used in the algorithm for $\exists\exists\exists$ and $\forall\exists\exists$ problems in Chapter 1, and the reduction algorithm for quantifier-rank 3 problems in Chapter 2.

Grouping reduction

The *grouping-reduction technique* is another useful reduction technique, which was introduced in [AWY15], that reduces the Batch OV problem to OV, and in [AWW16], that reduces Hitting Set to OV. In [GIKW17] it is used on the model checking on sparse structures. The reduction can be generalized to get the following statement: Assume the model checking for $\varphi = \exists x_1 \ldots \exists x_k$ $P(x_1,\ldots,x_k)$ can be decided in time $O(m^k/s(m))$ for some savings factor s, where P is a property on $x_1,\ldots,x_k$ that can be decided in time linear to the total degree of $x_1,\ldots,x_k$. Then we can list all x such that $\exists x_2 \ldots \exists x_k P(x_1,\ldots,x_k)$ can be decided in time $O(m^k/s(\mathsf{poly}(m)))$.

The idea is as follows: Pick a threshold size g, which is usually a polynomial of m. For

elements of weight greater than g, we enumerate all of them, and for each of them we run a $O(m^{k-1})$ algorithm to decide if P holds on x by doing exhaustive search on all other variables. So the total time is $O(m/g) \cdot O(m^{k-1}) = O(m^k/g)$.

Next, partition each of the domain of $x_1, \ldots, x_k$ into groups so that a group has total weight at most g, and there are $O(m/g)$ groups for each of $x_1, \ldots, x_k$. Every time we take a k-tuple of groups, and query φ on this smaller instance. As long as the query returns true, we can find a satisfying x_1 in $O(\log g)$ queries. On finding a satisfying x_1, we mark this x_1 and remove this x_1 from its group. Continue this process until either all x_1 are marked or no combination of groups satisfies φ. There are at most $O(m \log g + (m/g)^k)$ calls to the oracle of φ. The running time is $O(m \log g \cdot m \cdot (m/g)^k \cdot g^k/s(g)) = O(\log g \cdot m^k/s(g))$.

Thus the final running time is $O(m^k/g + \log g \cdot m^k/s(g))$. The logarithmic factor can be omitted if function s grows much faster than polylog functions.

Chapter 2

Consequences Under the Nondeterministic Strong Exponential Time Hypothesis

2.1 Introducing NSETH

The Strong Exponential Time Hypothesis conjectures that k-CNF-SAT cannot be solved efficiently by a deterministic algorithm, which can be considered as a fine-grained version of $\mathsf{P} \neq \mathsf{NP}$. If k-CNF-SAT is not only hard to solve using deterministic algorithms, but also by co-nondeterministic algorithms, then we get a fine-grained version of $\mathsf{P} \neq \mathsf{NP}$. This chapter introduces this new hardness conjecture, which we call the *Nondeterministic Strong Exponential Time Hypothesis* (NSETH). If NSETH is true, the hardness of problems under different popular conjectures is unlikely to be unified.

If any problem in $\mathsf{NP} \cap \mathsf{coNP}$ can be shown to be NP-complete, then $\mathsf{NP} = \mathsf{coNP}$. Hence, assuming $\mathsf{NP} \neq \mathsf{coNP}$, placing a problem L in $\mathsf{NP} \cap \mathsf{coNP}$ means that L cannot be NP-complete (or coNP-complete) and there is no polynomial time reduction from an NP-complete problem like 3-SAT to L. Similarly, by considering the nondeterministic and co-nondeterministic complexities of problems on a more fine-grained level, we deduce non-reducibility results based on NSETH.

We define NSETH formally as follows.

Definition 2.1.1 (Nondeterministic Strong Exponential Time Hypothesis (NSETH))**.** For every $\varepsilon > 0$, there exists a k so that k-SAT is not in $\mathsf{coNTIME}[2^{n(1-\varepsilon)}]$.

Equivalently, the k-TAUT problem, the tautology problem on k-DNF formulas, is not in $\mathsf{NTIME}[2^{n(1-\varepsilon)}]$ for sufficiently large k.

We feel that NSETH is plausible for many of the same reasons as SETH. We can think of NSETH as a statement about proof systems. Just as many algorithmic techniques have been developed for k-SAT, all of which approach exhaustive search for large k, many proof systems have been considered for k-TAUT, and none have been shown to have significantly less than 2^n complexity for large k. In fact, the tree-like ([PI00]) and regular resolution ([BI13]) proof systems have been proved to require such sizes. Moreover, we observe that results of [JMV15] that obtain circuit lower bounds assuming SETH is false yield the same bounds assuming that NSETH is false. So disproving NSETH would be both a breakthrough in proof complexity and in circuit complexity.

We also consider the natural nondeterministic variant of ETH.

Definition 2.1.2 (Nondeterministic Exponential Time Hypothesis (ETH))**.** The 3-SAT problem on n variables requires is not in $\mathsf{coNTIME}[2^{\varepsilon n}]$ for some $\varepsilon > 0$.

In [CGI+16], we show non-reducibility results for the following problems and time bounds.

- HITTINGSET for sets of total size m and time $T(m) = m^{1+\gamma}$ is not SETH-hard for any $\gamma > 0$, and no problem that is SETH-hard fine-grained reduces to HITTINGSET for any such time complexity.
- 3-SUM for $T(n) = n^{1.5+\gamma}$ is not SETH-hard for any $\gamma > 0$.
- MAXFLOW, minimum cost MAXFLOW, and maximum cardinality matching on a graph with m edges and $T(m) = m^{1+\gamma}$ are not SETH-hard.
- APSP on a graph with n vertices and $T(n) = n^{\frac{3+\omega}{2}+\gamma}$ is not SETH-hard.

All the results above are assuming NSETH and under deterministic reductions.

2.1.1 Reasons that NSETH Is Hard to Refute

SETH is an interesting hypothesis because both ¬SETH and SETH have interesting consequences that seem difficult to prove unconditionally. In this chapter, we show that the same proofs that show "¬SETH implies circuit lower bounds" can be applied to ¬NSETH as well. This is evidence that NSETH will be hard to refute.

Algorithms for CKT-SAT or CKT-TAUT imply circuit lower bounds (see [Wil13] and [Wil14b]). For some restricted circuit classes $\mathcal{C}$, we can reduce satisfiability or tautology of $\mathcal{C}$-circuits to k-SAT or k-TAUT by decomposing $\mathcal{C}$ circuits into a "big OR" of CNF formulas. Thus, both ¬SETH and ¬NSETH imply faster $\mathcal{C}$-circuit analysis algorithms (tautology or satisfiability) for these classes, which imply lower bounds.

The proofs of [JMV15] optimize the reduction of arbitrary nondeterministic time languages to 3-SAT to obtain new "failure of a hardness hypothesis about k-SAT implies circuit lower bounds" results for a variety of circuit classes. The followin is implicit in their work:

Theorem 1. *We have the following implications from failure of a k-TAUT hardness hypothesis to circuit lower bounds for restricted classes:*

1. *If the nondeterministic exponential time hypothesis (*NETH*) is false; i.e., for every $\varepsilon > 0$, 3-TAUT is in time $2^{\varepsilon n}$, then $\exists f \in \mathsf{E}^{\mathsf{NP}}$ such that f does not have linear-size circuits.*

2. *If the nondeterministic strong exponential time hypothesis (*NSETH*) is false; i.e., there is a $\delta < 1$ such that for every k, k-TAUT is in time $2^{\delta n}$, then $\exists f \in \mathsf{E}^{\mathsf{NP}}$ such that f does not have linear-size series-parallel circuits.*

3. *If there is $\alpha > 0$ such that n^{α}-TAUT is in time $2^{n-\omega(n/\log\log n)}$, then $\exists f \in \mathsf{E}^{\mathsf{NP}}$ such that f does not have linear-size log-depth circuits.*

The theorem is proved in the full version of [CGI+16]. Since (by item 2 above) refuting NSETH would give nontrivial circuit lower bounds, it is unlikely to be easy to refute.

2.1.2 Hardness of Reducibility under NSETH

How could we show that one language is not reducible to another language? There is an ever-growing web of problems, hypotheses, and reductions that reflect the fine-grained complexity approach to explaining hardness. Could this structure collapse into a radically simpler graph, with just a few equivalence classes? If we assume NSETH, probably not as much as one might hope.

We can broadly categorize computational problems into two sets. In the first category, the deterministic time complexity is higher than both the nondeterministic and co-nondeterministic time complexity. In the second category, at least one of nondeterminism or co-nondeterminism does not help in solving the problem more efficiently. Lemma 1.5.2 shows that savings in $(\mathsf{N} \cap \mathsf{coN})\mathsf{TIME}$ are preserved under deterministic fine-grained reductions. As a result, we can rule out tight reductions from a problem that is hard using nondeterminism or co-nondeterminism to a problem that is easy in $(\mathsf{N} \cap \mathsf{coN})\mathsf{TIME}$.

If NSETH holds, then k-SAT is in the category of problems that do not benefit fron co-nondeterminism. So, any problem that is SETH-hard under deterministic reductions also falls into this category.

In this chapter we explore problems that do benefit from $(\mathsf{N} \cap \mathsf{coN})\mathsf{TIME}$, i.e. we give nondeterministic algorithms that are faster than their presumed deterministic time complexities. This rules out deterministic fine-grained reductions from CNFSAT to these problems with their presumed time complexities. As a consequence, it is not possible to show that these problems are SETH-hard using a deterministic reduction.

As an immediate consequence of Lemma 1.5.2 we get a way to show non-reducibility assuming NSETH.

Corollary 2.1.1. *Assuming* NSETH*, for any problem L and time bound T, if*

$$L \in (\mathsf{N} \cap \mathsf{coN})\mathsf{TIME}[T(n)^{1-\delta}]$$

for some $\delta > 0$, then L is not SETH*-hard under deterministic reductions at time T.*

While Lemma 1.5.2 and Corollary 2.1.1 are formulated with respect to deterministic fine-grained reductions, we can extend the result to nondeterministic and zero-error reductions. On the other hand, the results do not extend to randomized reductions.

Theorem 2 (NSETH implies no reduction from CNFSAT)**.** *If* NSETH *and* $C \in (\mathsf{N} \cap \mathsf{coN})\mathsf{TIME}[T]$ *for some problem C and time T, then* $(\text{CNFSAT}, 2^n) \not\leq_{FGR} (C, T^{1+\gamma})$ *for any* $\gamma > 0$.

Proof. Assume NSETH and $(\text{CNFSAT}, 2^n) \leq_{FGR} (C, T^{1+\gamma})$, and $C \in (\mathsf{N} \cap \mathsf{coN})\mathsf{TIME}[T]$. By Lemma 1.5.2, preservation of $(\mathsf{N} \cap \mathsf{coN})\mathsf{TIME}$ savings under fine-grained reductions, there exists $\delta > 0$ such that $\text{CNFSAT} \in (\mathsf{N} \cap \mathsf{coN})\mathsf{TIME}[2^{n(1-\delta)}]$. This contradicts NSETH, therefore it cannot be the case (under NSETH) that $(\text{SAT}, 2^n) \leq_{FGR} (C, T)$. □

Corollary 2.1.2 (NSETH implies no reductions from SETH-hard problems)**.** *If* NSETH *holds and* $C \in (\mathsf{N} \cap \mathsf{coN})\mathsf{TIME}[T_1]$*, then for any problem B that is* SETH*-hard under deterministic reductions with time* T_2*, and* $\gamma > 0$*, we have*

$$(B, T_2) \not\leq_{FGR} (C, T_1^{1+\gamma})$$

Proof. Assume NSETH, and that (B, T_2) is SETH-hard. Therefore, we know

$$(\text{CNFSAT}, 2^n) \leq_{FGR} (B, T_2) \tag{2.1}$$

Now assume $(B, T_2) \leq_{FGR} (C, T_1^{1+\gamma})$. Then by Lemma 1.5.1, composition of fine-grained reductions, we have that $(\text{CNFSAT}, 2^n) \leq_{FGR} (C, T_1)$. But by Lemma 2 above, this is impossible under NSETH. □

We have the following theorem:

Theorem 3. *Under* NSETH*, there is no deterministic or zero-error fine-grained reduction from* SAT *or any* SETH*-hard problem to the following problems with the following time complexities for any* $\gamma > 0$.

- MAXFLOW, *min-cost* MAXFLOW, *and maximum matching with* $T(m) = m^{1+\gamma}$
- HITTINGSET *with* $T(m) = m^{1+\gamma}$
- 3-SUM *with* $T(n) = n^{1.5+\gamma}$
- *All-pairs shortest path with* $T(n) = n^{\frac{3+\omega}{2}+\gamma}$

The theorem is proved in the full version of [CGI+16].

2.2 Characterizing the Quantifier Structure of SETH-Hard FO Property Problems

This section studies the hardness of first-order model checking problems with the same number of quantifiers but different quantifier structures. If NSETH holds, all such formulas that are SETH hard are of a specific logical form. This is made precise as follows:

Theorem 4. *Let* $k \geq 3$. *If* NSETH *is true, then there is a problem in* FOP_k *that is* $O(m^{k-1})$ SETH-*hard, and all such formulas have the quantifier structure* $\forall^{k-1}\exists$ *or* $\exists^{k-1}\forall$.

The rest of this section proves the above theorem for the case where there are only unary and binary predicates (i.e., the input structure is a graph, instead of a hypergraph, so the problem is also called a *first-order graph property* problem). In this section, we will always assume formulas have k-quantifiers, instead of $(k+1)$ quantifiers.

There are graph properties with $\forall^{k-1}\exists$ and $\exists^{k-1}\forall$ quantifier structure that are SETH-hard for time $O(m^{k-1})$.

The $(\mathsf{k}-1)$-ORTHOGONALVECTORS problem is equivalent to the graph problem

$$\exists x_1 \ldots \exists x_{k-1} \forall x_k \left(P(x_1, x_k) \wedge \cdots \wedge P(x_{k-1}, x_k)\right) \tag{2.2}$$

The negation of k-ORTHOGONALVECTORS is also SETH-hard and has a $\forall^k\exists$ quantifier structure.

On the other hand if a problem is of any form other than $\forall^{k-1}\exists$ or its negation, we will show it has both smaller nondeterministic and co-nondeterministic complexity. We will assume without loss of generality that the outermost quantifiers is universal. We can handle the other case by simply negating the problem. A problem that does not have the quantifier structure $\forall^{k-1}\exists$ structure either has no existential quantifiers, or at least two existential quantifiers, or exactly one existential quantifier which is not in the innermost position.

Lemma 2.2.1. *If* φ *has more than one existential quantifier, then it can be solved in nondeterministic time* $O(m^{k-2})$. *If* φ *has more than one universal quantifier, then it can be solved in co-nondeterministic time* $O(m^{k-2})$

Proof. We concentrate on the case with more than one existential quantifier. The other case is symmetric.

These problems can be solved by guessing the existentially quantified variables, and doing exhaustive search on universally quantified variables. Because there are at most $k-2$ universal quantifiers, the algorithm runs in nondeterministic time $O(m^{k-2})$. Note that we operate on the sparse representation of the predicates. We assume that we are given the list of pairs that satisfy any predicate in a sorted order that allows us to do this exhaustive search without overhead.

Symmetrically, if φ has more than one universal quantifier, then it can be solved in co-nondeterministic time $O(m^{k-2})$. In particular, since $k \geq 3$, the model checking problem is always easy either nondeterministically or co-nondeterministically. □

Lemma 2.2.2. *If* φ *has exactly one existential quantifier which is not on the innermost position, then it can be solved in* $(\mathsf{N} \cap \mathsf{coN})\mathsf{TIME}[O(m^{k-2})]$. *Symmetrically, if* φ *has exactly one universal quantifier which is not on the innermost position, then it can be solved in* $(\mathsf{N} \cap \mathsf{coN})\mathsf{TIME}[O(m^{k-2})]$.

Proof. We only show the case with one existentially quantified variable. Since $k \geq 3$, the problem is in $\mathsf{coNTIME}[O(m^{k-2})]$ by Lemma 2.2.1. We will show that it is also in $\mathsf{NTIME}[O(m^k - 2)]$.

Let the existentially quantified variable be x_j.

We give all predicates a canonical order, and let the universes for $x_1, \ldots, x_k$ be $X_1, \ldots, X_k$ respectively. For a tuple of vertices $(v_{i_1} \in X_{i_1}, \ldots, v_{i_\ell} \in X_{i_\ell})$, we define its *color* $\chi(v_{i_1}, \ldots, v_{i_\ell})$ to be the concatenation of truth values of all predicates on any subset of the tuple of variables $(x_{i_1}, \ldots, x_{i_\ell})$. We also define For the color where all predicates are false, we call it the "background color".

Our nondeterministic algorithm will count the number of $v_k \in X_k$ with color $\chi(v_1 \ldots v_k) = c$ for all tuples $(v_1 \in X_1, \ldots, v_{k-1} \in X_{k-1})$ and all color c in time $O(m^{k-2})$. The main idea of the algorithm is to nondeterministically guess x_j and count the valid values of x_k, so that it saves the exhaustive search time on x_j and x_k.

1. For each combination of vertices $(v_1 \in X_1, \ldots, v_{j-1} \in X_{j-1})$, we nondeterministically bundle a fixed v_j vertex to it. This takes time $O(m^{j-1})$, which is at most $O(m^{k-2})$ because $j < k$. In the rest of this algorithm, given any $(v_1, \ldots, v_{j-1})$ values we can find their corresponding v_j value in constant time.

2. The algorithm runs a $(k-2)$-layer nested loop. On each layer we enumerate all (x_i, x_k) edges[1] where x_i is a variable other than x_j or x_k. Then inside all the loops, for each x_k in the $(k-2)$ current (x_i, x_k) edges, we record the color $\chi(x_1 \ldots x_{k-1})$, where the value of x_j come from the current tuple $(x_1, \ldots, x_{j-1})$.

   Inside the $(k-2)$-layer loop, we enumerate all possible colors on variables $(x_1, \ldots, x_k)$ that agrees with the color on the current tuple $(x_1, \ldots, x_{k-1})$. The total number of predicates is constant, therefore this loop only contributes a constant factor to the running time.

   Then inside the loops we can count the number of x_k's of for the current $(x_1, \ldots, x_{k-1})$ and the current color.

   This whole process takes time $O(m^{k-2})$.

3. The previous step did not count the x_k vertices that only appear in (x_j, x_k) edges, i.e. whose $\chi(x_1 \ldots x_k)$ is all-false on all non-(x_j, x_k) predicate positions, but not all-false on some (x_k, x_j)

[1] When dealing with hypergraphs where edges can have arity higher than 2, we enumerate all edges whose corresponding variable list contains variables x_i, x_k

predicate positions. We will count these x_k's in this step.

We use c_{jk} to denote the "partial color" corresponding to a truth value combination of all unary and binary predicates on variables x_j, x_k and on (x_j, x_k). For each x_j and each possible c_{ij} value, we can enumerate all the (x_j, x_k) edges to count the number of x_k where $\chi(x_j, x_k) = c_{jk}$ for any not all-false c_{jk}. Also, the previous step has shown that we can count the number of x_k where $\chi(x_1 \ldots x_k)$ is not all-false on some non-(x_j, x_k) predicates, and also equals c_{ij} on its (x_j, x_k) predicates. By subtraction we can get the number of x_k where $\chi(x_1 \ldots x_k)$ is all-zero on non-(x_j, x_k) predicate positions, and equals c_{ij} on its (x_j, x_k) predicate positions. Similarly as the previous step, this process runs in time $O(m^{k-2})$.

4. Now we have counted the x_k's for all non-background colors for all $(x_1, \ldots, x_{k-1})$. The number of x_k's where $\chi(x_1 \ldots x_k)$ is background color can be computed by $|X_k|$ subtracting the numbers of all non-background x_k's.

5. Finally, for all $(x_1, \ldots, x_{k-1})$, we sum the number of x_k's of all colors that satisfy φ. If it always equals $|X_k|$, then the algorithm accepts.

□

The last case, where either all quantifiers are existential or all quantifiers are universal is easy deterministically and therefore also not a candidate for SETH-hardness, independent of NSETH.

Lemma 2.2.3. *If all quantifiers are existential, or all quantifiers are universal, then the problem can be solved in deterministic time $O(m^{k-1.5})$.*

Proof. This lemma is implied by Lemma 3.9.2. □

Thus, only $\forall^{k-1}\exists$ formulas require $O(m^{k-1})$ nondeterministic time, and by looking at the complements, only $\exists^{k-1}\forall$ formulas require $O(m^{k-1})$ co-nondeterministic time. Thus, assuming NSETH, only these two types of first-order properties might be SETH-hard for the maximum difficulty of a k-quantifier formula.

2.3 Acknowledgments

Chapter 2 contains material from "Nondeterministic Extensions of the Strong Exponential Time Hypothesis and Consequences for Non-Reducibility", by Marco L Carmosino, Jiawei Gao, Russell Impagliazzo, Ivan Mihajlin, Ramamohan Paturi, and Stefan Schneider, which appeared in the proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science (ITCS 2016). The author of this dissertation was a principal author of this publication. We would like to thank Amir Abboud, Karl Bringmann, Bart Jansen, Sebastian Krinninger, Virginia Vassilevska Williams, Ryan Williams and the anonymous reviewers for many helpful comments on an earlier draft.

Chapter 3

The Completeness of Orthogonal Vectors

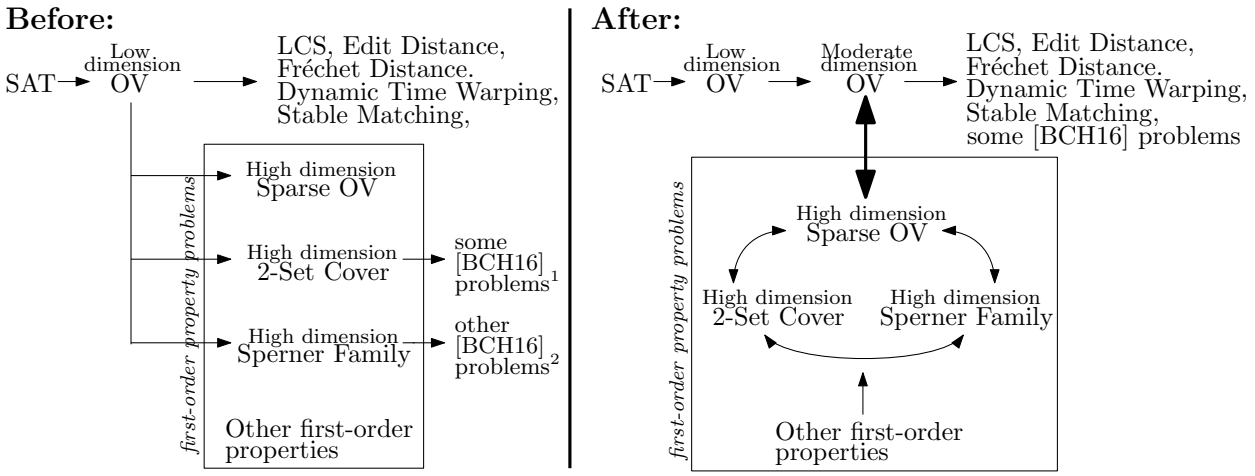
3.1 Chapter Overview

3.1.1 Motivation

In this chapter, we give the first results in fine-grained complexity that apply to an entire complexity class, namely the class of first-order definable properties (the uniform version of AC^0.)

It is not difficult to see that checking whether a property expressible by a first-order formula with $k+1$ quantifiers holds on a given structure with m tuples can be done in $O(m^k)$ time. If the Strong Exponential Time Hypothesis (SETH) is true, there are such properties that require $m^{k-o(1)}$ time to decide. The k-Orthogonal Vectors problem is an example that requires $m^{k-o(1)}$ time under SETH.

For $k = 1$ (i.e. there are only two quantified variables), this $O(m^k)$ upper bound is linear time and so cannot be improved. For any such problem with $k \geq 2$, we give an algorithm that solves it in $m^k/2^{\Theta(\sqrt{\log m})}$ time, which improves the standard algorithm by a factor more than any poly-log, but less than the polynomial improvement needed to refute SETH. Moreover, we show that any further improvement is *equivalent* to a similar algorithmic improvement for the well-studied Orthogonal Vectors problem. Surprisingly, both results are obtained by showing that (a version



[1] includes 3-Dominating Set and Bipartite Subset 2-Dominating Set.
[2] includes Graph Dominated Vertex, Maximum Simple Family of Sets, and Subset Graph.

Figure 3.1: A diagram of reductions. We simplify this picture, and make the reductions to Edit Distance, LCS, etc. more meaningful.

of) the Orthogonal Vectors problem is *complete* under fine-grained reductions for the class of all first-order properties. This is the first completeness result for a previously studied complexity class under fine-grained reducibility. By applying the algorithm for the Orthogonal Vectors problem of [AWY15], which uses techniques from circuit lower bounds, and later derandomized by [CW16], we obtain algorithmic results.

In addition to introducing new algorithms and giving completeness results, our results clarify and simplify our understanding of "complexity within P". For many of the known SETH-hard problems of interest such as Edit Distance [BI15], Longest Common Subsequence [AWW14, ABW15, BK15], Dynamic Time Warping [ABW15, BK15], Fréchet Distance [Bri14], Succinct Stable Matching [MPS16], etc., the reduction from SAT passes through the Orthogonal Vectors problem. Thus, if any of these SETH-hard problems had substantially improved algorithms, all first-order properties would have similarly improved algorithms. Thus FOPC, the hypothesis that *some* first-order property does not have a substantially faster algorithm, is a useful intermediary between SETH and many of its consequences. FOPC is both equivalent to conjectures concerning many of the previously studied problems ([BCH16]), and potentially more plausible to SETH-skeptics since it concerns an entire complexity class, while having most of the consequences of SETH. This is

summarized in Figure 3.1.

While we concentrate on the general picture of complexity classes, even special cases of our results for specific problems are of interest. There were no similarly improved algorithms for Orthogonal Vectors with small total Hamming weight (Sparse OV) or related problems such as Sperner Family and 2-Set Cover (in the sparse high-dimensional case), and it was not previously known that the sparse versions of these problems were equivalent.

In addition to having a natural and useful complete problem, the class of first-order properties is important in itself. This class includes many problems studied in the fine-grained complexity literature such as Hitting Set, Orthogonal Vectors, Sperner Family, Diameter 2, Radius 2, k-Independent Set, k-Dominating Set and so on.

Algorithms for model-checking first-order properties are inherent in databases (the core of the relational database language *SQL* is equivalent to first-order properties). Roughly speaking, first-order properties are essentially the uniform version of AC^0 in the complexity literature [BIS90].

Since fine-grained complexity is concerned with exact time complexities (distinguishing e.g. $n^{1.9}$ time from n^2 time), the problem representation is significant. For graph problems, there are two standard representations: adjacency lists (which are good for *sparse* graphs), in which running time is analyzed with respect to the number of edges m, and adjacency matrices (good for *dense* graphs), in which the runtime is a function of the number of vertices, n. For several reasons, we use the sparse adjacency list (list of tuples) representation. First, many of the problems considered such as Orthogonal Vectors are hard already on sparse instances. Secondly, the complexity of first-order problems in the dense model is somewhat unclear, at least for numbers of quantifiers between 3 and 7 ([Wil14a]). Third, the sparse model is more relevant for first-order model checking, as databases are represented as lists of records.

3.1.2 Main Results

Completeness. First, we show that conjectures for OV defined on dense (moderate-dimension) and sparse models are equivalent under fine-grained reductions, which means MDOVC is true iff SOVC is true (see Lemma 3.5.2). This also holds for k-OV.

Lemma 3.1.1. *For any integer $k \geq 2$, there exist $\delta, \varepsilon > 0$ and a $O(n^{k-\varepsilon})$ time algorithm solving k-OV with dimension $d = n^{\delta}$, if and only if there is an $\varepsilon' > 0$ and a $O(m^{k-\varepsilon'})$ time algorithm for Sparse k-OV, where m is the total Hamming weight of all input vectors.*

Our main result establishes an equivalence of MDOVC and FOPC, showing the completeness of Sparse OV and hardness of (dense) OV for the class of first-order property problems.

Theorem 5. *MDOVC, SOVC and FOPC are equivalent.*

This chapter also proves a hardness and completeness result for k-OV, connecting one combinatorial problem to a large and natural class of logical problems. The following theorem states that Sparse k-OV is complete for $MC(\exists^k\forall)$ (and its negation form $MC(\forall^k\exists)$), and hard for $MC(\forall\exists^{k-1}\forall)$ (and its negation form $MC(\exists\forall^{k-1}\exists)$) under fine-grained reductions.

Theorem 6. *If Sparse k-OV with total Hamming weight m can be solved in time $O(m^{k-\varepsilon})$ for some $\varepsilon > 0$, then all problems in $MC(\exists^k\forall)$, $MC(\forall^k\exists)$, $MC(\forall\exists^{k-1}\forall)$ and $MC(\exists\forall^{k-1}\exists)$ are solvable in time $O(m^{k-\varepsilon'})$ for some $\varepsilon' > 0$.*

$MC(\exists^k\forall)$ and $MC(\forall^k\exists)$ are interesting sub-classes of FOP_{k+1}: if Nondeterministic SETH is true, then all SETH-hard problems in FOP_{k+1} are contained in $MC(\exists^k\forall)$ or $MC(\forall^k\exists)$ ([CGI+16]).

We will also show that the 2-Set Cover problem and the Sperner Family problem, both in $MC(\exists\exists\forall)$, are equivalent to Sparse OV under fine-grained reductions, and thus complete for first-order properties under fine-grained reductions.

Algorithmic results. Combining our reductions with the surprisingly fast algorithm for Orthogonal Vectors by [AWY15] and [CW16], we obtain improved algorithms for every problem representable as a $(k+1)$-quantifier first-order property.

Theorem 7. *There is an algorithm solving* FOP_{k+1} *in time* $m^k/2^{\Theta(\sqrt{\log m})}$.

Let us consider the above results in context with prior work on the fine-grained complexity of first-order properties. In [Wil14a], Ryan Williams studied the fine-grained complexity of *dense* instances of first-order graph properties. He gave an $n^{k+o(1)}$-time algorithm for FOP_{k+1} on *graphs* when k is a sufficiently large constant, and showed that FOP_{k+1} requires at least $n^{k-o(1)}$ time under SETH. His algorithms only apply to graphs (they look difficult to generalize to even ternary relations), and it seems difficult to point to a specific simple complete problem in this setting. To compare, our results show that the *sparse* case of FOP_{k+1} (for *all* c-ary relations, for all constants c) is captured by very simple problems (e.g. sparse Orthogonal Vectors), which also leads to an algorithmic improvement for all c-ary relations.

3.1.3 Organization of this Chapter

We present a general outline of the proofs in Section 3.2, and a high-level explanation of key techniques in Section 3.3. The full proof starts with the reduction from $MC(\exists^k\forall)$ to k-OV (Section 3.4), with randomized universe-shrinking self-reduction described in Section 3.4.1, which is then derandomized in Section 3.5. Section 3.7 presents the reduction from $MC(\forall\exists^{k-1}\forall)$ to k-OV, and Section 3.8 discusses the $m^k/2^{\Theta(\sqrt{\log m})}$ time algorithm for Sparse OV and, therefore, FOP_{k+1}. We conclude with open problems in Section 3.10. .

3.2 Outline of the Proof

The main technical part of this chapter is in the proof of Theorem 6 showing hardness of k-OV for model-checking of $\exists^k\forall$ formulas under fine-grained reductions. The idea is to represent $\exists^k\forall$ formulas using combinations of basic "k-OV like" problems, each of which is either easy (solvable substantially faster than m^k time for sparse instances) or can be fine-grained reduced to k-OV. The latter is achieved using a universe-shrinking self-reduction, which converts a given

instance of a basic problem to a denser instance on a smaller universe, thus reducing Sparse k-OV to k-OV with dimension n^{δ} and proving Lemma 3.1.1. Converting an $MC(\exists^k\forall)$ to the "hybrid problem" combining all 2^k basic problems is done for graphs (all relations have arity at most 2), however we show that this is the hardest case. Additionally, $MC(\forall\exists^{k-1}\forall)$ is reduced to $MC(\exists^k\forall)$.

In Theorem 5 and Theorem 7, we consider the class of all $k+1$-quantifier first-order properties FOP_{k+1}, and reduce it to 2-OV, proceeding to use a faster algorithm for 2-OV to speed up model checking. The first step is to brute-force over first $k-2$ quantified variables, reducing to three-quantifier case at the cost $O(n^{k-2})$. The quantifier prefix $\exists\exists\forall$, with 2-OV and other basic problems (to be defined in Section 3.4.1), is the hardest ($\exists^3$, $\forall\exists\exists$ and their complements are easy, and the rest reduce to $\exists\exists\forall$). Appealing to lemmas in the proof of Theorem 6 with $k=2$ completes the proof of Theorem 5, and applying the OV algorithm in [AWY15, CW16] gives Theorem 7.

The following outlines the reduction from any arbitrary problem in FOP_{k+1} to OV for any integer $k \geq 2$, thus proving that FOPC implies SOVC. For the other direction of this eqiuvalence, SOVC implies FOPC because Sparse OV is in $MC(3)$. The equivalence between SOVC and MDOVC is proven in Lemma 3.1.1, which in turn follows from Lemma 3.4.2, Lemma 3.5.2, and corollary 3.4.1.

1. With brute-force over tuples of first $k-2$ variables, we reduce from the $(k+1)$-quantifier problem MC_{φ} down to a 3-quantifier problem $MC_{\varphi'}$. Thus, improving the $O(m^2)$ algorithm for $MC_{\varphi'}$ implies improving the $O(m^k)$ algorithm for MC_{φ}.
2. If $MC_{\varphi'}$ belongs to one of $MC(\exists\exists\exists)$, $MC(\forall\forall\forall)$, $MC(\forall\exists\exists)$, $MC(\exists\forall\forall)$, we solve it directly in time $O(m^{3/2})$, using ideas similar to triangle detection algorithms. If φ' has the quantifier structure $\forall\exists\forall$ (or its negated form $\exists\forall\exists$), we reduce $MC_{\varphi'}$ to $MC_{\varphi''}$ where φ'' has quantifier structure $\exists\exists\forall$, using Lemma 3.7.1. Otherwise, φ' is already in $\exists\exists\forall$ or equivalent form.
3. We reduce a general model checking problem for φ'' of the quantifier structure $\exists\exists\forall$ to a graph property problem of the same quantifier structure.
4. Using Lemma 3.4.5, we reduce formulas of form $\exists\exists\forall$ to a "hybrid" problem, which by Lemma 3.4.4 can be reduced to a combination of Sparse OV, Set Containment and 2-Set

Cover (which we call *Basic Problems*).

5. We use a "universe-shrinking" technique (Lemmas 3.4.2, 3.5.1, and 3.5.2) on each of the Basic Problems, to transform a sparse instance into an equivalent one of small dimension.
6. After applying this to the Hybrid Problem, we can complement edge relations as needed to transform all Basic Problems into OV (Lemmas 3.4.3 and 3.5.3).
7. By applying the [AWY15, CW16] algorithm to the instance of low-dimension OV, we get an improved algorithm.

Moreover, Lemmas 3.7.1, 3.4.5, 3.4.4 and 3.4.1 also work for any constant $k \geq 2$. So for a problem in $MC(\exists^k\forall)$ or $MC(\forall\exists^{k-1}\forall)$, we can reduce it to k-OV as follows:

1. If the problem belongs to $MC(\forall\exists^{k-1}\forall)$, reduce it to $MC(\exists^k\forall)$ using Lemma 3.7.1.
2. Eliminate hyperedges, then reduce the $MC(\exists^k\forall)$ to the Hybrid problem using Lemma 3.4.5.
3. Reduce from the Hybrid Problem to a combination of 2^k Basic Problems, using Lemma 3.4.4.
4. Reduce all Basic Problems to k-OV, using Lemma 3.4.1.

This completes the proof of Theorem 6.

3.3 The Building Blocks

Before the formal presentation of the reduction algorithms, this section gives an intuitive and high-level view of the techniques used to reduce a first-order property problem to OV, in the proofs of Theorems 5, 6 and 7. Because of Lemma 3.1.1, in the remainder of this paper, unless specified, we will use "OV" and "k-OV" to refer to sparse versions of these problems.

3.3.1 Complementing Sparse Relations

Consider the problems k-Empty Intersection, k-Set Cover and Set Containment problems, defined as follows:

k-Empty Intersection (k-EI): $(\exists S_1 \in \mathcal{S}_1)\dots(\exists S_k \in \mathcal{S}_k)(\forall u \in U)\left[\bigvee_{i=1}^{k} \neg(u \in S_i)\right]$.

k-Set Cover: $(\exists S_1 \in \mathcal{S}_1)\dots(\exists S_k \in \mathcal{S}_k)(\forall u \in U)\left[\bigvee_{i=1}^{k} (u \in S_i)\right]$.

Set Containment: $(\exists S_1 \in \mathcal{S}_1)(\exists S_2 \in \mathcal{S}_2)\ (\forall u \in U)\,[(\neg(u \in S_1)) \vee (u \in S_2)]$.

OV is also 2-EI or Set Disjointess: k-EI is equivalent to k-OV, where vectors are represented by sets containing their 1s. Set Containment is equivalent to the Sperner Family, and k-Set Cover to k-Dominating Set under linear-time reductions.

These problems have very similar structure: given set families $\mathcal{S}_1 \dots \mathcal{S}_k$ containing sets over elements of the universe U, each of them asks whether there is a tuple of sets, one in each family, such that a formula is satisfied for every element u of the universe. Moreover, the formulas themselves are disjunctions of $u \in S_i$ or $u \notin S_i$, with one predicate for each i. The only difference is the polarity of the $\in$ relation (whether or not it is negated). We will refer to these types of problems as the *Basic Problems*; they will be our main building blocks.

For $k = 2$, this gives us four basic problems: Set Disjointness, 2-Set Cover and two versions of Set Containment (direct and reversed). In each of them, the input consists of two set families $\mathcal{S}_1, \mathcal{S}_2$ of sizes n_1, n_2, respectively, and the universe U of size n_u. The goal is to decide if there exist sets $S_1 \in \mathcal{S}_1$ and $S_2 \in \mathcal{S}_2$ such that for every $u \in U$, the corresponding formula ψ_ℓ holds. Here, $\ell \in \{00, 01, 10, 11\}$ codes the sequence of polarities of occurrences of $\in$. This naturally generalizes to arbitrary k, with a Basic Problem for each $\ell \in \{0,1\}^k$; see Section 3.4.1 for formal definitions.

That is, Set Disjointness, 2-Set Cover and Set Containment can be stated as follows. Decide if $\exists S_1 \in \mathcal{S}_1 \exists S_2 \in \mathcal{S}_2 \forall u \in U \psi_\ell$ holds, where ψ_ℓ is:

Set Disjointness: There is no common element in S_1 and S_2: $\psi_\ell = \psi_{00} = \neg(u \in S_1) \vee \neg(u \in S_2)$.

2-Set Cover: Union of S_1 and S_2 covers all of U: $\psi_\ell = \psi_{11} = (u \in S_1) \vee (u \in S_2)$.

Set Containment: For $S_1 \subseteq S_2$, $\psi_\ell = \psi_{01} = \neg(u \in S_1) \vee (u \in S_2)$. Similarly, in reversed Set Containment with $S_2 \subseteq S_1$, $\psi_\ell = \psi_{10} = (u \in S_1) \vee \neg(u \in S_2)$.

All these problems are first-order properties: we can use unary relations to partition the

vertex set into $(\mathcal{S}_1, \ldots, \mathcal{S}_k, U)$, and consider the relation "$\in$" as a binary relation. We will use the context of hypergraphs to describe the input structure. We let n (corresponding to the number of vertices in the input graph) be the sum of $n_1, \ldots, n_k$ and n_u, and let the input size m (corresponding to the number of edges in the input graph) be the sum of all sets' sizes in all set families. Borassi et al. [BCH16] showed that when $k = 2$, these Basic Problems require time $m^{2-o(1)}$ under SETH, and that if the size of universe U is poly-logarithmic in the input size, then the three problems are equivalent under subquadratic-time reductions. The main idea of the reductions between these problems is to complement all sets in $\mathcal{S}_1$, or $\mathcal{S}_2$, or both. It is easy to see that $S_1 \cap S_2 = \emptyset \iff {S_1}^{\complement} \cup {S_2}^{\complement} = U \iff S_1 \subseteq {S_2}^{\complement} \iff S_2 \subseteq {S_1}^{\complement}$. Therefore, if we could complement the sets, we can easily prove the equivalences between the three Basic Problems. However we cannot do this when n_u is large.

For a sparse binary relation such as $(u \in S_1)$, we say that its complement, such as $(u \notin S_1)$, is *co-sparse*. Suppose we want to enumerate all tuples (S_1, u) s.t. $u \in S_1$; for that, we can go through all relations (aka edges) between U and $\mathcal{S}_1$, which takes time linear in m. On the contrary, if we want to enumerate all pairs (S_1, u) s.t. $u \notin S_1$, we cannot do this in linear time, because we cannot touch the pairs by touching edges between them. Moreover, when n_u is as large as $\Omega(n)$, the number of such pairs can reach $\Theta(m^2)$. When $k = 2$, a fine-grained reduction between $O(m^2)$-time problems allows neither quadratic time reductions, nor quadratic size problem instances.

Because of the above argument, it is hard to directly reduce between the Basic Problems, so instead we reduce each problem to a highly-asymmetric instance of the same problem, where sparse relations are easily complemented to relations that are also sparse. Observe that when the size of universe U is small enough, complementing all sets can be done in time $O(m \cdot |U|)$, which can be substantially faster than $O(m^2)$. The new instance created also has size $O(m \cdot |U|)$, so that it is only slightly larger than m. So by carefully choosing the size of U, we can construct truly subquadratic time reduction algorithms that preserve the improved factor in running time. Using this technique which we call *universe-shrinking self-reduction*, we can show that OV, 2-Set Cover and Set Containment are equivalent under fine-grained reductions.

The self-reduction employs the "high-degree low-degree" trick, which has been also used in other sparse graph algorithms [AYZ95]. First, consider sets of large cardinality: there cannot be too many of them, because the structure is sparse. Thus we can do exhaustive search over these sets to check if any of them is in a solution. For sets of small cardinality, we hash the universe U to a smaller universe, where complementing the sets does not take too much time and space. From this reduction, the claim follows:

Claim 3.3.1. *If any one of OV, 2-Set Cover and Set Containment has truly subquadratic time algorithms, then the other two are also solvable in subquadratic time. Thus these problems are all hard for* FOP_3.

Claim 3.3.1 is itself an interesting result: in [BCH16], conditional lower bounds for many problems stem from the above three problems, forming a tree of reductions. By our equivalence, the root of the tree can be replaced by the quadratic-time hardness conjecture on any of the three problems, simplifying the reduction tree. Claim 3.3.1 also shows that an improved algorithm for any of these three problems implies improved algorithms for the other two.

Claim 3.3.1 is proven by derandomizing Lemma 3.4.1 for $k = 2$; see Section 3.5 for details. In Section 3.4 we give randomized reductions for an arbitrary k.

3.3.2 Sparse and co-Sparse Relations

Having shown how to reduce any two Basic Problems with the same k to each other, we will now reduce generic first-order properties to the Basic Problems. The detailed processes are complicated, so here we start with a high-level idea in reductions and algorithm design throughout this chapter.

Our algorithms often need to iterate over all tuples or pairs (x_i, x_j) satisfying some conditions, to list such tuples, or to count the number of them, performing first-order *query processing*. A set of such tuples (pairs) (x_i, x_j) can be considered a result of a *first-order query* defined by an intermediate formula φ' on the (hyper)graph G (or some intermediate structures). Our reduction

algorithms often generate such queries, evaluate them, and combine the results (e.g. by counting) to compute the solutions.

There are three possible outcomes of such queries: the result can be a sparse set of tuples, a co-sparse set, or neither. If the result of the query is a sparse relation such as $[R_1(x_1,x_2) \wedge \neg R_2(x_1,x_2)]$, we can iterate over the tuples (say, first enumerate all pairs satisfying $R_1(x_1,x_2)$, then check for which of them $R_2(x_1,x_2)$ is false). Then, we can do further operations on the sparse set of (x_1,x_2) tuples resulting from the query. When the result of the query is a co-sparse set such as for $[\neg R_1(x_1,x_2) \wedge \neg R_2(x_1,x_2)]$, we cannot directly iterate over pairs satisfying the query. Instead, we work on its complement (which is sparse, instead of co-sparse), but then do some further processing to filter out those pairs from future use (say, find all pairs (x_1,x_2) so that at least one of $R_1(x_1,x_2)$ or $R_2(x_1,x_2)$ is true, then exclude those pairs from future use). Sometimes, the result of a query is neither sparse nor co-sparse, but we will show it is always a combination of sparse and co-sparse relations. Thus we need to distinguish them and deal with the sparse and co-sparse parts separately.

We exemplify this process by considering the query $[\neg R_1(x_1,x_2) \vee \neg R_2(x_1,x_2)]$. For a pair (x_1,x_2), to make the formula true, predicates R_1, R_2 can be assigned values from $\{(\mathit{True},\mathit{False})$, $(\mathit{False},\mathit{True})$, $(\mathit{False},\mathit{False})\}$. In the first two cases, the sets of pairs (x_1,x_2) satisfying $[R_1(x_1,x_2) \wedge \neg R_2(x_1,x_2)]$ and $[\neg R_1(x_1,x_2) \wedge R_2(x_1,x_2)]$ are sparse, while in the last case, the set of pairs satisfying $[\neg R_1(x_1,x_2) \wedge \neg R_2(x_1,x_2)]$ is co-sparse. So if we want to work on the tuples satisfying this query, we list tuples satisfying the first two cases directly by enumerating edges, and enumerate the tuples *not* satisfying the third case (i.e., the tuples where either $R_1(x_1,x_2)$ or $R_2(x_1,x_2)$ is true), in order to exclude them from future use.

In general, a query can be written as a DNF, where the result of each term is a conjunction of predicates and negated predicates, and therefore either sparse or co-sparse. Then we can deal with the sparse and co-sparse cases separately. We will use this technique for constructing the Hybrid Problem in Section 3.4.4.

Now, we would like to reduce MC_φ to OV for an arbitrary $\varphi = (\exists x)(\exists y)(\forall z)\psi(x,y,z)$. First, suppose that all predicates $R_1 \ldots R_r$ in ψ are at most binary, and all binary predicates involve z. One

attempt is to create a set S_x for each element x and a set S_y for each element y. Then, we create elements in universe U by creating 2^r elements $u_{(z,0^r)}, \ldots, u_{(z,1^r)}$ for each z, where r is the number of different predicates in ψ, and the length-r strings in the subscripts correspond to the 2^r truth assignments of all these predicates. We construct the sets so that S_x (or S_y) contains element $u_{(z,a)}$ iff the assignment a falsifies ψ and the relations between x (or y) and z agree with a. In this way, sets S_x and S_y both contain some element $u_{(z,a)}$ in U iff there is some z such that x, y, z do not satisfy ψ. Then, if there exists such pair of disjoint sets S_x and S_y, the corresponding x and y satisfy that for all z, ψ is true.

However, we cannot touch all z's for each x or y for creating this instance in substantially less than n^2 time. So, we divide the relations of this Set Disjointness instance into sparse and co-sparse ones. For that, we introduce a *Hybrid Problem* which is a combination of Basic Problems. Depending on the four combinations of sparsity or co-sparsity on the relations between variables x, z and y, z, we reduce MC_φ not only to OV, but to a combination of OV, Set Containment, reversed Set Containment (i.e. finding $S_2 \subseteq S_1$ instead of $S_1 \subseteq S_2$), and 2-Set Cover. (Namely, the sub-problem Set Disjointness deals with the case where the relations between x and z and between y and z are both sparse; the sub-problems Set Containment, reversed Set Containment and 2-Set Cover deals with the cases where these relations are sparse and co-sparse, co-sparse and sparse, co-sparse and co-sparse respectively.) We decide if there is a pair of sets being the solutions of all sub-problems. Finally, because these Basic Problems can be reduced to each other, we can use the algorithm for OV to solve the instance of the Hybrid Problem, and then to solve MC_φ.

This approach takes care of binary predicates involving z; to handle relations among existentially quantified variables, additional tools are needed. Thus, the Hybrid Problem definition also involves a relation $R(x, y)$ and a "sparsity type" designation, specifying whether R codes a sparse relation between x and y, or its sparse complement. However, this additional information can be modeled by adding new elements to the universe and strategically placing them in the corresponding sets, thus reducing the more complex case to a combination of four Basic Problems.

See Lemma 3.4.5 for the proof that covers more complicated cases.

3.4 Completeness of k-OV in $MC(\exists^k\forall)$

This section will prove the completeness of k-OV in $MC(\exists^k\forall)$ problems. Here we only consider the input structures that are graphs, i.e. where all relations are either unary or binary; see Section 3.6 for the reduction from hypergraphs to graphs. First, we introduce a class of Basic Problems, and prove that these problems are equivalent to k-OV under exact complexity reductions. Then, we show that any problem in $MC(\exists^k\forall)$ can be reduced to a combination of Basic Problems (aka. the Hybrid Problem).

3.4.1 How to Complement a Sparse Relation: Basic Problems, and Reductions Between Them

In this section we define the Basic Problems for $k \geq 2$, generalizing k-OV, k-Set Cover and Set Containment problems, and prove that these problems are fine-grained reducible to each other under randomized reductions. In Section 3.5 we will give deterministic reductions for $k = 2$.

Let $k \geq 2$. We introduce 2^k Basic Problems labeled by k-bit binary strings from 0^k to 1^k. The input of these problems is the same as that of k-EI defined in Appendix A: k set families $\mathcal{S}_1 \ldots \mathcal{S}_k$ of size $n_1, \ldots, n_k$ on a universe U of size n_u. We define 2^k quantifier-free formulas $\psi_{0^k}, \ldots, \psi_{1^k}$ such that

$$\psi_\ell = \left(\bigvee_{i,\ell[i]=0}(\neg(u \in S_i))\right) \vee \left(\bigvee_{i,\ell[i]=1}(u \in S_i)\right).$$

Here, $i \in \{1, \ldots, k\}$ and $\ell[i]$, the i-th bit of label ℓ, specifies whether u is in each S_i or not in the i-th term of ψ_ℓ.

For each ℓ, let $\varphi_\ell = (\exists S_1 \in \mathcal{S}_1) \ldots (\exists S_k \in \mathcal{S}_k)(\forall u \in U)\psi_\ell$. For simplicity, we will omit the domains of the variables in these formulas. We call $MC_{\varphi_{0^k}}, \ldots, MC_{\varphi_{1^k}}$ the Basic Problems. We refer to the Basic Problem MC_{φ_ℓ} as $BP[\ell]$. These problems are special cases of first-order model checking on graphs, where sets and elements correspond to vertices, and membership relations correspond to edges. Note that $BP[0^k]$ is k-EI, and $BP[1^k]$ is k-Set Cover. When $k = 2$, $BP[01]$ and

$BP[10]$ are Set Containment problems, and $BP[00]$ is the Set Disjointness problem. For a k-tuple $(S_1 \in \mathcal{S}_1, \ldots, S_k \in \mathcal{S}_k)$ satisfying $(\forall u)\psi_\ell$, we call it a *solution* of the corresponding Basic Problem $BP[\ell]$.

We present a randomized[1] fine-grained mapping reduction between any two Basic Problems, thus proving the following lemma, which generalizes Claim 3.3.1 to $k > 2$.

Lemma 3.4.1. *Let $s(m)$ be a non-decreasing function such that $2^{\Omega(\sqrt{\log m})} \leq s(m) < m^{1/5}$. For any $\ell_1, \ell_2 \in \{0,1\}^k$, there is a randomized exact complexity reduction $(BP[\ell_1], m^k/(s(m))^{1/6}) \leq_{EC} (BP[\ell_2], m^k/s(m))$.*

For problems $BP[\ell_1]$ and $BP[\ell_2]$ where ℓ_1 and ℓ_2 only differ in the i-th bit, if we are allowed to complement all sets in $\mathcal{S}_i$, we can easily reduce between them. Similarly, if ℓ_1 and ℓ_2 differ in more than one bit, we can complement all the sets in corresponding set families. However, complementing the sets in $\mathcal{S}_i$ takes time $O(n_i n_u)$, which might be as large as $\Theta(m^2)$. To solve this, we self-reduce $BP[\ell_1]$ on the universe U to the same problem on a smaller universe U', and then complement sets on U'. For any given δ, if the size of U' is $n'_u = O(m^\delta)$, then complementing all sets in $\mathcal{S}_i$ only takes time and space $m \cdot O(m^\delta) = O(m^{1+\delta})$.

Lemma 3.4.2. (Randomized universe-shrinking self-reductions of Basic Problems)
Let label ℓ be any binary string in $\{0,1\}^k$. For any $s(m) = 2^{\Omega(\sqrt{\log m})}$, given a $BP[\ell]$ instance I of size m and universe U of size n_u, we can either solve it in time $O(m^k/s(m))$, or use time $O(m^k/s(m))$ to create a $BP[\ell]$ instance I' of size $O(m \cdot s(m)^5)$ on universe U' whose size is $n'_u = O(s(m)^5)$, so that $I \in BP[\ell]$ iff $I' \in BP[\ell]$ with error probability bounded by $O(1/s(m))$.

Note that the self-reduction of k-OV actually reduces the Sparse k-OV to a moderate-dimension version of k-OV, implying Lemma 3.1.1. The other direction (moderate-dimension k-OV to Sparse k-OV) is easy since if the dimension $d = n^\delta$, then m is at most $d \cdot n = n^{1+\delta}$, as required.

Corollary 3.4.1. (Reverse direction of Lemma 3.1.1)
Suppose that for any $k \geq 2$ there exists $\delta, \varepsilon > 0$ and a (randomized) $O(n^{k-\varepsilon})$ algorithm solving k-OV

[1] The deterministic reduction will be presented in Section 3.5.

with dimension $d = n^{\delta}$. Then there is an $\varepsilon' > 0$ and a (randomized) $O(m^{k-\varepsilon'})$ time algorithm solving Sparse k-OV.

Proof. The algorithm converts an instance of Sparse k-OV to an instance of k-OV of dimension n^{δ} using universe-shrinking self-reduction (Lemma 3.4.2) and then applies assumed $O(n^{k-\varepsilon})$ time algorithm to the reduced instance. More specifically, let $m = O(n^{1+\gamma})$, where n is the number of vectors. Choosing $s(m) = O(m^{\delta/5(1+\gamma)})$ for some $\delta > 0$ creates an instance of OV with dimension $n'_u = O(s(m)^5) = O(n^{\delta})$, and size $m' = O(n^{1+\delta+\gamma})$; number of vectors n remains unchanged. Now, the reduction takes time $O(m^k/(s(m))^5) = O(m^{k-\delta/(1+\gamma)})$, and running the $O(n^{k-\varepsilon})$ time algorithm on the reduced instance takes $O(n^{k-\varepsilon}) \leq O(m^{k-\varepsilon/(1+\gamma)})$ time. Setting $\varepsilon' = \min\{\delta/(1+\gamma), \varepsilon/(1+\gamma)\}$ completes the proof. □

We will present the randomized self-reductions for problems $BP[\ell]$ s.t. $\ell \neq 1^k$ in Section 3.4.2. For $BP[1^k]$, we will prove that it is either easy to solve or easy to complement in Section 3.4.3.

After shrinking the universe, we complement the sets to reduce between two Basic Problems $BP[\ell_1]$ and $BP[\ell_2]$ according to the following lemma.

Lemma 3.4.3. (Reduction between different Basic Problems)
For two different labels $\ell_1, \ell_2 \in \{0,1\}^k$, given set families $\mathcal{S}_1, \ldots, \mathcal{S}_k$, let $\mathcal{S}'_1, \ldots, \mathcal{S}'_k$ be defined such that

$$\mathcal{S}'_i = \begin{cases} \left\{ S_i^{\complement} \mid S_i \in \mathcal{S}_i \right\}, & \text{if } \ell_1[i] \neq \ell_2[i] \\ \mathcal{S}_i, & \text{otherwise} \end{cases},$$

then, $(\exists S_1 \in \mathcal{S}_1) \ldots (\exists S_k \in \mathcal{S}_k)(\forall u)\psi_{\ell_1}$ iff $(\exists S'_1 \in \mathcal{S}'_1) \ldots (\exists S'_k \in \mathcal{S}'_k)(\forall u)\psi_{\ell_2}$.

The proof of correctness is straightforward.

Proof of Lemma 3.4.1. Pick $s'(m) = s(m)^{1/(6k)}$. Using Lemma 3.4.2, we shrink the universe to size $n'_u = s'(m)^5$. So the time complexity in this step is bounded by $O(m \cdot s'(m)^5)$, which is significantly less than $m^k/s(m)$ even if $k = 2$.

Let new instance size be m'. So $m' = m \cdot s'(m)^5$. Given that the constructed instance can be decided in time $m'^k/s(m')$, we get $m'^k/s(m') < (m(s(m)^{1/(6k)})^5)^k/s(m) < m^k/s(m)^{1/6}$. Thus, by the two-step fine-grained mapping reductions given by Lemma 3.4.2 and Lemma 3.4.3, we have an exact complexity reduction between any two Basic Problems, completing the proof for Lemma 3.4.1. □

3.4.2 Randomized Universe-Shrinking Self-Reduction of $BP[\ell]$ where $\ell \neq 1^k$

This section proves part of Lemma 3.4.2, by giving a randomized universe-shrinking self-reduction of $BP[\ell]$ where $\ell \neq 1^k$. The main idea is to divide the sets into large and small ones. For large sets, there are not too many of them in the sparse structure, so we can work on them directly. For small sets, we use a Bloom Filter mapping each element in U to some elements in U' at random, and then for each set on universe U, we compute the corresponding set on universe U'. Next we can decide the same problem on these newly computed sets, instead of sets on U. ([CIP02] used a similar technique in reducing from Orthogonal Range Search to the Subset Query problem.) Because the sets are small, it is unlikely that some elements in two different sets on U are mapped to the same element on U', bounding the error probability.

- **Step 1: Large sets.** Let $d = s(m)$. For sets of size at least d, we directly check if they are in any solutions. There are at most $O(m/d) = O(m/s(m))$ of such large sets. In the outer loop, we enumerate all large sets in $\mathcal{S}_1, \ldots, \mathcal{S}_k$. If their sizes are pre-computed, we can do the enumeration in $O(m/s(m))$. Assume the current large set is $S_i \in \mathcal{S}_i$. Because variables quantified by $\exists$ are interchangeable, we can interchange the order of variables, and let S_i be the outermost quantified variable S_1. On each such S_i (or S_1 after interchanging), we create a new formula ψ_{S_1} on variables $S_2, \ldots, S_k, u$ from formula ψ, by replacing $u \in S_1$ ($u \notin S_1$) by a unary relation on u. Then, we decide if the graph induced by $\mathcal{S}_2, \ldots, \mathcal{S}_k$ and U satisfies $(\exists S_2) \ldots (\exists S_k)(\forall u)\psi_{S_1}$, using the baseline algorithm, which takes time $O(m^{k-1})$ for each such large set S_1. Thus the overall running time is $O(m/s(m)) \cdot O(m^{k-1}) = O(m^k/s(m))$. If no

solution is found in this step, proceed to Step 2.

- **Step 2: Small sets.** Now we can exclude all the sets of size at least d. For sets of size smaller than d, we do the self-reduction to universe U' of size $n'_u = s(m)^5$. Let $t = s(m)$, and let $h\colon U \to U'^t$ be a function that independently maps each element $u \in U$ to t elements in U' at random. On set $S \subseteq U$, we overload the notation h by defining $h(S) = \bigcup_{u\in S} h(u)$. For all set families $\mathcal{S}_i$, we compute new sets $h(S_i)$ for all $S_i \in \mathcal{S}_i$. Then, we decide whether the new sets satisfy the following sentence, which is another $BP[\ell]$ problem:

$$(\exists S_1)\dots(\exists S_k)(\forall u)\bigvee_{i,\ell[i]=0}\neg(u\in h(S_i))\vee\bigvee_{i,\ell[i]=1}(u\in h(S_i))$$

  The size of the new instance is $O(nt) = O(m\cdot s(m))$, and the running time of the self-reduction is also $O(nt) = O(m\cdot s(m))$. So it is a fine-grained mapping reduction for any $k \geq 2$.

Figure 3.2 illustrates an example of the universe-shrinking self-reduction for $BP[01]$, where we look for S_1, S_2 so that $S_1 \subseteq S_2$. If they exist, then after the self-reduction, it is always true that $h(S_1) \subseteq h(S_2)$. Still, it might happen that some $S_1 \not\subseteq S_2$ but $h(S_1) \subseteq h(S_2)$. In this case, a false positive occurs. In $BP[00]$, a false negative may occur when there are two disjoint sets, but some elements in $S_1 \cap S_2$ are mapped to the same element in U'. Next we will analyze the error probability of this reduction.

Analysis. Because variables quantified by $\exists$ are interchangeable, w.l.o.g. for ℓ containing i ($i \geq 1$) zeros and $k-i$ ones, assume $BP[\ell]$ is defined by

$$(\exists S_1)\dots(\exists S_k)(\forall u)\left[\left(\bigvee_{j=1}^{i}(u\notin S_j)\right)\vee\left(\bigvee_{j=i+1}^{k}(u\in S_j)\right)\right],$$

equivalently, $(\exists S_1)\dots(\exists S_k)\left[\left(\bigcap_{j=1}^{i} S_j\right)\subseteq\left(\bigcup_{j=i+1}^{k} S_j\right)\right]$.

Let sets $A = \bigcap_{j=1}^{i} S_j$ and $B = \bigcup_{j=i+1}^{k} S_j$. Then the problem is to decide whether there exists $(S_1,\dots,S_k)$ so that $A \subseteq B$. After the self-reduction, let $A' = \bigcap_{j=1}^{i} h(S_j)$ and $B' = \bigcup_{j=i+1}^{k} h(S_j)$, and decide if there exists $(S_1,\dots,S_k)$ such that $A' \subseteq B'$.

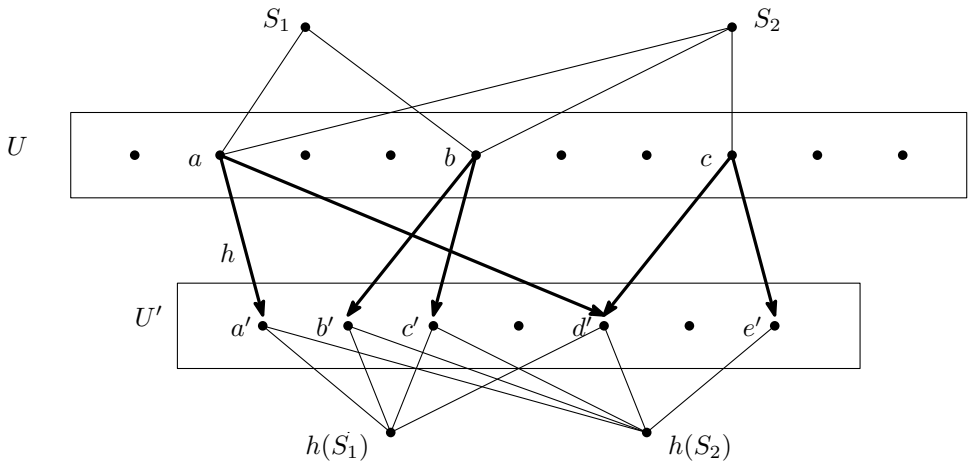


Figure 3.2: The universe-shrinking process. $S_1 = \{a,b\}$ and $S_2 = \{a,b,c\}$. After the mapping h, the new sets are $h(S_1) = \{a',b',c',d'\}$ and $h(S_2) = \{a',b',c',d',e'\}$.

- **False positive.** A false positive occurs when $\forall(S_1,\ldots,S_k), A \nsubseteq B$, but $\exists(S_1,\ldots,S_k), A' \subseteq B'$. For a fixed tuple $(S_1,\ldots,S_k)$ such that $A \nsubseteq B$, an error occurs when $h(u) \subseteq B'$ for all $u \in A - B$. The size of B' is at most kdt. So the error probability $\Pr[h(u) \subseteq B'] \leq (kdt/n'_u)^t = (ks(m) \cdot s(m)/s(m)^5)^t < s(m)^{-t}$. The size of $A - B$ is bounded by kd, so the probability $\Pr[\exists u \in A - B, h(u) \subseteq B'] \leq kd \cdot s(m)^{-t}$. There are $O(m^k)$ tuples of $(S_1,\ldots,S_k)$, so the total error probability is at most $O(m^k) \cdot kd \cdot s(m)^{-t} = O(m^k \cdot s(m)/s(m)^{s(m)})$, which is exponentially small.

- **False negative.** A false negative occurs when $\exists(S_1,\ldots,S_k), A \subseteq B$, but $\forall(S_1,\ldots,S_k), A' \nsubseteq B'$. Fix any tuple $(S_1,...,S_k)$ that satisfies $A \subseteq B$ in the original instance, and consider the distribution on the corresponding $h(S_1),..,h(S_k)$. By definition, $B' = \bigcup_{u \in B} h(u)$, and so contains $\bigcup_{u \in A} h(u)$. So if $A' \subseteq \bigcup_{u \in A} h(u)$, we will have $A' \subseteq B'$, and there will not be a false negative. If not, then there is some $u' \in A' = \bigcap_{j=1}^{i} h(S_j)$, such that $u' \notin \bigcup_{u \in A} h(u)$. Then for each $j \in \{1,\ldots,i\}$, in each S_j there is a $u_j \in S_j$ with $u' \in h(u_j)$, but not all u_j are identical. (Otherwise the $u_j \in A$, so $u' \in h(u_j) \subseteq \bigcup_{u \in A} h(u)$, contradicting $u' \notin \bigcup_{u \in A} h(u)$). In particular, this means that for some j_1, j_2, there are $u_{j_1} \in S_{j_1}, u_{j_2} \in S_{j_2}$, such that $h(u_{j_1}) \cap h(u_{j_2}) \neq \emptyset$. So the error probability is bounded by $k^2 \cdot \Pr[\exists(u_1 \in S_{j_1}, u_2 \in S_{j_2}), h(u_1) \cap h(u_2) \neq \emptyset]$. Because $|S_{j_1}|$ and $|S_{j_2}|$ are at most d, by Birthday Paradox, the probability is at most $O(k^2 d^2 t^2 / n'_u) =$

$O(s(m)^{-1})$. This is the upper bound of the error probability for the fixed $(S_1,\ldots,S_k)$ tuple. Then, the probability of the event "$\forall(S_1,\ldots,S_k), A' \nsubseteq B'$" is even smaller.

3.4.3 Deterministic Universe-Shrinking Self-Reduction of $BP[1^k]$

This section proves the remaining part of Lemma 3.4.2, by showing $BP[1^k]$ is either easy to solve or easy to complement. $BP[1^k]$ is the k-Set Cover problem, which decides whether there exist k sets covering the universe U. It is special in the Basic Problems: when n_u is small, the sets are easy to complement; when n_u is large, the problem is easy to solve.

- **Case 1: Large universe.** If $n_u > s(m)$, then in a solution of this problem, at least one set has size at least n_u/k. There are at most $m/(k/n_u) = O(m/s(m))$ such large sets, thus they can be listed in time $O(m/s(m))$, after pre-computation on the sizes of all sets. Our algorithm exhaustively searches all such large sets. And then, similarly to "Step 1" in Section 3.4.2, for each of the large sets, we run the baseline algorithm to find the remaining $k-1$ sets in the k-set cover, which takes time $O(m^{k-1})$. So the overall running time is $O(m/s(m)) \cdot O(m^{k-1}) = O(m^k/s(m))$.

- **Case 2: Small universe.** If $n_u \leq s(m)$, then we do not need a universe-shrinking self-reduction, because the universe is already small enough.

3.4.4 Hybrid Problem

Next we reduce general $MC(\exists^k\forall)$ problems to an intermediate problem called the Hybrid Problem, which is a combination of 2^k Basic Problems. Then by reducing from the Hybrid Problem to Basic Problems, we can set up a connection between $MC(\exists^k\forall)$ and OV.

Let $k \geq 2$. The input to the Hybrid Problem is:

1. Set families $\mathcal{S}_1 \ldots \mathcal{S}_k$ defined on universe U, where U is partitioned into 2^k disjoint sub-universes: $U = \bigcup_{\ell\in\{0,1\}^k} U_\ell$.

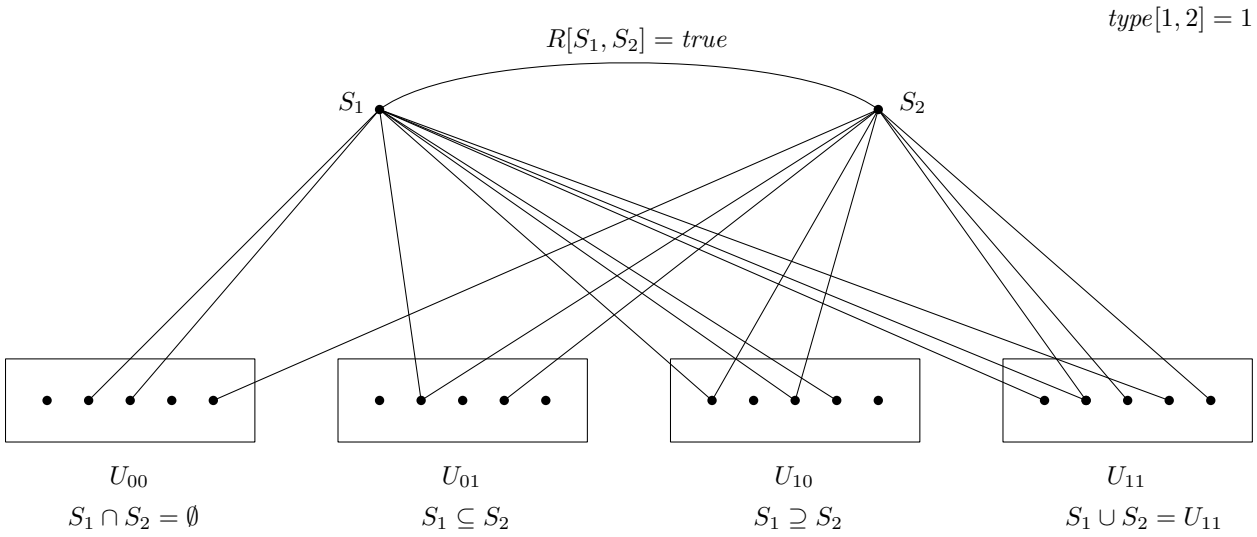


Figure 3.3: An example of a solution to a Hybrid Problem instance, when $k = 2$.

2. A binary relation R defined on pairs of sets from any two distinct set families. R is a symmetric relation ($R(S_i, S_j)$ iff $R(S_j, S_i)$).
3. *type* is binary string of length $\binom{k}{2}$, indexed by two integers $[i, j]$, s.t. $i, j \in \{1, \ldots, k\}$ and $i < j$.

The goal of the problem is to decide if there exist $S_1 \in \mathcal{S}_1, \ldots, S_k \in \mathcal{S}_k$ such that both of the following constraints are true:

(A) For each $\ell \in \{0,1\}^k$, $(S_1, \ldots S_k)$ is a solution of $BP[\ell]$ defined on sub-universe U_ℓ.

(B) For all pairs of indices $i, j \in \{1, \ldots, k\}, i < j$, we have that $R(S_i, S_j) = true$ iff $type[i, j] = 1$.

We let n be the sum of $|\mathcal{S}_1|, \ldots, |\mathcal{S}_k|$ and U, and let m be the number of tuples in all unary and binary relations. The Hybrid Problem is a first-order property on graphs with additional constraints. As usual, we assume all relations in the Hybrid Problem are sparse ($m \leq n^{1+o(1)}$).

Figure 3.3 shows a solution to a Hybrid Problem instance when $k = 2$. In sub-universes $U_{00}, U_{01}, U_{10}, U_{11}$, sets S_1 and S_2 are solutions of $BP[00]$(Set Disjointness), $BP[01]$(Set Containment), $BP[10]$(Set Containment in the reversed direction) and $BP[11]$(2-Set Cover), respectively. Furthermore, $type[1,2] = 1$ specifies that the predicate R on (S_1, S_2) must be true.

Intuition behind the Hybrid Problem. In the Hybrid Problem, the set families $S_1, \ldots, S_k$ encode the conditions on relations involving x_{k+1}, while the binary relation R and the *type*s encode

the conditions on relations not involving x_{k+1}. We mentioned in Section 3.3 that any first-order query containing two variables can be written in a "normal form", which is a combination of sparse and co-sparse relations. The Hybrid Problem is designed for separating sparse relations from co-sparse ones, for *all* pairs of variables in formula φ.

The relation between the pair of variables (x_i, x_{k+1}) where $1 \leq i \leq k$ can be either sparse or co-sparse. Because there are k such variables x_i, there are 2^k cases for a combination $((x_1, x_{k+1}), \ldots, (x_k, x_{k+1}))$. These cases correspond to the 2^k Basic Problems. In each Basic Problem, we deal with one of the 2^k cases.

For a relation between the pair of variables (x_i, x_j) where $1 \leq i < j \leq k$, it also can be either sparse or co-sparse. We use $type[i, j]$ to distinguish the two cases: when it is set to 1, we expect a sparse relation for (x_i, x_j), otherwise a co-sparse relation.

3.4.5 Reduction to Basic Problems

Lemma 3.4.4. *Let $s(m)$ be a non-decreasing function such that $2^{\Omega(\sqrt{\log m})} \leq s(m) < m^{1/5}$. Then, $(\textit{Hybrid Problem}, m^k/(s(m))^{1/6}) \leq_{EC} (k\textit{-OV}, m^k/(s(m)))$.*

Given an instance of the Hybrid Problem, we can do the following modification in time $O(m)$. For each pair of indices i, j where $1 \leq i < j \leq k$, we construct auxiliary elements depending on the value of $type[i, j]$.

• **Case 1:** If $type[i, j] = 0$, then if a pair $S_i \in \mathcal{S}_i$, $S_j \in \mathcal{S}_j$ occurs in a solution to the Hybrid Problem, then there should be no edge $R(S_i, S_j)$. Let ℓ be the length-k binary string where the i-th and j-th bits are zeros and all other bits are ones. For each edge $R(S_i, S_j)$ on $S_i \in \mathcal{S}_i$ and $S_j \in \mathcal{S}_j$, we add an extra element $u_{S_iS_j}$ in U_ℓ and let $u_{S_iS_j} \in S_i$, $u_{S_iS_j} \in S_j$. Thus, $S'_i \in \mathcal{S}_i$ and $S'_j \in \mathcal{S}_j$ can both appear in the solution only when for all $u_{S_iS_j}$, $(u_{S_iS_j} \notin S'_i) \vee (u_{S_iS_j} \notin S'_j)$, and this holds iff $R(S'_i, S'_j) = \textit{false}$.

• **Case 2:** If $type[i, j] = 1$, then in a solution to the Hybrid Problem, S_i and S_j should have an edge $R(S_i, S_j)$ between them. Let ℓ be the length-k binary string where the j-th bit is zero and all

other bits are ones. For each $S_j \in \mathcal{S}_j$, we add an extra element u_{S_j} in U_ℓ and let $u_{S_j} \in S_j$. For each edge $R(S_i, S_j)$, we let $u_{S_j} \in S_i$. Thus, $S_i' \in \mathcal{S}_i$ and $S_j' \in \mathcal{S}_j$ can both appear in the solution only when for all u_{S_j}, $(u_{S_j} \notin S_j') \vee (u_{S_j} \in S_i')$, and it holds iff $R(S_i', S_j') = \mathit{true}$.

After the above construction, we can drop the constraint (B) of the Hybrid Problem. We will ignore the relation R and *type* in the Hybrid Problem. The problem now is to decide whether there exists tuple $(S_1, \ldots, S_k)$ being a solution to all 2^k Basic Problems. Then we can use the reductions in Lemma 3.4.1 to reduce all these Basic Problems to $BP[0^k]$, and then combine the 2^k instances to a large $BP[0^k]$ instance. Because the reductions do not change the solutions $S_1, \ldots, S_k$, there exists a solution to the large $BP[0^k]$ instance iff there exists a solution simultaneously to all the Basic Problem instances. Let $U_\ell{}'$ be the sub-universe of the $BP[0^k]$ instance reduced from the $BP[\ell]$ sub-problem. $(S_1, \ldots, S_k)$ is a solution to all Basic Problems iff their intersection is empty on every sub-universe U_ℓ', iff their intersection is empty on universe $\bigcup_{\ell \in \{0,1\}^k} U_\ell'$, i.e., it is a solution of a $BP[0^k]$ instance.

Multiplying the error probability in the reductions between Basic Problems by 2^k, which is a constant number, and then taking a union bound, we get similar bounds of error probability for the Hybrid Problem.

3.4.6 Turing reduction from general $MC(\exists^k\forall)$ problems to the Hybrid Problem

The following lemma provides the last piece of the proof that sparse k-OV is complete for $MC(\exists^k\forall)$ under fine-grained Turing reductions. The result follows by combining this lemma with Lemma 3.4.4.

Lemma 3.4.5. *For any integer $k \geq 2$, any problem in $MC(\exists^k\forall)$ is linear-time Turing reducible to the Hybrid Problem, namely, $(MC(\exists^k\forall), T(m)) \leq_{EC} (\textit{Hybrid Problem}, T(O(m)))$.*

Consider the problem MC_φ where $\varphi = (\exists x_1)\ldots(\exists x_k)(\forall x_{k+1})\psi(x_1, \ldots, x_{k+1})$. An input graph G can be preprocessed in linear time to ensure that it is a $(k+1)$-partite graph on vertices

$V = (V_1, \ldots, V_{k+1})$, for example by creating $k+1$ copies of original vertex set.

W.l.o.g, we assume that for each occurrence of binary predicate $R_t(x_i, x_j)$, $i \leq j$. Let P_{k+1} be the set of unary and binary predicates in ψ that involve variable x_{k+1}, and let $P_{\overline{k+1}}$ denote the set of the other predicates not including x_{k+1}. A *partial interpretation* α for $P_{\overline{k+1}}$ is a binary string of length $|P_{\overline{k+1}}|$, that encodes the truth values assigned to all predicates in $P_{\overline{k+1}}$. For each i s.t. $1 \leq i \leq |P_{\overline{k+1}}|$, if the i-th predicate in $P_{\overline{k+1}}$ is assigned to *true*, then we set the i-th bit of α to one, otherwise we set it to zero. For a tuple $(v_1, \ldots, v_k)$, we say it *implies* α (denoted by $(v_1, \ldots, v_k) \models \alpha$) iff when $(x_1 \leftarrow v_1, \ldots, x_k \leftarrow v_k)$, the evaluations of all predicates in $P_{\overline{k+1}}$ are the same as the values specified by α.

For each $\alpha \in \{0,1\}^{P_{\overline{k+1}}}$, we create a distinct Hybrid Problem instance H_α. If any of the Hybrid Problems accepts, we accept. Let $\psi|_\alpha(x_1, \ldots, x_{k+1})$ be ψ after replacing all occurrences of predicates in $P_{\overline{k+1}}$ by their corresponding truth values specified by α. The following steps show how to create H_α from α and $\psi|_\alpha(x_1, \ldots, x_{k+1})$.

Step 1: Construction of sets.

We introduce *colors*, which are partial interpretations defined on some specific subsets of the predicates concerning variable x_{k+1}. We call them "colors" because they can be considered as a kind of labels on (v_i, v_{k+1}) pairs. For each $i \in \{1, \ldots, k\}$, we give all the unary predicated defined on x_i and binary predicates defined on (x_i, x_{k+1}) (including those on (x_{k+1}, x_i)) a canonical order. We use P_i to denote the set of these predicates for each i. Let a color be a partial interpretation for P_i, which is a binary string of length $|P_i|$, encoding the truth values assigned to all predicates in P_i. For each j s.t. $1 \leq j \leq |P_i|$, if the j-th predicate in P_i is assigned to *true*, then we set the j-th bit of the color to one, otherwise we set it to zero. For a color $c_i \in \{0,1\}^{|P_i|}$, we say $(v_i, v_{k+1}) \models c_i$ iff when $x_i \leftarrow v_i$ and $x_{k+1} \leftarrow v_{k+1}$, the values of all predicates in P_i are the same as the corresponding bits of c_i. We refer to the colors where all bits are zeros as the *background colors*. These colors are special because they correspond to interpretations where all predicates in P_i are false, i.e., we cannot directly go through all pairs (v_i, v_{k+1}) where $(v_i, v_{k+1}) \models 0^{|P_i|}$, since this is a co-sparse relation. So

we need to deal with these pairs separately.

For a vertex combination $(v_1, \ldots, v_{k+1})$ where $(v_i, v_{k+1}) \models c_i$ on all $1 \leq i \leq k$, the k-color-tuple $(c_1, \ldots, c_k)$ forms a *color combination*, which corresponds to truth values assigned to all the predicates in P_{k+1}.

For each $v_i \in V_i$ where $1 \leq i \leq k$, we create set S_{v_i} in the set family $\mathcal{S}_i$. For each $v_{k+1} \in V_{k+1}$, and each color combination $(c_1, \ldots, c_k)$ s.t. $c_i \in \{0,1\}^{|P_i|}$ and the values of all predicates specified by $(c_1, \ldots, c_k)$ make $\psi|_\alpha$ evaluate to *false* (in which case we say $(c_1, \ldots, c_k)$ does not satisfy $\psi|_\alpha$), we create an element $u_{(v_{k+1}, c_1, \ldots, c_k)}$ in U. We call a string $C \in \{0,1\}^k$ an *encoding* of a color combination $(c_1, \ldots, c_k)$ when on all indices $i \in \{1, \ldots, k\}$, $C[i] = 1$ iff $c_i = 0^{|P_i|}$. We put each element $u_{(v_{k+1}, c_1, \ldots, c_k)}$ in the sub-universe U_C iff C is an encoding of $(c_1, \ldots, c_k)$.

Next we will construct the sets. For each $v_i \in V_i$, let S_{v_i} be

$$S_{v_i} = \{u_{(v_{k+1}, c_1, \ldots, c_k)} \mid (c_1, \ldots, c_k) \text{ does not satisfy } \psi|_\alpha, \text{ and } ((c_i \neq 0^{|P_i|}, (v_i, v_{k+1}) \models c_i), \text{ or } (c_i = 0^{|P_i|}, (v_i, v_{k+1}) \not\models c_i = 0^{|P_i|}))\}.$$

To construct such sets, for each edge on (x_i, x_{k+1}) (and (x_{k+1}, x_i)), we do the following. Assume the current vertex pair is (v_i, v_{k+1}).

1. First, let set S_{v_i} contain all elements $u_{(v_{k+1}, c_1, \ldots, c_k)}$ in U where c_i is a fixed color such that $(v_i, v_{k+1}) \models c_i$, and the other colors c_j can be any string in $\{0,1\}^{|P_j|}$.
2. Next, let set S_{v_i} contain all elements $u_{(v_{k+1}, c_1, \ldots, c_k)}$ in U where $c_i = 0^{|P_i|}$ (here $(v_i, v_{k+1}) \not\models c_i = 0^{|P_i|}$ because there is some edge connecting v_i and v_{k+1}, meaning at least one bit in c_i is 1), and the other colors c_j can be any string in $\{0,1\}^{|P_j|}$.

In other words, in the sub-universe labeled by 0^k, which is made up of elements $u_{(v_{k+1}, c_1, \ldots, c_k)}$ such that none of the c_i equals $0^{|P_i|}$, and that $(c_1, \ldots, c_k)$ does not satisfy $\psi|_\alpha$, a set S_{v_i} contains an element $u_{(v_{k+1}, c_1, \ldots, c_k)}$ iff $(v_i, v_{k+1}) \models c_i$. On the other hand, in any sub-universe labeled by C where the i-th bit of C is 1, i.e. those are made up of elements $u_{(v_{k+1}, c_1, \ldots, c_k)}$ such that $c_i = 0^{|P_i|}$ and that $(c_1, \ldots, c_k)$ does not satisfy $\psi|_\alpha$, a set S_{v_i} contains an element $u_{(v_{k+1}, c_1, \ldots, c_k)}$ iff $(v_i, v_{k+1}) \not\models c_i = 0^{|P_i|}$.

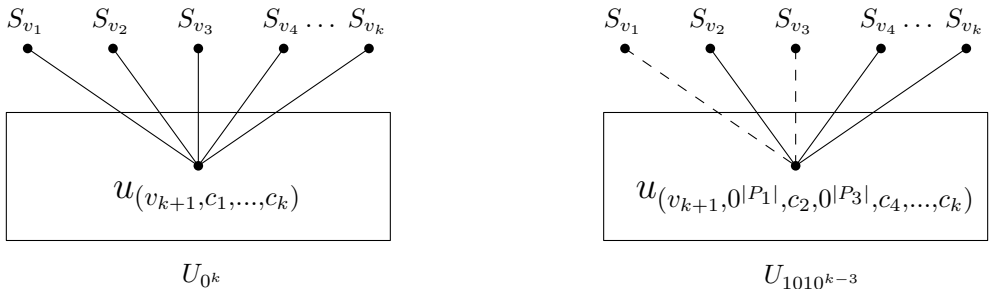


Figure 3.4: The formula is satisfied iff there exists $(S_{v_1}, S_{v_2}, \ldots, S_{v_k})$ so that there does not exist such an element u in any of the sub-universes.

Analysis. Now we show the above construction achieves constraint (A) in the definition of the Hybrid Problem.

• Assume that $(v_1, \ldots, v_k)$ does not satisfy $(\forall v_{k+1})\psi|_\alpha(x_1, \ldots, x_{k+1})$, i.e., there exists some $v_{k+1} \in V_{k+1}$ such that $\psi|_\alpha(v_1, \ldots, v_{k+1})$ is false. Then consider the specific color combination $(c_1, \ldots, c_k)$ where on each i, $(v_i, v_{k+1}) \models c_i$. So $(c_1, \ldots, c_k)$ does not satisfy $\psi|_\alpha(x_1, \ldots, x_{k+1})$. Thus there exists an element $u_{(v_{k+1},c_1,\ldots,c_k)}$ in U.

If none of the colors in combination $(c_1, \ldots, c_k)$ is the background color, then the encoding of $(c_1, \ldots, c_k)$ is the string 0^k. Thus, the element $u_{(v_{k+1},c_1,\ldots,c_k)}$ is in sub-universe U_{0^k}. By our construction, $u_{(v_{k+1},c_1,\ldots,c_k)}$ is contained in all of $S_{v_1}, \ldots, S_{v_k}$, as shown on the left side of Figure 3.4. (In the figure, the formula is satisfied iff there exists $(S_{v_1}, S_{v_2}, \ldots, S_{v_k})$ so that there does not exist such an element u in any of the sub-universes: the left figure illustrates the case where none of $c_1, \ldots, c_k$ is a background color. The right is the case where only c_1 and c_3 are background colors. The dashed lines stand for non-existing edges.) This is because when we went through all the edges, at the edge between (v_i, v_{k+1}), we put $u_{(v_{k+1},c_1,\ldots,c_k)}$ in S_{v_i}, since none of the colors is background. Thus $(\exists u \in U_{0^k})\left[\bigwedge_{i=1}^{k}(u \in S_{v_i})\right]$, so it is not the case that $(\forall u \in U_{0^k})\left[\bigvee_{i=1}^{k} \neg(u \in S_{v_i})\right]$, which means $S_{v_1}, \ldots, S_{v_k}$ is not a solution of $BP[0^k]$ on sub-universe U_{0^k}.

If some of the colors c_i in the color combination $(c_1, \ldots, c_k)$ equal the background color $0^{|P_i|}$, then in the encoding C of $(c_1, \ldots, c_k)$, $C[i] = 1$. Thus, the element $u_{(v_{k+1},c_1,\ldots,c_k)}$ is in the sub-universe U_C. By our construction, $u_{(v_{k+1},c_1,\ldots,c_k)}$ is contained in sets S_{v_i} for all indices i where c_i is not the background color $0^{|P_i|}$, and is not contained in sets S_{v_j} for all indices j where c_j is the

background color $0^{|P_j|}$. The latter case is because for each index j where c_j is the background color, there is no edge connecting the pair of vertices (v_j, v_{k+1}). So we did not put $u_{(v_{k+1},c_1,\ldots,c_k)}$ in S_{v_j}.

(The right side of Figure 3.4 demonstrates the example where c_1 and c_3 are the background colors while other colors are not.)

Thus

$$(\exists u \in U_C)\left[\bigwedge_{i\in\{1,\ldots,k\},C[i]=0}(u \in S_{v_i}) \wedge \bigwedge_{i\in\{1,\ldots,k\},C[i]=1}(\neg(u \in S_{v_i}))\right],$$

so it is not the case that

$$(\forall u \in U_C)\left[\bigvee_{i\in\{1,\ldots,k\},C[i]=0}(\neg(u \in S_{v_i})) \vee \bigvee_{i\in\{1,\ldots,k\},C[i]=1}(u \in S_{v_i})\right],$$

which means $S_{v_1},\ldots,S_{v_k}$ is not a solution of $BP[C]$ on sub-universe U_C.

• On the other hand, assume that $(v_1,\ldots,v_k)$ satisfies $(\forall v_{k+1})\psi|_\alpha(v_1,\ldots,v_{k+1})$. We claim that for *all* $\ell \in \{0,1\}^k$, $(S_{v_1},\ldots,S_{v_k})$ is a solution to Basic Problem $BP[\ell]$.

Consider the sub-universe U_C for each $C \in \{0,1\}^k$. If $C = 0^k$, i.e., the sub-universe is U_{0^k} corresponding to $BP[0^k]$, then none of the elements $u_{(v_{k+1},c_1,\ldots,c_k)}$ in U_{0^k} contains any background color among its $c_1,\ldots,c_k$. For the sake of contradiction, suppose there exists an element $u_{(v_{k+1},c_1,\ldots,c_k)}$ that is contained in all sets $S_{v_1},\ldots,S_{v_k}$. So by our construction of sets, for each $i \in \{1,\ldots,k\}$, $(v_i, v_{k+1}) \models c_i$. Recall that the color combination $(c_1,\ldots,c_k)$ in any element $u_{(v_{k+1},c_1,\ldots,c_k)}$ does not satisfy $\psi|_\alpha$. Then this means the vertex v_{k+1} does not satisfy $\psi|_\alpha(v_1,\ldots,v_k,v_{k+1})$, which leads to a contradiction.

Thus on $(S_{v_1},\ldots,S_{v_k})$, it is not the case that $(\exists u \in U_{0^k})\left[\bigwedge_{i=1}^k (u \in S_{v_i})\right]$, implying that $(S_{v_1},\ldots,S_{v_k})$ satisfies $(\forall u \in U_{0^k})\left[\bigvee_{i=1}^k \neg(u \in S_{v_i})\right]$. So it is a solution of the Basic Problem $BP[0^k]$ on sub-universe U_{0^k}.

If $C \neq 0^k$, for the sake of contradiction, suppose there exists an element $u_{(v_{k+1},c_1,\ldots,c_k)}$ such that among $S_{v_1},\ldots,S_{v_k}$, it is contained in set S_{v_i} iff $C[i] = 0$. Then by our construction of sets, this means for all i such that $C[i] = 0$, $(v_i, v_{k+1}) \models c_i$; while for all i such that $C[i] \neq 0$, $(v_i, v_{k+1}) \models 0^{|P_i|} = c_i$. Combining the two statements, for all i, $(v_i, v_{k+1}) \models c_i$. Recall again that the

color combination $(c_1,\ldots,c_k)$ in any element $u_{(v_{k+1},c_1,\ldots,c_k)}$ does not satisfy $\psi|_\alpha$. This implies the vertex v_{k+1} does not satisfy $\psi|_\alpha(v_1,\ldots,v_{k+1})$, which leads to a contradiction.

Thus on $(S_{v_1},\ldots,S_{v_k})$, it is not the case that

$$(\exists u \in U_C)\left[\bigwedge_{i\in\{1,\ldots,k\},C[i]=0}(u \in S_{v_i}) \wedge \bigwedge_{i\in\{1,\ldots,k\},C[i]=1}(\neg(u \in S_{v_i}))\right],$$

implying $(S_{v_1},\ldots,S_{v_k})$ satisfies

$$(\forall u \in U_C)\left[\bigvee_{i\in\{1,\ldots,k\},C[i]=0}(\neg(u \in S_{v_i})) \vee \bigvee_{i\in\{1,\ldots,k\},C[i]=1}(u \in S_{v_i})\right].$$

So it is a solution of the Basic Problem $BP[C]$ on sub-universe U_C.

In summary, there exists a tuple $(v_1,\ldots,v_k)$ such that $(\forall v_{k+1})\psi|_\alpha(v_1,\ldots,v_k,v_{k+1})$ holds true, iff there exist sets $(S_{v_1},\ldots,S_{v_k})$ such that for all $\ell \in \{0,1\}^k$, $(S_{v_1},\ldots,S_{v_k})$ is a solution of Basic Problem $BP[\ell]$ on sub-universe U_ℓ. Thus our reduction satisfies constraint (A) of the Hybrid Problem.

Step 2: Construction of *R* and *type*.

Next, we consider the predicates in $P_{\overline{k+1}}$, which are predicates unrelated to variable x_{k+1}. We create edges for predicate R according to the current partial interpretation α.

For a pair of vertices $v_i \in V_i$ and $v_j \in V_j$ where $1 \le i < j \le k$, we say (v_i,v_j) agrees with α if the evaluations of all predicates on (x_i,x_j) (including (x_j,x_i)) when $x_i \leftarrow v_i, x_j \leftarrow v_j$, equals the truth values of corresponding predicates specified by α.

• **Case 1: At least one predicate on (x_i,x_j) in α is true.** (i.e., (x_i,x_j) is in a sparse relation) For all edges (v_i,v_j) (including (v_j,v_i)) where $v_i \in V_i$ and $v_j \in V_j$ and $i < j \le k$, if (v_i,v_j) agrees with α, then we create edge $R(S_{v_i},S_{v_j})$. Finally we make $type[i,j] = 1$ in the Hybrid Problem H_α.

• **Case 2: All predicates on (x_i,x_j) in α are false.** (i.e., (x_i,x_j) is in a co-sparse relation) For all edges (v_i,v_j) (including (v_j,v_i)) where $v_i \in V_i$ and $v_j \in V_j$ and $i < j \le k$, if (v_i,v_j) does not agree with α, then we create edge $R(S_{v_i},S_{v_j})$. Finally we make $type[i,j] = 0$ in the Hybrid Problem H_α.

Analysis. We prove that (v_i, v_j) can appear in the solution of H_α only if when it agrees with α. If (v_i, v_j) does not agree with α, we should not let them be in any solution of H_α. This is done by the relation R and the string *type*.

Consider the two cases. If in α some predicates on (x_i, x_j) are true (i.e., set of tuples that agree with α is sparse), then in any (v_i, v_j) that agrees with α, there must be an edge in G connecting v_i and v_j. So we can add an edge (defined by relation R) on the corresponding sets S_{v_i}, S_{v_j} and require there must be such an edge in the solution (i.e., *type* being 1).

On the other hand, if all predicates on (x_i, x_j) in α are false (i.e., set of tuples agreeing with α is co-sparse), then in any (v_i, v_j) that agrees with α, there should not be any edge connecting v_i and v_j. In this case we turn to consider the tuples (v_i, v_j) that do not agree with α (which is a sparse relation, instead of co-sparse). We create edges on the corresponding sets S_{v_i}, S_{v_j} and require there must *not* be such an edge in the solution (i.e., *type* being 0).

Therefore, a tuple $(v_1, \ldots, v_k)$ implies α iff for all $i, j \in \{1, \ldots, k\}, i < j$, the truth value of relation $R(S_{v_i}, S_{v_j})$ equals whether $type[i, j] = 1$. Thus our reduction satisfies constraint (B) of the Hybrid Problem.

From the analyses of the two steps, we have justified that: there exists $(v_1, \ldots, v_k)$ so that $(v_1, \ldots, v_k) \models \alpha$, and $\psi|_\alpha$ holds for all $v_{k+1} \in V_{k+1}$, iff there exists $(S_{v_1}, \ldots, S_{v_k})$ being a solution to the Hybrid Problem H_α. Thus, if for any $\alpha \in \{0,1\}^{P_{k+1}}$, the Hybrid Problem H_α accepts, then there exists a solution $(v_1, \ldots, v_k)$ so that $\psi(v_1, \ldots, v_k, v_{k+1})$ holds for all $v_{k+1} \in V_{k+1}$. Otherwise there does not exist such a solution. From the above argument, we have proved the following claim.

Claim 3.4.1. *The two propositions are equivalent:*

(1) MC_φ has a solution $x_1 \leftarrow v_1, \ldots, x_k \leftarrow v_k$ such that $(\forall v_{k+1} \in V_{k+1})\psi(v_1, \ldots, v_{k+1})$ is satisfied.

(2) There exists an $\alpha \in \{0,1\}^{P_{k+1}}$ so that $(S_{v_1}, \ldots, S_{v_k}) \models \alpha$, and $S_{v_1}, \ldots, S_{v_k}$ is a solution to the Hybrid Problem H_α.

The running time of the whole reduction process is linear in the total number of edges in the graph, because the number of predicates is constant. Thus Lemma 3.4.5 follows.

3.5 Derandomization

We derandomize the reduction in Section 3.4 for the $k = 2$ case, so that the whole proof of Theorems 5 and 7 is determistic. The derandomization of the randomized universe-shrinking self-reduction uses the technique of nearly disjoint sets similar to the construction of pseudorandom generator by Nisan and Widgerson in [NW94].

In this section, for simplicity we use $SC(x)$ (resp. $SD(x)$) to denote Set Containment, a.k.a. the Basic Problem $BP[01]$ (resp. Set Disjointness, a.k.a. the Basic Problem [00] or Sparse OV) on universe of size x, and use HP for the Hybrid Problem.

Lemma 3.5.1. *For any $2^{\Omega(\sqrt{\log n})} \leq s < m^{1/3}$, there is a deterministic universe-shrinking self-reduction for SC such that $(SC(n), m^2/s) \leq_{EC} (SC(O(s^2 \log^2 n/\log^2 s), m^2/s^3 \frac{\log^2 n}{\log^2 s}))$.*

Lemma 3.5.2. *For any $2^{\Omega(\sqrt{\log n})} \leq s < m^{1/3}$, there is a deterministic universe-shrinking self-reduction for SD such that $(SD(n), m^2/s) \leq_{EC} (SD(O(s^2 \log n/\log s), m^2/s^3 \frac{\log n}{\log s}))$.*

The following reduction from the Hybrid Problem to Set Disjointness implies the model checking for any $\exists\exists\forall$ sentences on sparse structures can be reduced to moderate-dimension SD, and then to OV.

Lemma 3.5.3. *For any $2^{\Omega(\sqrt{\log n})} \leq s < m^{1/3}$, where m is the input size to the Hybrid Problem, there is a deterministic reduction algorithm such that*

$$(HP, m^2/s) \leq_{EC} (SD(O(s^2 \log^2 n/\log^2 s), m^2/s^7 \frac{\log^3 n}{\log^3 s})).$$

3.5.1 Proof of Lemma 3.5.1

This section presents the derandomization of the universe-shrinking self-reduction in Sections 3.4.2 for the Basic Problem $BP[01]$ (and equivalently $BP[10]$), i.e. when the corresponding Basic Problem is the Set Containment problem.

Pick $\ell = O(\log n/\log s)$ and prime number $q = O(s \log n/\log s)$, so that $s\ell < q$ and $q^\ell > n$. By Bertrand's postulate and that PRIMES is in P, we can find such a q in time $\tilde{O}(s \log n/\log s)$.

First, we use the algorithm in Section 3.4.2 to decide if there is a solution containing a set of size at least s, which takes time $O(m^2/s)$. So next we only consider sets of size smaller than s.

We create a new universe U' of size q^2. Let U' be $GF(q) \times GF(q)$. Let element u in universe U correspond to a unique polynomial p_u over $GF(q)$ of degree ℓ. The number of different polynomials is q^ℓ. Since $q^\ell > n$, the number of different polynomials is greater than the number of elements of U.

Let h be a hash function so that each element in U is mapped to a set $h(u) = \{\langle i, p_u(i)\rangle \mid i \in GF(q)\}$ of size q. For set $S \subseteq U$, define $h(S) = \bigcup_{u \in S} h(u)$. Finally, $\mathcal{S}_1' = \{h(S) | S \in \mathcal{S}_1\}$, and $\mathcal{S}_2'$ is constructed similarly. Then we decide the $SC(q^2)$ instance that takes $\mathcal{S}_1'$ and $\mathcal{S}_2'$ as input.

If $S_1 \subseteq S_2$, then $h(S_1) \subseteq h(S_2)$, and the call to the $SC(q^2)$ instance returns true.

If $S_1 \not\subseteq S_2$ for all sets, we need to show that for each element $u_1 \in S_1 \backslash S_2$, $|h(u_1) \cap h(S_2)| < q$. Then because $|h(u_1)| = q$, some element in $h(u_1)$ is not in $h(S_2)$, therefore $h(S_1) \not\subseteq h(S_2)$. To show $|h(u_1) \cap h(S_2)| < q$, observe that for each element $u_2 \in S_2$, the intersection $h(u_1) \cap h(u_2)$ has size at most ℓ, the degree of polynomial $p_{u_1} - p_{u_2}$. There are at most s elements in S_2, thus $|h(u_1) \cap h(S_2)| \leq s\ell < q$.

Thus, there exist $S_1 \subseteq S_2$ in the original instance iff there exist $h(S_1) \subseteq h(S_2)$ in the constructed instance.

The time to create the new set is $O(mq\ell)$, which is less than $O(m^2/s)$. And its size is $m' \leq mq$. Thus, if we can solve it in time $O(m'^2/\mathsf{poly}(s))$ where $s < m^\varepsilon$ for all $\varepsilon > 0$, we can solve it in time $O(m^2q^2/\mathsf{poly}(s)) = O(m^2/\mathsf{poly}(s))$.

3.5.2 Proof of Lemma 3.5.2

First, we use the algorithm in Section 3.4.2 to decide if there is a solution containing a set of size at least s, which takes time $O(m^2/s)$. So next we only consider sets of size smaller than s.

Let $\ell = O(\log n/\log s)$, and let q be a prime $\geq s^2\ell$, thus $q = O(s^2\ell) = O(s^2\frac{\log n}{\log s})$. So $q^\ell > n$. By Bertrand's postulate, we can find such a q in time $O(s^2\frac{\log n}{\log s})$. We create a universe U' of size q.

Each element u of U, which is a string of length $\log n$, can be viewed as the encoding of a polynomial p_u over $GF(q)$ of degree $\frac{\log n}{\log q} \leq \frac{\log n}{\log s} = \ell$.

Let a be an element in group $GF(q)$. For each element u in U, we let hash function $h_a(u) = p_u(a)$. For set $S \subseteq U$, define $h_a(S) = \bigcup_{u \in S}\{h_a(u)\}$. The algorithm in the outermost loop enumerates all elements $a \in GF(q)$. For each a, we compute $h_a(S)$ for all sets S in the input. Then we decide if there are two disjoint sets in the new instance. The algorithm makes q queries to $SD(q)$ instances of input size m, each taking time $T(m) = m^2/s^3 \frac{\log n}{\log s} = m^2/sq$, the running time for moderate-dimension OV. The total time is $qT(m) = O(m^2/s)$.

For each pair of different elements u and v in U, the number of elements a in $GF(q)$ so that $p_u(a) = p_v(a)$ is at most their degree $l < \log n$. Suppose $S_1 \in \mathcal{S}_1$ and $S_2 \in \mathcal{S}_2$ are a pair of disjoint sets. $h_a(S_1)$ and $h_a(S_2)$ are disjoint if all pairs of their elements are mapped to different elements in $GF(q)$. The total number of possible collisions is at most $s^2 \log n$. Because $q > s^2 \log n$, there exists at least one element a in $GF(q)$ so that all pairs of elements in S_1 and S_2 are mapped to different elements by h_a.

If there are no disjoint sets, then for each $S_1 \in \mathcal{S}_1$ and $S_2 \in \mathcal{S}_2$, $h(S_1 \cap S_2) \subseteq h(S_1) \cap h(S_2)$, so $h(S_1)$ and $h(S_2)$ are not disjoint. Thus, for every $a \in GF(q)$, the call to the $SD(\log n)$ instance returns false.

3.5.3 Hybrid Problem

In this section we combine the above two deterministic reductions to solve the Hybrid Problem, which yields a deterministic reduction for Theorem 5 and Theorem 7. Here we use a similar version of the Hybrid Problem as defined in Section 3.4.4 but without the relation R and the string $type$. More formally, we consider the Hybrid Problem defined as follows:

Problem HP

Input: $\mathcal{S}_1, \mathcal{S}_2$, each a set family of sets $S_i = A_i \cup B_i \cup C_i \cup D_i$ where A_i, B_i, C_i, D_i are subsets of disjoint universes U_A, U_B, U_C, U_D respectively.

Output: Whether there exist $S_i \in \mathcal{S}_1$ and $S_j \in \mathcal{S}_2$ so that

1. $A_i \cap A_j = \emptyset$ (Set Disjointness)
2. $B_i \subseteq B_j$ (Set Containment)
3. $C_i \supseteq C_j$ (Set Containment reversed)
4. $D_i \cup D_j = U_D$ (2-Set Cover)

The results in Section 3.4.4 can be applied to this version of Hybrid Problem, so that the model checking for first-order sentences of form $\exists\exists\forall$ can be reduced to the Hybrid Problem. More precisely, $(MC(\exists\exists\forall), T(O(m))) \leq (HP, T(m))$.

Proof of Lemma 3.5.3.

First, we decide if there is a solution containing a set of size at least s, as described in the previous subsections, using time $O(m^2/s)$. So next we only consider sets of size smaller than s.

If $|U_D| \geq 2s$, then for all pairs of i, j, D_i and D_j cannot cover U_D, so we return false. Otherwise for i and all j we create sets $U_D \backslash D_i$ and $U_D \backslash D_j$. So $D_1 \cup D_2 = U_D$ iff $(U_D \backslash D_i) \cap (U_D \backslash D_j) = \emptyset$. The resulting instance size is $O(ms)$.

Then, we use Lemma 3.5.1 self-reductions for Set Containment on the B's and C's, so the created sets B'_i, B'_j and C_i, C_j are on universes of size $O(s^2 \frac{\log^2 n}{\log^2 s})$. For each j, we create set $U_B \backslash B'_j$, so $B_i \subseteq B_j$ iff $B'_i \subseteq B'_j$ iff $B'_i \cap (U_B \backslash B'_j) = \emptyset$. Similarly for each i we create $U_C \backslash C'_i$, so $C_i \supseteq C_j$ iff $C'_i \supseteq C'_j$ iff $(U_C \backslash C'_i) \cap C'_j = \emptyset$. The resulting instance size is $O(m \cdot s^2 \frac{\log^2 n}{\log^2 s})$.

Finally, we use Lemma 3.5.2 self-reductions for Set Disjointness on the original A's. So in each call to the oracle, the created sets A'_i, A'_j are on universes of size $O(s^2 \frac{\log n}{\log s})$. For each i and each j, we create sets $S'_i = A'_i \cup B'_i \cup (U_C \backslash C'_i) \cup (U_D \backslash D_i)$ and $S'_j = A'_j \cup (U_B \backslash B'_j) \cup C'_j \cup (U_D \backslash D_j)$. By the argument above, $S'_i \cap S'_j = \emptyset$ iff $A'_i \cap A'_j = \emptyset$ and $B_i \subseteq B_j$ and $C_i \supseteq C_j$ and $D_i \cup D_j = U_D$. If $A_i \cap A_j = \emptyset$, then in at least one call to the oracle $A'_i \cap A'_j = \emptyset$ and thus the call will return true as long as the conditions on B, C, D's are satisfied. If $A_i \cap A_j \neq \emptyset$, all calls return false.

The size of the new instance is $O(m \cdot s^2 \frac{\log^2 n}{\log^2 s})$. In the reduction we make $s^2 \frac{\log n}{\log d}$ calls to the algorithm for Set Disjointness on small universe. Thus if $SD(O(s^2 \frac{\log^2 n}{\log^2 s}))$ has algorithms in time

$O(m^2/s^7\frac{\log^5 n}{\log^5 s})$, we get running time $s^2\frac{\log n}{\log d}\cdot O((m\cdot s^2\frac{\log^2 n}{\log^2 s})^2/s^7\frac{\log^5 n}{\log^5 s}) = O(m^2/s)$. □

This gives a reduction from the general first-order model checking problems to the Hybrid Problem.

3.5.4 Extending to More Quantifiers

The derandomization can be extended to quantifiers $k+1$ for integer $k \geq 2$. The reduction combines the reductions for Set Containment and Set Disjointness.

Recall from Section 3.4.1, a Basic Problem $BP[\ell]$ where $\ell \neq 1^k$ can be considered as deciding $(\exists S_1)\ldots(\exists S_k)(\forall u)\left[\left(\bigvee_{j=1}^{i}(u \notin S_j)\right) \vee \left(\bigvee_{j=i+1}^{k}(u \in S_j)\right)\right]$, or equivalently $(\exists S_1)\ldots(\exists S_k)$ $\left[\left(\bigcap_{j=1}^{i} S_j\right) \subseteq \left(\bigcup_{j=i+1}^{k} S_j\right)\right]$. for some i such that $0 \leq i \leq k$. Again, we map each element in U to a set of elements in a small universe U' by some function h, and thus map each set S in U to a set $h(S)$ in U'.

Let q_1, q_2 be the q defined in Sections 3.5.1 and 3.5.2 respectively. Here q_2 is a prime number larger than $s^{k-i}\ell$. For each element u, for each element $a \in GF(q_2)$ we map it to a set of tuples $h(u) = \{\langle u_{SC}, u_{SD}\rangle \mid u_{SC} \in h^{SC}(u), u_{SD} \in h_a^{SD}(u)\}$, where h^{SC} and h_a^{SD} are the functions h and h_a defined in Sections 3.5.1 and 3.5.2 respectively, and then we make a query for the $BP[\ell]$ instance created from the mapping h. Thus we make q_2 queries in all, and accept if at least one of the queries is accepted.

If there exist sets $S_1,\ldots,S_k$ such that $\bigcap_{j=1}^{i} S_j \subseteq \bigcup_{j=i+1}^{k} S_j$, by generalizing the analysis in Section 3.5.2, in at least one query, the set $\bigcap_{j=1}^{i} h(S_j)$ does not contain any element not in $h(\bigcap_{j=1}^{i} S_j)$. And by generalizing the analysis in Section 3.5.1, in each query, the set $\bigcup_{j=i+1}^{k} h(S_j) = h(\bigcap_{j=1}^{i} S_j)$ is always contained in $h(\bigcup_{j=i+1}^{k} S_j)$ which is contained in $\bigcup_{j=i+1}^{k} h(S_j)$. So we get the following reduction: $(BP[\ell](n), m^k/s) \leq_{EC} (BP[\ell](\mathsf{poly}(s)), m^k/\mathsf{poly}(s))$.

3.6 Extending to Hypergraphs

This section gives a reduction from $MC(\exists\exists\forall)$, i.e., the model checking for $\exists\exists\forall$ formulas on hypergraphs, to the model checking for $\exists\exists\forall$ formulas on graphs, where there are only unary and binary relations. We will prove the following lemma.

Lemma 3.6.1. *If $MC(\exists^k\forall)$ on graphs is solvable in time $T(m)$, then $MC(\exists^k\forall)$ on hypergraphs is solvable in $T(O(m)) + O(m^{k-1/2})$.*

For a three-quantifier formula $(\exists x)(\exists y)(\forall z)\ \psi(x,y,z)$ where $x \in X, y \in Y, z \in Z$, we prove that it can be decided in time $O(m^{3/2} + T(O(m)))$, where T is the running time for the model checking of three-quantifier formulas on *graphs*.

Let relation $N(x,y)$ be the edges of the Gaifman graph, which means $N(x,y) = \mathit{true}$ iff there exists some z such that there is a hyperedge $R_i(x,y,z) = \mathit{true}$ (the order of x,y,z can be interchanged). Note that each tuple in the relations contributes to only constantly many tuples of N. So $|N| = O(m)$, and we can construct N in linear time.

Let $\psi(x,y,z)$ be a quantifier-free formula. We define $\psi^*(x,y,z)$ be $\psi(x,y,z)$ where all occurrences of ternary predicates are replaced by *false*. Thus, it contains only unary and binary predicates. Formula $(\exists x)(\exists y)(\forall z)\psi(x,y,z)$ is equivalent to $(\exists x)(\exists y)(\forall z)[N(x,y) \wedge \psi(x,y,z)] \vee (\exists x)(\exists y)(\forall z)[\neg N(x,y) \wedge \psi^*(x,y,z)]$.

We can decide $(\exists x)(\exists y)(\forall z)[\neg N(x,y) \wedge \psi^*(x,y,z)]$ using the algorithm for graphs, because all relations are binary. To decide $(\exists x)(\exists y)(\forall z)[N(x,y) \wedge \psi(x,y,z)]$, we consider three types of x's and y's. Let $deg(x)$ be the degree of x in the Gaifman graph.

• **Type 1: $deg(x) \geq \sqrt{m}$.** It is similar to deciding "large sets" for Basic Problems in Section 3.4.2. In the outer loop, enumerate all such x's. For each x, we modify the model checking problem to an instance of FOP_2, by treating x as a constant. The number of such x's is at most $O(m/\sqrt{m}) = O(\sqrt{m})$, and deciding an FOP_2 problem runs in time $O(m)$. So the total running time is $O(\sqrt{m} \cdot m) = O(m^{3/2})$.

• **Type 2:** $deg(y) \geq \sqrt{m}$**.** Use the same method as above by exchanging the order of x and y. The running time is also $O(m^{3/2})$.

• **Type 3:** $deg(x) < \sqrt{m}$ **and** $deg(y) < \sqrt{m}$**.** Enumerate all pairs of such x's and y's. Then in the inner loop, we enumerate all their neighbors in Z. In this way, for each $z \in Z$ such that z is a neighbor of x or y, we can categorize it by the truth value of all predicates. For all other z's, we know all the predicates are false. Thus we can decide if all $z \in Z$ satisfy ψ. Because all these x's and y's are adjacent, the time for enumerating pairs of x and y is $O(m)$, and the time for enumerating all their neighbors in Z is $O(\sqrt{m})$. So the total running time is $O(\sqrt{m} \cdot m) = O(m^{3/2})$.

Thus, for each pair (x,y) where $N(x,y) = true$, we can decide the model checking for $(\forall z)\psi(x,y,z)$ in time $O(m^{3/2})$. For each pair (x,y) where $N(x,y) = false$, $(\forall z)\psi(x,y,z)$ is true iff $(\forall z)[\neg N(x,y) \wedge \psi^*(x,y,z)]$.

Similarly, for $MC(\exists^k\forall)$ problems where $\varphi = (\exists x_1)\ldots(\exists x_k)(\forall x_{k+1})\psi(x_1,\ldots,x_{k+1})$, we still consider the cases whether there exist some hyperedge between any pair of x_i, x_j, where $i, j \leq k$. We define relation $N(x_i,x_j) = true$ iff there exists some x_k such that there is some hyperedge containing vertices x_i, x_j. We also define $\psi^*(x_1,\ldots,x_{k+1})$ be $\psi(x_1,\ldots,x_{k+1})$ where all occurrences of predicates with arities greater than two are replaced by *false*. So

$$\begin{aligned}
\varphi =& (\exists x_1)\ldots(\exists x_k)(\forall x_{k+1}) \left[\bigvee_{i,j\in\{1,\ldots,k\}, i\neq j} (N(x_i,x_j) \wedge \psi(x_1,\ldots,x_{k+1}) \right] \\
& \vee \left[\left(\bigwedge_{i,j\in\{1,\ldots,k\}, i\neq j} \neg N(x_i,x_j) \right) \wedge \psi^*(x_1,\ldots,x_{k+1}) \right] \\
=& \bigvee_{i,j\in\{1,\ldots,k\}, i\neq j} \left[(\exists x_1)\ldots(\exists x_k)(\forall x_{k+1})[N(x_i,x_j) \wedge \psi(x_1,\ldots,x_{k+1})] \right] \\
& \vee (\exists x_1)\ldots(\exists x_k)(\forall x_{k+1}) \left[\left(\bigwedge_{i,j\in\{1,\ldots,k\}, i\neq j} \neg N(x_i,x_j) \right) \wedge \psi^*(x_1,\ldots,x_{k+1}) \right]
\end{aligned}$$

To decide $(\exists x_1)\ldots(\exists x_k)(\forall x_{k+1})[N(x_i,x_j) \wedge \psi(x_1,\ldots,x_{k+1})]$, we do exhaustive search on the $k-2$ variables other than x_i and x_j (which in essence is a quantifier-eliminating downward

reduction), which takes a factor of $O(m^{k-2})$ in the running time. Then we process the variables x_i, x_j, x_k in the same way as variables x, y, z in the three-quantifier problem, that takes time $O(m^{3/2})$. The total running time is $O(m^{k-1/2})$.

To decide$(\exists x_1)\dots(\exists x_k)(\forall x_{k+1})\left[\left(\bigwedge_{i,j\in\{1,\dots,k\},i\neq j}\neg N(x_i,x_j)\right)\wedge\psi^*(x_1,\dots,x_{k+1})\right]$, we can use the algorithm for $MC(\exists^k\forall)$ problems on graphs, because the new formula has only unary and binary relations.

3.7 Hardness of k-OV for $MC(\forall\exists^{k-1}\forall)$

In this section we present an exact complexity reduction from any $MC(\forall\exists^{k-1}\forall)$ problem to a $MC(\exists^k\forall)$ problem, establishing the hardness of k-OV for these problems. This reduction gives an extension of the reduction from Hitting Set to Orthogonal Vectors in [AWW16] to sparse structures.

Lemma 3.7.1. *For $k\geq 2$ and $s(m)$ a non-decreasing function such that $2^{\Omega(\sqrt{\log m})}\leq s(m)<m^{1/5}$, let $\varphi'=(\exists x_2)\dots(\exists x_k)(\forall x_{k+1})\psi(x_1,\dots,x_{k+1})$. There is an exact complexity reduction*

$$(MC_{(\forall x_1)\varphi'},\tfrac{m^k}{s(\sqrt{m})})\leq_{EC}(MC_{(\exists x_1)\varphi'},\tfrac{m^k}{s(m)}).$$

First, we show that in problem $MC_{(\exists x_1)\varphi'}$, if graph G satisfies $(\exists x_1)\varphi'$, then we can find a satisfying value v_1 for variable x_1 by binary search. We divide the set V_1 into two halves, take each half of V_1 and query whether $(\exists x_1)\varphi'$ holds true on the graph induced by this half of V_1 together with the original sets $V_2,\dots,V_{k+1}$. If any half of V_1 works, then we can shrink the set of candidate values for x_1 by a half, and then recursively query again, until there is only one vertex v_1 left. So it takes $O(\log|V_1|)$ calls to find a v_1 in some solution. This means as long as there is a solution for $MC_{\exists x_1\varphi'}$, we can find a satisfying v_1 efficiently, with $O(\log m)$ queries to the decision problem.

Step 1: Large degree vertices. Let $t=m^{(k-1)/k}$. We deal with vertices in $V_1\dots V_k$ with degree greater than t. There are at most $m/t=m^{1/k}$ such vertices. After pre-computing the sizes of all the sets, these large sets can be listed in time $O(m^{1/k})$.

• **Step 1-1: Large degree vertices in V_1.** For each vertex $v_1 \in V_1$ with degree at least t, we create a formula ψ_{v_1} on variables $x_2, \ldots, x_{k+1}$ from formula ψ, by replacing occurrences of unary predicates in ψ on x_1 by constants, and replacing occurrences of binary predicates involving x_1 by unary predicates on the other variables. Then we check if the graph induced by $V_2, \ldots, V_{k+1}$ satisfies $(\exists x_2)\ldots(\exists x_k)(\forall x_{k+1})\psi_{v_1}(x_2, \ldots, x_{k+1})$ by running the baseline algorithm in time $O(m^{k-1})$. If the new formula is satisfied, then we mark v_1 as "good". The total time complexity is $O(m^{1/k}) \cdot O(m^{k-1}) = O(m^{k-1+1/k})$.

• **Step 1-2: Large degree vertices in $V_2, \ldots, V_k$.** Now we exhaustively search over all vertices $v_1 \in V_1$ with degree less than t in the outermost loop. For each such v_1, we find out all vertices $v_i \in V_i$ for $2 \leq i \leq k$, with degree at least t. Again, there are at most $O(m^{1/k})$ of them.

◦ **Case 1: $k > 2$.** Because variables x_2 through x_k are all quantified by $\exists$, we interchange their order so that the variable x_i becomes the second-outermost variable x_2 (and thus the current v_i becomes v_2). Next, for each v_1 and v_2 we construct a new formula $\psi_{(v_1,v_2)}$ on variables $x_3, \ldots, x_{k+1}$, by regarding x_1 and x_2 as fixed values v_1 and v_2, and then modify ψ into $\psi_{(v_1,v_2)}$ similarly to the previous step. Again, we run the baseline algorithm to check whether the graph induced by the current $V_3, \ldots, V_{k+1}$ satisfies $(\exists x_3)\ldots(\exists x_{k+1})\psi_{(v_1,v_2)}(x_3, \ldots, x_{k+1})$, using time $O(m^{k-2})$. If the formula is satisfied, we mark the current v_1 as "good". The total time complexity is $O(m \cdot m^{1/k}) \cdot (m^{k-2}) = O(m^{k-1+1/k})$.

◦ **Case 2: $k = 2$.** For each vertex v_2, we mark all the v_1's satisfying $\forall x_3 \psi(v_1, v_2, x_3)$ as "good". This can be done in $O(m)$ using the algorithm for the base case of the baseline algorithm, by treating the current v_2 as constant. So this process runs in time $O(m^{1/k}) \cdot O(m) = O(m^{3/2})$.

If not all vertices in V_1 with degree at least t are marked "good", we reject. Otherwise, go to Step 2.

Step 2: Small degree vertices. First we exclude all the large vertices from the graph. Then for the "good" vertices found in the previous step, we also exclude them from V_1.

Now all vertices have degree at most t. In each of $V_1, \ldots, V_k$, we pack their vertices into groups where in each group the total degree of vertices is at most t. Then the total number of groups is bounded by $O(m/t)$.

For each k-tuple of groups $(G_1, \ldots, G_k)$ where $G_1 \subseteq V_1, \ldots, G_k \subseteq V_k$, we query the oracle deciding $MC_{(\exists x_1)\varphi'}$ whether it accepts on the subgraph induced by vertices in $G_1, \ldots, G_k$. If so, then we find a vertex v_1 in V_1 so that when $x_1 \leftarrow v_1$, the current subgraph satisfies φ'. We remove this v_1 from V_1. Then we repeat this process to find new satisfying v_1's in V_1, and remove these v_1's from V_1. When V_1 is empty, or when no new solution is found after all group combinations are exhausted, the algorithm terminates. If in the end V_1 is empty, then all $v_1 \in V_1$ are in solutions of $MC_{\exists x_1 \varphi'}$, so we accept. Otherwise we reject.

Each query to $MC_{\exists x_1 \varphi'}$ has size $m' = O(kt) = O(t)$. Because the number of different k-tuples of groups is $O(m/t)^k = O((m/t)^k)$, the number of queries made is $O((m/t)^k + |V_1|) \cdot O(\log m) = O((m^{1/k})^k + |V_1|) \cdot O(\log m) = O(m \log m)$ times. If $MC_{\exists x_1 \varphi'}$ on input size m' is solvable in time $O(m'^k / s(m'))$, then the running time for $MC_{\forall x_1 \varphi'}$ is $O(m \log m) \cdot O(m'^k / s(m')) = O(m \log m \cdot (m^{(k-1)/k})^k / s(m^{(k-1)/k}) \leq O(m^k / s(\sqrt{m}) \cdot \log m)$. The exponent of m is less than k. Thus this is a fine-grained Turing reduction. Lemma 3.7.1 follows.

Note that this reduction works not only on graphs but also on structures with relations of arity greater than two.

3.8 Improved Algorithms

In this section we present an algorithm solving Sparse OV in time $m^2 / 2^{\Theta(\sqrt{\log m})}$. It is based on the papers [AWY15, CW16], which solves dense OV for vectors of dimension d in time $n^{2-\Omega(1/\log(d/\log n))}$.

Consider the universe-shrinking self-reduction for Sparse OV (Set Disjointness) in Section 3.5. We show that for $s(m) = 2^{\Theta(\sqrt{\log m})}$, by the above theorem, this reduction gives an algorithm in time $m^2 / 2^{\Theta(\sqrt{\log m})}$. We deal with large sets and small sets separately. For sets of size at least $s(m)$, we check if each of them is disjoint with some other set. From the argument for large sets, this is in time $m^2 / s(m)$. Then, for sets of size less than $s(m)$, we use the universe-shrinking self-reduction to reduce this instance to a Sparse OV instance on universe of size

$s(m)^{\frac{5}{6k}}$ (in which case $k=2$). Using the algorithm from [AWY15, CW16], we can solve it in time $n^{2-\Theta(1/\log(s(m)^{\frac{5}{6k}}))} \leq m^{2-\Theta(1/\log(s(m))} \leq m^2/2^{\Theta(\log m/\log s(m))} = m^2/2^{\Theta(\sqrt{\log m})}$. So the total running time is bounded by $m^2/2^{\Theta(\sqrt{\log m})}$.

By the above argument and Theorem 5, since all the Basic Problems are solvable in time $m^2/2^{\Theta(\sqrt{\log m})}$, so is any other problem in $MC(\exists\exists\forall)$. The reduction from $MC(\forall\exists\forall)$ to $MC(\exists\exists\forall)$ in Section 3.7 gives $2^{\Theta(\sqrt{\log m})}$ savings for $MC(\forall\exists\forall)$ problems. Reducing to three-quantifier case by brute-forcing over the first $k-2$ variables we get Theorem 7, that states all FOP_{k+1} problems can be solved in $m^k/2^{\Theta(\sqrt{\log m})}$ time.

3.9 Baseline and Improved Algorithms

In this section, we first present a baseline algorithm for $MC(k+1)$ that runs in time $O(n^{k-1}m)$, which also implicitly gives us a quantifier-eliminating downward reduction from any $MC(k+1)$ problem to $MC(k)$ problems for $k \geq 2$. Then, we show how to get an improved algorithm in time $m^k/2^{\Theta(\sqrt{\log m})}$ using our reductions and the result by [AWY15, CW16]. Finally, we present the algorithms for some specific quantifier structures in $O(m^{3/2})$, so that these problems are easy cases in first-order property problems.

3.9.1 Baseline Algorithm for First-Order Properties

This section gives an $O(n^{k-1}m)$ time algorithm solving FOP_{k+1} with any quantifier structure for $k \geq 1$, thus proving Lemma 3.9.1.

Lemma 3.9.1. (Quantifier-eliminating downward reduction for FOP_{k+1})
Let the running time of FOP_{k+1} *on graphs of n vertices and m edges be* $T_k(n,m)$. *We have the*

recurrence

$$T_k(n,m) \leq n \cdot T_{k-1}(n, O(m)) + O(m), \textit{ for } k \geq 2.$$
$$T_1(n,m) = O(m).$$

By this lemma, if all problems in FOP_k have algorithms in time $T(n,m)$, then any problem in FOP_{k+1} can be solved in time $n \cdot T(n,m)$.

Base Case. We prove that when $k = 1$, $T_k(n,m) = m$. For each $v_1 \in V_1$, the algorithm computes $\#(v_1) = |\{v_2 \in V_2 \mid (v_1, v_2) \models \psi\}|$. Thus we can list the sets of v_1 s.t. $\#(v_1) > 0$ (if the inner quantifier is $\exists$), or those that satisfy $\#(v_1) = |V_2|$ (if it is $\forall$).

Let there be p_1 different unary predicates on v_1 and p_2 different unary predicates on v_2. We partition the universes V_1 and V_2 respectively into 2^{p_1} and 2^{p_2} subsets, based on the truth values of all the unary predicates of the corresponding variable. The number of different pairs of subsets is a constant. Each time, we pick a pair consisting of one subset from V_1 and one subset from V_2, and replace the unary predicates by constants. In this way, we can just consider binary predicates in the following argument.

Let $\bar{\psi}(v_1, v_2)$ be the formula where each occurrence of each negated binary relation $R_i(v_1, v_2)$ is replaced by *false*. We enumerate all tuples (v_1, v_2) connected by at least one edge. For each tuple, we evaluate $\psi(v_1, v_2)$ and $\bar{\psi}(v_1, v_2)$. Let

$$\#\psi(v_1) = \sum_{v_2 \text{ adjacent to } v_1} ([\psi(v_1, v_2) = \textit{true}] - [\bar{\psi}(v_1, v_2) = \textit{true}])$$

(in which the brackets are Iverson brackets). It can be computed by enumerating all tuples (v_1, v_2) connected by at least one edge. Next, because in $\bar{\psi}$ there are no occurrences of negated binary predicates, we can compute

$$\#\bar{\psi}(v_1) = \text{The number of } v_2 \text{ s.t. } \bar{\psi}(v_1, v_2) \text{ holds}$$

by first enumerating all tuples (v_1, v_2) connected by at least one edge and checking if $\bar{\psi}(v_1, v_2)$

holds, and then considering the number of non-neighboring v_2's for each v_1, if being a non-neighbor of v_1 also makes $\bar{\psi}(v_1, v_2)$ true. Finally, let $\#(v_1) = \#\psi(v_1) + \#\bar{\psi}(v_1)$.

This algorithm is correct, because whenever a pair (v_1, v_2) satisfies $\psi(v_1, v_2)$, there are two cases. The first is that there exists an edge between v_1 and v_2. In this case, when we enumerate all edges, $[\psi(v_1, v_2) = \mathit{true}]$ equals one and $[\bar{\psi}(v_1, v_2) = \mathit{true}]$ equals its contribution to $\#\bar{\psi}(v_1)$. On the other hand, if there does not exist an edge between v_1 and v_2, then the contribution of (v_1, v_2) to $\#\psi(v_1)$ is 0 and to $\#\bar{\psi}(v_1)$ is 1.

Whenever a pair (v_1, v_2) does not satisfy $\psi(v_1, v_2)$, there are also two cases. If there exists an edge between v_1 and v_2, when we enumerate all edges, $[\psi(v_1, v_2) = \mathit{true}]$ equals zero and $[\bar{\psi}(v_1, v_2) = \mathit{true}]$ equals its contribution to $\#\bar{\psi}(v_1)$. On the other hand, if there does not exist an edge between v_1 and v_2, the contributions of (v_1, v_2) to $\#\psi(v_1)$ and to $\#\bar{\psi}(v_1)$ are both 0.

Inductive Step. For $k \geq 2$, we give a quantifier-eliminating downward reduction, thus proving the recurrence relation. Assume $\varphi = (Q_1 x_1) \ldots (Q_{k+1} x_{k+1}) \psi(x_1, \ldots, x_{k+1})$ For each $v_1 \in V_1$, create new formula $\varphi_{v_1} = (Q_2 x_2) \ldots (Q_{k+1} x_{k+1}) \psi(x_2, \ldots, x_{k+1})$, and in ψ we replace each occurrence of unary predicate $R_i(x_1)$ with a constant $R_i(v_1)$, and replace each occurrence of binary predicate $R_i(x_1, x_j)$ (or $R_i(x_j, x_1)$) with unary predicate $R'_i(x_j)$ whose value equals $R_i(v_1, x_j)$ (or $R_i(x_j, v_1)$), etc. Our algorithm enumerates all $v_1 \in V_1$, and then computes if the graph induced by $V_2, \ldots, V_{k+1}$ satisfies φ_{v_1}. If x_1 is quantified by $\exists$, we accept iff any of them accepts. Otherwise we accept iff all of them accept. The construction of φ_{v_1} takes time $O(m)$. The created graph has $O(n)$ vertices and $O(m)$ edges. Thus the recursion follows.

This process is a quantifier-eliminating downward reduction from an FOP_{k+1} problem to an FOP_k problem. It makes $O(m)$ queries, each of size $O(m)$. Then if problems in FOP_k are solvable in time $O(m^{k-1-\varepsilon})$, then problems in FOP_{k+1} are solvable in time $m \cdot O(m^{k-1-\varepsilon}) = O(m^{k-\varepsilon})$. This quantifier-eliminating downward reduction implies that if all FOP_k have $T(n, m)$ time algorithms, then all FOP_{k+1} problems have $n \cdot T(n, m)$ time algorithms.

From the recursion and the base case, we have the running time $O(n^{k-1} m)$ by induction.

The quantifier-eliminating downward reduction from FOP_{k+1} to FOP_3 in Lemma 3.9.1 also works for hypergraphs. We exhaustively search the first $k-2$ quantified variables, and by replacing the occurrences of these variables by constants in the formula, we can reduce the arities of relations. After the reduction, we get a hypergraph of max arity at most three.

3.9.2 Algorithms for Easy Cases

In this section we show that any $(k+1)$-quantifier problem with a quantifier sequence ending with $\exists\exists$ or $\forall\forall$ is solvable in time $O(m^{k-1/2})$. First of all, we use the quantifier-eliminating downward reduction to reduce the problem to a FOP_3 problem. Then from the next subsections we see that these problems are solvable in $O(m^{3/2})$. [Wil16] shows improved algorithms that run in time $O(m^{1.41})$ for detecting triangles and detecting induced paths of length 2, which are special cases of $MC(\exists\exists\exists)$.

Lemma 3.9.2. *Problems in $MC(\exists\exists\exists)$, $MC(\forall\forall\forall)$, $MC(\forall\exists\exists)$ and $MC(\exists\forall\forall)$ are solvable in $O(m^{3/2})$.*

In the first two subsections, we consider when the input structures are graphs. Then in the last subsection, we consider the cases when the input structures have higher arity relations.

Problems in $MC(\exists\exists\exists)$ and $MC(\forall\forall\forall)$

For problems in $MC(\forall\forall\forall)$, we decide its negation, which is a $MC(\exists\exists\exists)$ problem.

We define nine *Atomic Problems*, which are special FOP_3 problems. Let the Atomic Problem labeled by ℓ to be $MC_{(\exists x\in X)(\exists y\in Y)(\exists z\in Z)\,\psi_\ell}$, and referred to as $\Delta[\ell]$. It is defined on a tripartite graph on vertex sets (X,Y,Z), whose edge sets are E_{XY},E_{YZ},E_{XZ} defined on $(X,Y),(Y,Z),(X,Z)$ respectively. The graph is undirected, i.e., E_{XY},E_{YZ} and E_{XZ} are symmetric relations. For simplicity we define an edge predicate E so that $E(v_1,v_2)$ is true iff there is an edge in any of E_{XY},E_{YZ},E_{XZ} connecting (v_1,v_2) or (v_2,v_1). Besides, we use $deg_Y(x)$ to denote the number of x's neighbors in Y.

The ψ_ℓ for all Atomic Problems are defined in Table 3.1. For problem MC_φ where $\varphi = (\exists x\in X)(\exists y\in Y)(\exists z\in Z)\,\psi(x,y,z)$, we write ψ as a DNF, and split the terms. Then we decide if

Table 3.1: Atomic Problems

$\psi_2 = E(x,y) \wedge E(x,z)$	$\psi_{2+} = E(x,y) \wedge E(x,z) \wedge E(y,z)$	$\psi_{2-} = E(x,y) \wedge E(x,z) \wedge \neg E(y,z)$
$\psi_1 = E(x,y) \wedge \neg E(x,z)$	$\psi_{1+} = E(x,y) \wedge \neg E(x,z) \wedge E(y,z)$	$\psi_{1-} = E(x,y) \wedge \neg E(x,z) \wedge \neg E(y,z)$
$\psi_0 = \neg E(x,y) \wedge \neg E(x,z)$	$\psi_{0+} = \neg E(x,y) \wedge \neg E(x,z) \wedge E(y,z)$	$\psi_{0-} = \neg E(x,y) \wedge \neg E(x,z) \wedge \neg E(y,z)$

there is a term so that there exist x, y, z satisfying this term. On each term t, which is a conjunction of predicates and negated predicates, we work on the induced subgraph whose vertices satisfy all the positive unary predicates and falsify all the negated unary predicates defined on them in t. Then we can remove all unary predicates from the conjunction, which is now a conjunction of binary predicates or their negations. (If the conjunction is a single predicate or a single negated predicate, then we can deal with it easily, so we don't consider this case here.) If we define $E(x,y) = \bigwedge_{R \text{ is a positive binary predicate in } t} R(x,y) \wedge \bigwedge_{R \text{ is a negative binary predicate in } t} \neg R(x,y)$, and define $E(y,z)$ and $E(x,z)$ similarly, then t becomes equivalent with some Atomic Problem, or a disjunction of Atomic Problems (because variables y and z are interchangeable, the Atomic Problems and their disjunctions cover all possible cases).

In our algorithm for each problem $\Delta[\ell]$, instead of deciding the existence of satisfying x, y, z, we consider these problems as counting problems, where for each x we compute

$$\#_\ell(x) = |\{(y,z) \mid x,y,z \text{ satisfy } \psi_\ell\}|.$$

Problems $\Delta[2], \Delta[1], \Delta[0]$ can be computed straightforwardly.

- In $\Delta[2]$, $\#_2(x) = deg_Y(x) \times deg_Z(x)$.
- In $\Delta[1]$, $\#_1(x) = deg_Y(x) \times (|Z| - deg_Z(x))$.
- In $\Delta[0]$, $\#_0(x) = (|Y| - deg_Y(x)) \times (|Z| - deg_Z(x))$.

Next we show for labels $\ell \in \{2+, 1+, 0+, 2-, 1-, 0-\}$, problems $\Delta[\ell]$ can be computed in $O(m^{3/2})$.

Algorithm 1 solves $\Delta[2+]$,that is, for each x, counting the number of triangles that contain x. The first part of the algorithm only considers small degree y. On each iteration of the outer loop, the inner loop is run for at most $\sqrt{m}$ times. The second part only considers large degree y. Because

Algorithm 1: $\Delta[2+]$

```
for all (x,y) ∈ E_XY do
    // Small degree y
    if deg_Z(y) ≤ √m then
        for all z s.t. (y,z) ∈ E_YZ do
            if (x,z) ∈ E_XZ then
                #_{2+}(x) ← #_{2+}(x) + 1
for all y ∈ Y s.t. deg_Z(y) > √m do
    // Large degree y
    for all (x,z) ∈ E_XZ do
        if (x,y) ∈ E_XY and (y,z) ∈ E_YZ then
            #_{2+}(x) ← #_{2+}(x) + 1
if #_{2+}(x) > 0 for some x ∈ X then
    Accept
else
    Reject
```

there are at most $\sqrt{m}$ of them, the outer loop is run for at most $\sqrt{m}$ times. Therefore the running time of the algorithm is $O(m^{3/2})$.

Algorithm 2 solves $\Delta[1+]$, which for each x counts $(x-y-z)$ paths where there is no edge between x and z. The first part is similar as $\Delta[2+]$. The second part first over-counts $(x-y-z)$ paths for all large degree y without restricting the edge between x and z, and then counts the number of over-counted cases in order to exclude them from the final result. In the first block, the inner loop is run for at most $\sqrt{m}$ times for each edge in E_{XY}. The second block takes time $O(m)$. The outer loop of the third block is run for at most $\sqrt{m}$ times, because there are at most $\sqrt{m}$ sets with degree at least $\sqrt{m}$. So in all, the running time is $O(m^{3/2})$.

For $\Delta[0+]$, we first compute $\#_{2+}(x)$ which is the result of $\Delta[2+]$, and then compute $\#_{1+}(x)$ and $\#'_{1+}(x)$, which are results of $\Delta[1+]$ on vertex sets (X,Y,Z) and (X,Z,Y) respectively. Finally let $\#_{0+}(x) \leftarrow |E_{YZ}| - (\#_{2+}(x) + \#_{1+}(x) + \#'_{1+}(x))$.

$\#_{2-}(x)$, $\#_{1-}(x)$, $\#_{0-}(x)$ can be computed by respectively taking the differences of $\#_2(x)$, $\#_1(x)$, $\#_0(x)$ and $\#_{2+}(x)$, $\#_{1+}(x)$, $\#_{0+}(x)$.

Algorithm 2: $\Delta[1+]$

```
for all (x,y) ∈ E_XY do
    // Small degree y
    if deg_Z(y) ≤ √m then
        for all z s.t. (y,z) ∈ E_YZ do
            if (x,z) ∉ E_XZ then
                #_{1+}(x) ← #_{1+}(x) + 1
for all (x,y) ∈ E_XY do
    // Large degree y
    if deg_Z(y) ≥ √m then
        // Over-counting
        #_{1+}(x) = #_{1+}(x) + deg_Z(y)
for all y ∈ Y s.t. deg_Z(y) > √m do
    for all (x,z) ∈ E_XZ do
        if (x,y) ∈ E_XY and (y,z) ∈ E_YZ then
            #_{1+}(x) ← #_{1+}(x) − 1 // if we just over-counted the pair (y,z),then we
                exclude the pair by subtracting one.
if #_{1+}(x) > 0 for some x ∈ X then
    Accept
else
    Reject
```

Problems in $MC(\forall\exists\exists)$ and $MC(\exists\forall\forall)$

For problems in $MC(\exists\forall\forall)$, we decide its negation, which is a $MC(\forall\exists\exists)$ problem.

For problem MC_φ where $\varphi = (\forall x \in X)(\exists y \in Y)(\exists z \in Z)\,\psi(x,y,z)$, we use the same algorithm to compute $\#_\ell(x)$ for all $x \in X$. If the value of $\#_\ell(x)$ is greater than zero for all $x \in X$, then we accept, otherwise reject. Again, we write ψ as a DNF, and split the terms. By the same argument as the previous lemma, we transform the problem to a disjunction of Atomic Problems. If for all $x \in X$, at least in one of the Atomic Problem, $\#_\ell(x)$ is greater than zero, then we accept, otherwise reject.

Structures with higher arity relations

The above algorithms can be extended to structures with relations of arity greater than two. First, we write the quantifier-free part ψ in DNF and split each term to a separate $\exists\exists\exists$ problem (or $\forall\exists\exists$ respectively). Then for each term ψ_t, we decide if there exist x_1, x_2, x_3 satisfying it. Let ψ_{t1} be the part of the conjunction containing all ternary predicates in ψ_t, and ψ_{t2} be the rest of term ψ_t. Thus $\psi_t = \psi_{t1} \wedge \psi_{t2}$.

If in ψ_t, some ternary predicate occurs positively, we can just count $\#(x_1)$ on the subgraph where ψ_{t1} is true.

If all ternary predicates in ψ_t occur negatively, then we first count $\#(x_1)$ satisfying formula ψ_{t2}, and then we count $\#'(x_1)$ on the subgraph where ψ_{t1} is true. Finally, we subtract $\#'(x_1)$ from $\#(x_1)$ for each x_1.

If ψ_t has no ternary relations, we just count $\#(x_1)$ using the algorithm for graphs.

3.10 Open Problems

An obvious open problem is whether a similar kind of equivalence exists for the dense case of OV. Is it "fine-grained equivalent" to some natural complexity class?

It would be interesting to find more reductions between and equivalences among the problems that are proven hard under some conjecture. For example, Edit Distance, Fréchet Distance, and Longest Common Subsequence are all almost quadratically hard assuming SETH. Are there any reductions between these problems? Are they all equivalent as far as having subquadratic algorithms? All of these problems have similar dynamic programming formulations. Can we formalize a class of problems with such dynamic programming algorithms and find complete problems for this class? More generally, we would like taxonomies of the problems within P that would classify more of the problems that have conjectured hardness, or have provable hardness based on conjectures about other problems. Such a taxonomy might have to be based on the structure of the conjectured best

algorithms for the problems rather than on resource limitations.

3.11 Acknowledgments

Chapter 3 contains material from "Completeness for First-Order Properties on Sparse Structures with Algorithmic Applications", by Jiawei Gao, Russell Impagliazzo, Antonina Kolokolova, and Ryan Williams, which appeared in the proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2017). The author of this dissertation was a principal author of this publication. We would thank Virginia Vassilevska Williams for her inspiring ideas. We would like to thank Marco Carmosino, Anant Dhayal, Ivan Mihajlin and Victor Vianu for proofreading and suggestions on this paper. We also thank Valentine Kabanets, Ramamohan Paturi, Ramyaa Ramyaa and Stefan Schneider for many useful discussions. Finally, we really appreciate the suggestions from the reviewers about the writing and expression.

Chapter 4

The Model Checking for Extensions of First-Order Logic

4.1 Chapter Overview

Here, we extend this class-based approach to fine-grained complexity. We consider the fine-grained complexities of various well-studied extensions of first-order logic. Surprisingly, we find that some extensions of first-order that *increase* the expressive power greatly *do not change* the fine-grained complexity of the corresponding class of model checking problems, while some extensions that *do not change* expressibility *substantially increase* the complexity of model checking. (This may not be as paradoxical as it would appear at first glance. Parity is not expressible in first-order, but a pre-processing stage to compute the parity would not greatly change the running time of most queries. In the other direction, although at a qualitative level, expressive power might be the same in an extension, the simulation might change quantitative aspects, such as the number of quantifiers. Our bounds give limits on the quantitative price that must be paid in doing the conversion from one logic to another.)

The basic logic that we consider is first-order relational logic. There is a finite list $R_1, ..., R_t$ of relation symbols, each with a non-negative integer arity a_i. A formula is built from these

symbols applied to variables, with Boolean connectives and the quantifiers $\forall$ and $\exists$. A finite model is specified as a universe U and for each R_i a set of tuples $(u_1, \ldots, u_{a_i}) \in U^{a_i}$. For algorithmic purposes, we can either represent a relation as a matrix or tensor of Booleans, which for every tuple of elements, specifies whether or not it is in the relation (analogous to an adjacency matrix of a graph), or as a list of possible tuples in the relation (analogous to a list of edges in a graph). In this chapter, we generally use the list representation, but sometimes need to refer to algorithms using the matrix representation. We use n to mean the size of U, and m to be $|U|$ plus the total number of tuples in all relations.

Our starting place is the characterization of the fine-grained complexity of sparse first-order properties from Chapter 3:

1. If the OV conjecture is true, then for every $\varepsilon > 0$, worst-case model checking of k-quantifier formulas requires time $\Omega(m^{k-1-\varepsilon})$. (This follows from [Wil14a]).

2. OV is complete for first-order: If the OV conjecture is false, then there is an $\varepsilon > 0$ so that for every $k \geq 3$, model checking of k quantifiers can be done in time $O(m^{k-1-\varepsilon})$.

3. Model checking of k quantifiers can be improved using the fastest OV algorithm: Unconditionally, model checking of k quantifiers can be done in time $m^{k-1}/2^{\Omega(\sqrt{\log m})}$.

So improvements in the entire class of model checking problems are captured by the extent to which OV algorithms can be improved.

We show, somewhat surprisingly, that different natural extensions of first-order logic sometimes maintain the complexity of model checking exactly, but sometimes change the complexity in an interesting way. Whether the complexity of the model checking problem is changed seems completely independent of whether the extension increases the expressive power of the logic.

In particular, we consider:

- **Other complexity measures of first-order logic:** instead of quantifier number k, we consider formulas of quantifier rank k and variable complexity k.

- **Extensions of first-order logic:** we can add function symbols, ordering, and transitive closure operations to first-order logic.

We get the following results:

First-order logic on ordered structures: The next three extensions all increase the expressive power of the logics in different ways, but do not change the complexity of the corresponding model checking problems.

In many applications, the elements of the universe are ordered, e.g., by time or location in memory. First-order properties on ordered structures have strictly more expressive power than those on unordered structures, even for properties not involving the ordering itself (by Gurevich, Theorem 5.3 in [Lib13]). However, we show that adding orderings does not change the complexity of model checking for k-quantifier first-order formulas:

1. If the OV conjecture is true, then for every $\varepsilon > 0$, worst-case model checking on ordered structures requires time $\Omega(m^{k-1-\varepsilon})$. (This follows from the earlier result for unordered structures).
2. OV is complete for first-order on ordered structures: If the OV conjecture is false, then there is an $\varepsilon > 0$ so that for every $k \geq 3$, model checking for structures with ordering can be done in time $O(m^{k-1-\varepsilon})$.
3. Model checking can be improved using the fastest OV algorithm: Unconditionally, model checking for formulas over ordered structures can be done in time $m^{k-1}/2^{\Omega(\sqrt{\log m})}$. Any future improvements to OV directly carry over to the model checking problems, regardless of how those improvements are obtained.

Transitive closures of symmetric relations: Adding transitive closure operations to first-order logic enhances its power significantly. However, here we find a special class involving transitive closure that is still equivalent to first-order: when transitive closure operations are only taken on symmetric input relations. This allows us to query whether two elements are in

the same connected component of an undirected graph. We show that the same conclusions above for first-order logic on ordered structures also hold for this class. Let φ be a first-order formula with a fixed number of transitive closure operations that are only taken on symmetric input relations, φ has $k \geq 3$ quantifiers, then

1. If the OV conjecture is true, then for every $\varepsilon > 0$, worst-case model checking for φ requires time $\Omega(m^{k-1-\varepsilon})$. (This follows from the earlier result for FO without transitive closure).
2. OV is complete for first-order: If the OV conjecture is false, then there is an $\varepsilon > 0$ so that for every $k \geq 3$, model checking for φ can be done in time $O(m^{k-1-\varepsilon})$.
3. Model checking can be improved using the fastest OV algorithm: Unconditionally, model checking for φ can be done in time $m^{k-1}/2^{\Omega(\sqrt{\log m})}$.

First-order formulas of quantifier rank k: We get a similar equivalence if we measure the complexity of first-order formulas by quantifier rank rather than number of quantifiers. Quantifier rank counts only the depth of nesting of quantifiers, rather than the total number of them. For example $\forall x[(\exists y R(x,y)) \wedge (\forall z S(x,z))]$ has three quantifiers, but since the two inside quantifiers are parallel rather than nested, has only quantifier rank two. However, we show that problems of any fixed quantifier rank, have the same complexity of model checking as for that number of total quantifiers. More precisely, let φ be a first-order formula of quantifier rank k, where $k \geq 3$, then

1. If the OV conjecture is true, then for every $\varepsilon > 0$, worst-case model checking for φ requires time $\Omega(m^{k-1-\varepsilon})$. (This follows from the earlier result for quantifier number.)
2. OV is complete for first-order: If the OV conjecture is false, then there is an $\varepsilon > 0$ so that for every $k \geq 3$, model checking for φ can be done in time $O(m^{k-1-\varepsilon})$.
3. Model checking can be improved using the fastest OV algorithm: Unconditionally, model checking for φ can be done in time $m^{k-1}/2^{\Omega(\sqrt{\log m})}$.

Transitive closure of arbitrary binary relations: For the next two extensions, we give some evidence that their model checking problems are not reducible to unextended first-order logic. While we do not give stronger conditional lower bounds in terms of running time, we show that these conditional bounds hold under a substantially weaker assumption than OVC, the analog of SETH for constant depth circuits.

Allowing general transitive closures of arbitrary binary relations increases the power of first-order logic dramatically. If a relation obtained from a transitive closure operation can be negated then the formula can express an even larger class of problems. For the model checking problem on graphs and for binary relations, however, the increase is less dramatic. We can compute each new transitive closure of an existing relation in $O(n^3)$ time. Then we can solve the model checking problem on the corresponding dense graph. This gives an $O(n^{k-1})$ algorithm for model checking k-quantifier problems for $k \geq 9$, and $O(n^k)$ for any $k \geq 3$. While we were unable to close this gap, or to show that transitive closures increase the complexity, we were able to prove the corresponding lower bound under a weaker assumption than SETH, SETH for polynomial-sized bounded depth formulas. Under this hypothesis, the class of 2-quantifier first-order formulas allowing transitive closure on arbitrary relations cannot be solved in time $O(m^{2-\varepsilon})$ for any $\varepsilon > 0$. More specifically,

1. The SETH of depth 2 circuit SAT implies the quadratic time hardness of 2-quantifier FO with positive TC only on original relations.
2. The SETH of depth 3 circuit SAT implies the quadratic time hardness of 2-quantifier FO with positive TC on subformulas containing TC only on original relations.
3. In general, the SETH of depth d circuit SAT implies the quadratic time hardness of 2-quantifier FO with $d-1$ nested layers of TC operations.

First-order formulas of variable complexity k: We get a similar result for another measure of the logical depth of formulas. When the variable names can be reused in different scopes, but there are no restrictions on the quantifier rank, then the situation is similar as when we allow

transitive closures on arbitrary binary relations: on formulas of variable complexity k, it is not known whether it can be computed faster than $O(n^{k-1+\omega})$ for $3 \leq k < 9$, or faster than $O(n^{k-1+o(1)})$ when $k \geq 9$, even if a large number of problems in this class can be solved in quadratic time (if they are "weakly succinct", see Appendix 4.10). Here we prove that under the SETH for polynomial-sized constant depth formulas, the class of first-order formulas of variable complexity 3 cannot be solved in time $O(m^{2-\varepsilon})$ for any $\varepsilon > 0$.

First-order logic with unary function symbols: In general, function symbols can be replaced with relations representing their graphs, so adding functions does not change the expressive power of first-order logic. However, in doing so, we might increase the number of quantifiers needed to express a property. We show that, assuming the low-dimension OV conjecture (which follows from SETH), the complexity of model checking problem increases by almost a linear factor when functions are added. More precisely, we show that if the low-dimension OV conjecture holds, then $\forall k \geq 2, \forall \varepsilon > 0$, there exists a k-quantifier first-order formula φ with unary functions, so that no algorithm can decide φ in time $O(m^{k-\varepsilon})$. In contrast, 2-quantifier first-order formulas without function symbols can be decided in linear time.

This approaches a factor of m over the upper bound for model checking such formulas without function symbols.

Table 4.1 shows the main results about different extensions of first-order logic. Figure 4.1 shows the reductions among problems. FO_k stands for FO formulas with k quantifiers. The solid arrow heads are reductions directly implied from the expressive power, and the hollow arrow heads are non-trivial reductions. The solid lines are reductions preserving subquadratic time (except for the reduction between exponential time problems, which does not preserve subquadratic time), while the dashed lines are reductions between problems with different conjectured running time. The thick lines are new results in this chapter, and the underlined problems get improved algorithms from these results.

Table 4.1: Best algorithms and conjectured hardness of different classes of logic.

Logic	Best Algorithm	Conjectured Hardness	Under Assumption
FOP_k, $k \geq 3$	$m^{k-1}/2^{\Omega(\sqrt{\log m})}$ (Chapter 3)	$\Omega(m^{k-1-\varepsilon})$	OVC (Chapter 3)
$\mathsf{FOP}_3(\leq)$, $k \geq 3$	$m^{k-1}/2^{\Omega(\sqrt{\log m})}$ (this chapter)	$\Omega(m^{k-1-\varepsilon})$	OVC (Chapter 3)
$\mathsf{FOP}_{\mathrm{qr}=k}$, $k \geq 3$	$m^{k-1}/2^{\Omega(\sqrt{\log m})}$ (this chapter)	$\Omega(m^{k-1-\varepsilon})$	OVC (Chapter 3)
$\mathsf{FOP}_k(\mathrm{TC}_{\mathrm{sym}})$, $k \geq 3$	$m^{k-1}/2^{\Omega(\sqrt{\log m})}$ (this chapter)	$\Omega(m^{k-1-\varepsilon})$	OVC (Chapter 3)
$\mathsf{FOP}_k(\mathrm{TC})$, $k \geq 2$	$\tilde{O}(n^{k-3+\omega})$, $n^{k-1+o(1)}$ for $k \geq 9$ [Wil14a]	$\Omega(m^{k-\varepsilon})$	constant depth circuit SETH (this chapter)
$\mathsf{FOP}_{\mathrm{vc}=k}$, $k \geq 3$	$\tilde{O}(n^{k-3+\omega})$, $n^{k-1+o(1)}$ for $k \geq 9$ [Wil14a]	$\Omega(m^{k-1-\varepsilon})$	constant depth circuit SETH (this chapter)
FOPF_k, $k \geq 2$	$\tilde{O}(m^k)$ (trivial)	$\Omega(m^{k-\varepsilon})$	SETH (this chapter)

4.2 Organization of this Chapter

Section 4.5 shows adding function symbols to first-order logic will make the model checking problem strictly harder by a reduction from OV. In Section 4.6 we show that introducing ordering to FO will not make the model checking harder. Section 4.7 shows when we allow transitive closures taken on symmetric input relations, the complexity stays the same. Section 4.3 proves that quantifier rank k formulas are no harder than formulas with k quantifiers in prenex normal form. In Section 4.4 we study two classes of problems that are hard under the SETH of constant depth circuits: formulas of variable complexity 3, and 2-quantifier formulas with transitive closure operations. In Section 4.8 we will talk about the open problems. Section 4.9 gives the baseline algorithm for first-order with ordering and first-order formulas of fixed quantifier rank. In section 4.10 we will discuss the running time for formulas of fixed variable complexity.

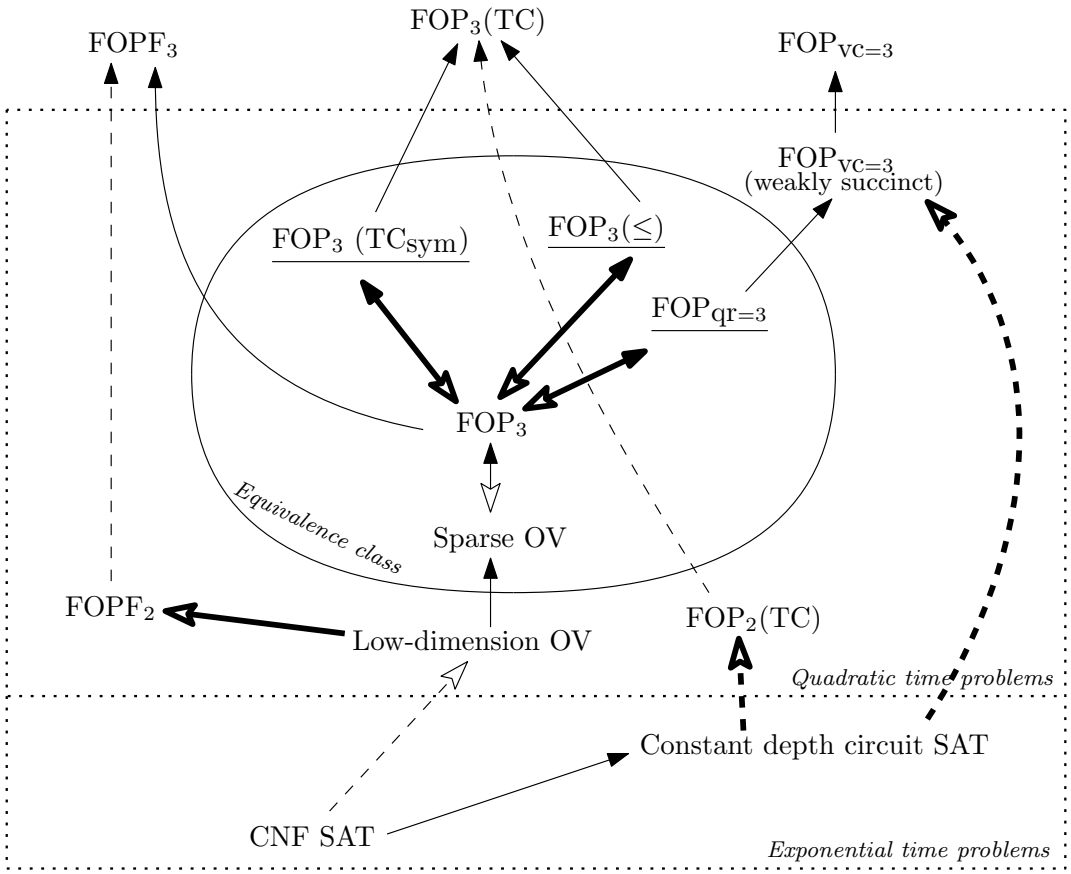


Figure 4.1: The expressive power and complexity of problems and classes of problems.

4.3 FO Formulas of Quantifier Rank k

A formula with quantifier rank k may have more than k variables when converted to prenex normal form, but the following theorem shows that even if it seems more powerful, it is reducible to quantifier number k problems.

Theorem 8. *For non-decreasing* $2^{\Omega(\sqrt{\log m})} \leq s(m) \leq m$*, and* $k \geq 3$*,*

$$(\mathsf{FOP}_{qr=k}, m^{k-1}/\mathsf{poly}(s(m))) \leq_{EC} (\mathsf{FOP}_3, m^2/s(m)).$$

To prove this theorem, we will show that $(\mathsf{FOP}_{\mathrm{qr}=3}, m^3) \leq_{\mathrm{FGR}} (\mathsf{FOP}_3, m^2)$. The reduction from $\mathsf{FOP}_{\mathrm{qr}=k}$ follows from the quantifier-eliminating downward self-reduction.

We will use a "Normal Problem" as an intermediate problem in the reduction. It is defined as follows:

List all x such that $\varphi_N(x)$ holds, where $\varphi_N(x) = \exists y[(\bigwedge_{1\le i\le L} \exists z_i \psi_i(x,y,z_i)) \wedge (\forall z' \psi'(x,y,z'))]$. Each z_i is a distinct variable from scope Z_i. Each ψ_i is a conjunction of predicates[1]. The predicates can appear either positively or negated.

In Lemma 4.3.1 we show a reduction from the Normal Problem to FOP_3, and in Lemma 4.3.2 we show a reduction from an $\mathsf{FOP}_{\mathrm{qr}=3}$ problem to the Normal Problem.

Lemma 4.3.1. $(\textit{Normal Problem}, m^2/\mathsf{poly}(s(m))) \le_{EC} (\mathsf{FOP}_3, m^2/s(m))$.

Proof. Let d be a threshold value of the degree of elements.

Step 1. Decide for x of degree at least d on some Z_i.

For those x of degree greater than d on some Z_i, there can be at most m/d of them. We enumerate all such x and then solve corresponding 2-quantifier problems in linear time by the baseline algorithm. The total time is $O(m^2/d)$.

Step 2. Decide for y of degree at least d on some Z_i.

For those y of degree greater than d on some z_i, there can be at most m/d of them. We enumerate all such y's and then list all the x in the corresponding 2-quantifier problems in linear time by the baseline algorithm. Finally we can merge all the lists of x. The total time is $O(m^2/d)$.

Step 3. Decide for x and y of degree less than d.

For each ψ_i, we consider the two cases based on whether it contains any positive occurrences of binary predicates on z_i.

Step 3-1. Consider the ψ_i's where all binary predicates involving z_i are negative.

In this case, we will show that if $|Z_i|$ is large enough, then the conjunction of all negative predicates on z_i is always true, otherwise the problem is easy to be transformed to the case in Step 3-2.

If $|Z_i|$ is greater than $2d$, then for any pair of x and y, at least one $z_i \in Z_i$ is not adjacent to x or y by any relation, because x and y together have no more than $2d$ neighbors. Thus all the

[1]Here we assume all predicates are either unary or binary. A ternary predicate on x, y, z_i can be treated in a similar way as a binary predicate on either x, z_i or y, z_i.

negative predicates on z_i are true on the triple (x, y, z_i). So we can replace the conjunction of all these negative predicates by *true* in ψ_i, leaving only binary predicates on x, y and unary predicates.

If $|Z_i|$ is less than $2d$, then in time $O(m \cdot 2d)$ we can create a new relation $R_c(x, z_i)$. $R_c(x, z_i)$ is true iff all binary relations on (x, z_i) evaluate to false. We remove all binary relations on (x, z_i) from ψ_i, and append "$\wedge R_c(x, z_i)$" to the end of it. Because $R_c(x, z_i)$ appears positively, we will decide it in the next step. After creating the new relation, the size of the structure has become $m' = m \cdot 2d$.

Step 3-2. Now in all of the remaining ψ_i's, either some binary predicate on (x, z_i) is positive or some on (y, z_i) is positive.

Without loss of generality, assume that in the formulas ψ_i, for $1 \leq i \leq u$ the pairs (x, z_i) are in some positive predicates, and for $i > u$, the pairs (x, z_i) do not appear in any positive binary predicate. Similarly, assume for $j > t$ the pairs (y, z_j) are in some positive predicates, and for $j \leq t$, the pairs (y, z_j) do not appear in any positive binary predicate. It must be $t \leq u$, because for any z_i there is at least one positive predicate on z_i.

The idea of this reduction is to let the big variable $\widetilde{x}$ contain variables z_i for $i \leq u$, and let the big variable $\widetilde{y}$ contain variables z_j for $j > u$. For each pair of $\widetilde{x}$ and $\widetilde{y}$, there is a unique tuple of $(z_1, \ldots, z_L)$. Therefore, each $\exists z_i$ can be replaced by $\forall z_i$.

On each x, for all u tuples of neighbors $z_1 \in Z_1, \ldots, z_u \in Z_u$, we create a new element $\widetilde{x} = (x, z_1, \ldots, z_u)$ in the domain $\widetilde{X}$. Because x has at most d neighbors, the number of distinct $\widetilde{x}$ is bounded by d^u. Next, we create the following new relations on $\widetilde{x}$. Define an auxiliary relation $R_\ni$ where $R_\ni(\widetilde{x}, x)$ is true iff the tuple represented by $\widetilde{x}$ contains x, and $R_\ni(\widetilde{x}, z_i)$ is true iff $\widetilde{x}$ contains z_i.

Next we replace all relations on x by relations on $\widetilde{x}$.

- For each unary relation $R_k(x)$, we replace it by new unary relation R_k^* where $R_k^*(\widetilde{x})$ is true iff $R_k(x) \wedge R_\ni(\widetilde{x}, x)$.
- For each unary relation $R_k(z_i)$ where $i \leq u$, we replace it by new unary relation R_k^* where $R_k^*(\widetilde{x})$ is true iff $R_k(z_i) \wedge R_\ni(\widetilde{x}, z_i)$.

- For each binary relation $R_k(x, z_i)$ where $i \leq u$, we replace it by new unary relation R_k^* where $R_k^*(\widetilde{x})$ is true iff $R_k(x, z_i) \wedge R_\ni(\widetilde{x}, x) \wedge R_\ni(\widetilde{x}, z_i)$.

- For each binary relation $R_k(x, z_j)$ where $j > u$, we replace it by new binary relation R_k^* where $R_k^*(\widetilde{x}, z_j)$ is true iff $R_k(x, z_j) \wedge R_\ni(\widetilde{x}, x)$.

- For each binary relation $R_k(x, z')$, we replace it by new binary relation R_k^* where $R_k^*(\widetilde{x}, z')$ is true iff $R_k(x, z') \wedge R_\ni(\widetilde{x}, x)$.

There are at most $m \cdot d^u$ distinct elements of $\widetilde{x}$, so the unary relations on it is also bounded by this value. From each edge on x we have created at most d^u new edges. Since there are m' edges, the total size of new binary relations is $O(m' \cdot d^L)$.

Similarly, on each y, for all $L - u$ tuples of neighbors $z_1 \in Z_{u+1}, \ldots, z_L \in Z_L$, we create a new element $\widetilde{y} = (y, z_{u+1}, \ldots, z_L)$ in the domain $\widetilde{Y}$. Again we replace old relations by new relations on $\widetilde{y}$ in a similar way as those on $\widetilde{x}$.

- For each unary relation $R_k(y)$, we replace it by new unary relation R_k^* where $R_k^*(\widetilde{y})$ is true iff $R_k(y) \wedge R_\ni(\widetilde{y}, y)$.

- For each unary relation $R_k(z_j)$ where $j > u$, we replace it by new unary relation R_k^* where $R_k^*(\widetilde{y})$ is true iff $R_k(z_j) \wedge R_\ni(\widetilde{y}, z_j)$.

- For each binary relation $R_k(y, z_j)$ where $j > u$, we replace it by new unary relation R_k^* where $R_k^*(\widetilde{y})$ is true iff $R_k(y, z_j) \wedge R_\ni(\widetilde{y}, y) \wedge R_\ni(\widetilde{y}, z_j)$.

- For each binary relation $R_k(y, z_i)$ where $i \leq u$, we replace it by new binary relation R_k^* where $R_k^*(\widetilde{y}, z_i)$ is true iff $R_k(y, z_i) \wedge R_\ni(\widetilde{y}, y)$.

- For each binary relation $R_k(y, z')$, we replace it by new binary relation R_k^* where $R_k^*(\widetilde{y}, z')$ is true iff $R_k(y, z') \wedge R_\ni(\widetilde{y}, y)$.

Similarly, the total size of the new relations is also $O(m' \cdot d^L)$.

Next, we deal with relations between $\widetilde{x}$ and $\widetilde{y}$.

- For each binary relation $R_k(x,y)$, we replace it by new binary relation R_k^* where $R_k^*(\widetilde{x},\widetilde{y})$ is true iff $R_k(x,y) \wedge R_\ni(\widetilde{x},x) \wedge R_\ni(\widetilde{y},y)$.

Because the number of $\widetilde{x}$ corresponding to an x is d^u and the number of $\widetilde{y}$ corresponding to an y is d^{L-u}, and because there are m' relations on x,y, the total size of the new relations is $O(m' \cdot d^L)$.

After modifying the input structure, next we will modify the formula $\varphi_N(x)$. We change the predicates in each formula ψ_i to get ψ_i^*, and change the predicates in ψ' to get ψ'^*.

- For each occurrence of predicate $R_k(x)$, we replace it by $R_k^*(\widetilde{x})$. For each occurrence of predicate $R_k(x)$, we replace it by $R_k^*(\widetilde{y})$.
- For each occurrence of predicate $R_k(z_i)$ where $i \leq u$, we replace it by $R_k^*(\widetilde{x})$. For each occurrence of predicate $R_k(z_j)$ where $j > t$, we replace it by $R_k^*(\widetilde{y})$.
- For each occurrence of predicate $R_k(x,z_i)$ where $i \leq u$, we replace it by $R_k^*(\widetilde{x})$.
- For each occurrence of predicate $R_k(x,z_j)$ where $j > u$, we replace it by $R_k^*(\widetilde{x},z_j)$.
- For each occurrence of predicate $R_k(y,z_j)$ where $j > u$, we replace it by $R_k^*(\widetilde{y})$.
- For each occurrence of predicate $R_k(y,z_i)$ where $i \leq u$, we replace it by $R_k^*(\widetilde{y},z_i)$.
- For each occurrence of predicate $R_k(x,y)$, we replace it by $R_k^*(\widetilde{x},\widetilde{y})$.
- For each occurrence of predicate $R_k(x,z')$, we replace it by $R_k^*(\widetilde{x},z')$. For each occurrence of predicate $R_k(y,z')$, we replace it by $R_k^*(\widetilde{y},z')$.

We merge all z_i elements and z' elements into the same universe Z, and create unary predicates $IsZ_1(z),\ldots,IsZ_L(z)$ and $IsZ'(z)$ to represent whether a z is originally from some certain Z_i or in Z'.

Now the goal of the new problem is to list all $\widetilde{x}$ such that $\exists\widetilde{y}\forall z$, all of the following holds.

- For i in range 1 to t, $(IsZ_i(z) \wedge R_\ni(\widetilde{x},z)) \rightarrow \psi_i^*(\widetilde{x},\widetilde{y},z)$

- For i in range $u+1$ to L, $(IsZ_i(z) \wedge R_{\ni}(\widetilde{y},z)) \rightarrow \psi_i^*(\widetilde{x},\widetilde{y},z)$
- $IsZ'(z) \rightarrow \psi'^*(\widetilde{x},\widetilde{y},z)$

So it is a "List-$\exists$-$\forall$" type problem of size $O(m \cdot 2d \cdot d^L) = O(md^{L+1})$. By the grouping-reduction technique, it is reducible to FOP_3. Assume "List-$\exists$-$\forall$" problems are in time $m^2/s(m)$. Then the Normal Problem is in time $O(m^2/d + (md^{L+1})^2/s(md^{L+1})) \leq O(m^2/d + (md^{L+1})^2/s(m))$. By choosing $d = s(m)^{\frac{1}{2L+3}}$, we get running time $O(m^2/s(m)^{\frac{1}{2L+3}})$.

□

Lemma 4.3.2. *There is a linear time Turing reduction from any $\mathsf{FOP}_{qr=3}$ problem to the Normal Problem.*

Proof. Let the variables defined in the outermost layer be named x, let the variables defined in the middle layer be named y and let the variables defined in the innermost layer be named z.

The structure outside the outermost quantifiers

For a quantifier rank 3 formula φ that is composed of form $(Q_i x \varphi_i(x))$ connected by ANDs and ORs, we decide each $Q_i x \varphi_i(x)$ separately. For each $Q_i x \varphi_i(x)$, we will show that we can compute a list of x such that $\varphi_i(x)$ is true, so that we can do union and intersection operations on all the lists and decide the value of φ.

The outermost layer of quantifiers

To decide the truth value of $\varphi_i(x)$ for every x, we write $\varphi_i(x)$ in DNF, treating any quantified subformulas as atoms, so that $\varphi_i(x)$ has form $\bigvee_{j\in[J]} \bigwedge_{k\in[K]} (Q_{ijk} y \varphi'_{ijk}(x,y))$ for some constants J and K. For each x, we will decide for all j,k the truth value of $Q_{ijk} y \varphi'_{ijk}(x,y)$. Thus finally we can decide whether for each x there exists some j such that for all k, $Q_{ijk} y \varphi'_{ijk}(x,y)$ holds. If Q_{ijk} is $\forall$, we decide its negation (that is $\exists y \neg \varphi'_{ijk}(x,y)$), and after we finally get a list of x, we complement the list. Thus we can only consider the case $Q_{ijk} = \exists$.

The second layer of quantifiers

We fix the x,i,j,k in the previous step, and consider subformula of form $\exists y \varphi'(x,y)$. We write it in DNF so that it has form

$$\exists y \bigvee_{g\in[G]} \bigwedge_{h\in[H]} (Q_{gh} z \psi'_{gh}(x,y,z)) = \bigvee_{g\in[G]} \exists y \bigwedge_{h\in[H]} (Q_{gh} z \psi_{gh}(x,y,z)).$$

Then $\bigwedge_{h\in[H]}(Q_{gh}z\psi'_{gh}(x,y,z))$ can be written in form $(\bigwedge_\ell \exists z \psi_\ell(x,y,z)) \wedge \forall z \psi'(x,y,z)$ by merging all the $\forall z$ subformulas into one big $\forall z$ connected by $\wedge$. Next, write each ψ_ℓ in DNF, and move the $\vee$'s outside the the"$\exists z$"s, and distribute with the "$\bigwedge_\ell$", so that $(\bigwedge_\ell \exists z \psi_\ell(x,y,z))$ is equivalent to a disjunction of $(\bigwedge_{\ell'} \exists z \psi_{\ell'}(z))$ where each $\psi_{\ell'}$ is a conjunction of predicates and negated predicates. Because "$\vee$" commutes with "$\exists$", the big disjunction before "$\exists z$" can be moved outside "$\exists y$". So in the end, we only need to solve a constant number of instances of form "List all x such that $\exists y[(\bigwedge_{\ell'} \exists z \psi_{\ell'}(z)) \wedge (\forall z \psi'(x,y,z))]$. □

By the algorithm for OV in [AWY15, CW16] and the algorithm for FOP_k in Chapter 3, we get an improved algorithm for $\mathsf{FOP}_{\mathrm{qr}=k}$ in time $m^k/2^{\Theta(\sqrt{\log m})}$.

4.4 Conditional Hardness under the SETH of Constant Depth Circuits

The satisfiability of higher depth circuits may be harder than the satisfiability of CNF. Thus, the Strong Exponential Time Hypothesis of a circuit of depth greater than 2 is weaker than the SETH of CNF-SAT. It may be possible that even if the SETH of CNF-SAT is refuted, the SETH of higher depth circuits still holds true. This section shows that in this case, variable complexity 3 formulas and 2-quantifier formulas with transitive closures would require quadratic time.

4.4.1 Hardness of Variable Complexity 3 Formulas

In this section we prove the following theorem, which shows that the SETH of constant depth circuit implies the quadratic-time hardness of $\mathsf{FOP}_{\mathrm{vc}=3}$.

Theorem 9. *If $\mathsf{FOP}_{vc=3}$ is solvable in time $O(m^{2-\varepsilon})$ for some $\varepsilon > 0$, then the satisfiability of constant depth circuits of size M with N variables is solvable in time $2^{(1-\varepsilon/2)N} \cdot \mathsf{poly}(M)$.*

In Circuit SAT, without loss of generality we assume that the circuit is in De Morgan form: it has d levels, where the gates of level $(i+1)$ only have input wires from gates of level i. The NOT gates only appear in the bottom level, which we call level 0. Let the level of AND and OR gates nearest to the input wires be level 1, and the output gate be level d.

Now we reduce the satisfiability of this circuit to a property defined by an FO formula of variable complexity 3. For the N input variables, we split them into two sets of size $N/2$ each. Let α represent partial assignments of the first $N/2$ variables, and let β represent those of the rest $N/2$ variables. So there are $2^{N/2}$ distinct α and β. For each gate g and each partial assignment α, create a variable g_α representing the tuple (g,α). Define predicate $Same(g_\alpha, g)$ to be true iff the gate in g_a is the same gate as g. Define unary predicate $IsAND(g_\alpha)$ to true iff g is an AND gate, and similarly define $IsOR(g_\alpha)$ for whether g is an OR gate. For any tuple (g'_α, g_α) sharing the same α where gate g' is input to g, we let relation $Input(g'_\alpha, g_\alpha)$ be true on this tuple.

For each level 1 gate g and each α, consider the two cases. If g is an AND gate, we evaluate whether the partial assignment α does not falsify g. If so, then let the relation $Sat_1(g_\alpha)$ be true, otherwise false. If g is an OR gate, we evaluate whether the partial assignment already makes g true. If so, let the relation $Sat_1(g_\alpha)$ be true, otherwise false.

Similarly, for each level 1 gate g and each β, if g is an AND gate and β does not falsify g, then let the relation $Sat_2(g,\beta)$ be true, otherwise false. If g is an OR gate and β makes g true, then we let the relation $Sat_2(g,\beta)$ be true, otherwise false.

Next we will compute d intermediate relations from $TrueGates_1$ to $TrueGates_d$. The relation

$TrueGates_i(g_\alpha, \beta)$ holds iff g is a level-i gate, and the assignment by α and β satisfies g.

$$\begin{aligned}
TrueANDGates_1 = \{(g_\alpha, \beta) \mid & (g_\alpha \in Level_1) \wedge IsAND(g_\alpha) \\
& \wedge Sat_1(g_\alpha) \wedge \exists g(Same(g_\alpha, g) \wedge Sat_2(g, \beta))\} \\
TrueORGates_1 = \{(g_\alpha, \beta) \mid & (g_\alpha \in Level_1) \wedge IsOR(g_\alpha) \\
& \wedge (Sat_1(g_\alpha) \vee \exists g(Same(g_\alpha, g) \wedge Sat_2(g, \beta)))\} \\
TrueGates_1 = \{(g_\alpha, \beta) \mid & TrueANDGates_1(g_\alpha, \beta) \vee TrueORGates_1(g_\alpha, \beta)\}
\end{aligned}$$

For $i = 2$ to $i = d$, we define the following intermediate relations:

$$\begin{aligned}
TrueANDGates_i = \{(g_\alpha, \beta) \mid & (g_\alpha \in Level_i) \\
& \wedge IsAND(g_\alpha) \\
& \wedge \forall g'_\alpha \in Level_{i-1}(Input(g'_\alpha, g_\alpha) \rightarrow TrueGates_{i-1}(g'_\alpha, \beta))\} \\
TrueORGates_i = \{(g_\alpha, \beta) \mid & (g_\alpha \in Level_i) \\
& \wedge IsOR(g_\alpha) \\
& \wedge \exists g'_\alpha \in Level_{i-1}(Input(g'_\alpha, g_\alpha) \wedge TrueGates_{i-1}(g'_\alpha, \beta))\} \\
TrueGates_i = \{(g_\alpha, \beta) \mid & TrueANDGates_i(g_\alpha, \beta) \vee TrueORGates_i(g_\alpha, \beta)\}
\end{aligned}$$

Finally, the circuit is satisfiable iff $\exists g_\alpha \; \exists \beta \; TrueGates_d(g_\alpha, \beta)$.

Here, each intermediate relation is defined by 3 variables, therefore, the total variable complexity is 3. Also, in the definition formula of each intermediate relation, there is at most one occurrence of any previously computed intermediate binary relations. By Appendix 4.10, the formula is weakly succinct, therefore it can be solved in quadratic time.

The total number of elements is $2^{N/2}\mathsf{poly}(M)$. The total number of original relations is also $2^{N/2}\mathsf{poly}(M)$. So if $\mathsf{FOP}_{\mathrm{vc}=3}$ is time $O(m^{2-\varepsilon})$, then the Circuit SAT instance is in $2^{(1-\varepsilon/2)N} \cdot \mathsf{poly}(M)$ time, contradicting the SETH of constant depth circuits.

4.4.2 Hardness of 2 Variable Formulas with Transitive Closure

This section proves the conditional hardness for the case where the formula has only two quantifiers, where transitive closure operations can appear arbitrarily.

Theorem 10. *Let SAT_d be the satisfiability problem of depth d circuits with N variables of size M.*

1. *If the model checking for 2-quantifier FO formula with positive TC only on original relations is in time $O(m^{2-\varepsilon})$ for some $\varepsilon > 0$, then SAT_2 can be solved in time $2^{(1-\varepsilon/2)N} \cdot \mathsf{poly}(M)$.*

2. *If the model checking for 2-quantifier FO formula with positive TC on subformulas containing TC on original relations can be solved in time $O(m^{2-\varepsilon})$ for some $\varepsilon > 0$, then SAT_3 can be solved in time $2^{(1-\varepsilon/2)N} \cdot \mathsf{poly}(M)$.*

3. *In general, if the model checking for 2-quantifier FO formula with d nested layers of TC operations can be solved in time $O(m^{2-\varepsilon})$ for some $\varepsilon > 0$, then SAT_{d-2} can be solved in time $2^{(1-\varepsilon/2)N} \cdot \mathsf{poly}(M)$.*

The reduction is similar to the one for $\mathsf{FOP}_{\mathrm{vc}=3}$. We again use partial assignments α, β, and let variable g_α to represent tuple (g, α). Relations *Same*, *Input* are also defined in the same way. This time we assume that in the circuit, on each level either all gates are AND or all gates are OR.

The bottom 2 levels of gates

• If the bottom level are AND gates, and the next level are OR gates:

First, define a relation *Sat* as follows:

$$\begin{aligned} Sat = &\{(g,\beta) \mid g \in Level_1 \wedge (\beta \text{ does not make } g \text{ false})\} \\ &\cup \{(g_\alpha, g) \mid g_\alpha \in Level_1 \wedge Same(g_\alpha, g) \wedge (\alpha \text{ does not make } g \text{ false})\} \\ &\cup \{(g'_\alpha, g_\alpha) \mid g'_\alpha \in Level_2 \wedge g_\alpha \in Level_1 \wedge Input(g_\alpha, g'_\alpha)\} \end{aligned}$$

The above relation can be created in time $2^{n/2}\mathsf{poly}(m)$, where m is the size of the circuit. This relation is like a union of $Sat_1, Sat_2, TrueGates_1$ and *Input* relations in the previous section.

Next, we define an intermediate relation for level 2 gates:

$$\begin{aligned} \mathit{TrueGates}_2 = &\{(g_\alpha, \beta) \mid g_\alpha \in \mathit{Level}_2 \wedge \mathit{TC}_{\mathit{Sat}}(g_\alpha, \beta)\} \\ &\cup \{(g'_\alpha, g_\alpha) \mid g'_\alpha \in \mathit{Level}_3 \wedge g_\alpha \in \mathit{Level}_2 \wedge \mathit{Input}(g_\alpha, g'_\alpha)\} \end{aligned}$$

We claim that level 2 gate g' is satisfied by α and β iff $\mathit{TrueGates}_2$ is true on tuple g'_α, β. This is because $\mathit{TrueGates}_2$ is true on g'_α, β where g' is a level 2 gate iff there is a path $g'_\alpha \to g_\alpha \to g \to \beta$ by *Sat*, where g is a level 1 gate. This means neither α and β make the AND gate g false, so g is satisfied by α and β. Also g is input to the OR gate g', so g' is also satisfied. In the other direction, if g' is satisfied by α and β then there must be such a path.

• If the bottom level are OR gates, and the next level are AND gates:

This case is analogous to the previous case because AND is the negation of OR. First, define a relation *Falsify* as follows:

$$\begin{aligned} \mathit{Falsify} = &\{(g, \beta) \mid g \in \mathit{Level}_1 \wedge (\beta \text{ does not make } g \text{ true})\} \\ &\cup \{(g_\alpha, g) \mid g_\alpha \in \mathit{Level}_1 \wedge \mathit{Same}(g_\alpha, g) \wedge (\alpha \text{ does not make } g \text{ true})\} \\ &\cup \{(g'_\alpha, g_\alpha) \mid g'_\alpha \in \mathit{Level}_2 \wedge g_\alpha \in \mathit{Level}_1 \wedge \mathit{Input}(g_\alpha, g'_\alpha)\} \end{aligned}$$

Then we define an intermediate relation for level 2 gates:

$$\begin{aligned} \mathit{TrueGates}_2 = &\{(g_\alpha, \beta) \mid g_\alpha \in \mathit{Level}_2 \wedge \neg \mathit{TC}_{\mathit{Falsify}}(g_\alpha, \beta)\} \\ &\cup \{(g'_\alpha, g_\alpha) \mid g'_\alpha \in \mathit{Level}_3 \wedge g_\alpha \in \mathit{Level}_2 \wedge \mathit{Input}(g_\alpha, g'_\alpha)\} \end{aligned}$$

We claim that level 2 gate g' is satisfied by α and β iff $\mathit{TrueGates}_2$ is true on tuple g'_α, β. This is because $\mathit{TrueGates}_2$ is false on g'_α, β where g' is a level 2 gate iff there exists a path $g'_\alpha \to g_\alpha \to g \to \beta$ by *Falsify*, where g is a level 1 gate. This means neither α and β make the OR gate g true, so g is falsified by α and β. Also g is input to the AND gate g', so g' is also falsified.

Higher levels of gates

From level 3 up, if the current level i are OR gates and the level $i-1$ are AND gates, then define intermediate relations

$$\begin{aligned} \mathit{TrueGates}_i = &\{(g_\alpha, \beta) \mid g_\alpha \in \mathit{Level}_i \wedge \mathit{TC}_{\mathit{TrueGates}_{i-1}}(g_\alpha, \beta)\} \\ &\cup \{(g'_\alpha, g_\alpha) \mid g'_\alpha \in \mathit{Level}_{i+1} \wedge g_\alpha \in \mathit{Level}_i \wedge \mathit{Input}(g_\alpha, g'_\alpha)\} \end{aligned}$$

Otherwise, if the current level i are AND gates and the level $i-1$ are OR gates, then define

$$\begin{aligned} \mathit{TrueGates}_i = &\{(g_\alpha, \beta) \mid g_\alpha \in \mathit{Level}_i \wedge \neg \mathit{TC}_{\mathit{FalseGates}_i}(g_\alpha, \beta)\} \\ \mathit{FalseGates}_i = &\{(g_\alpha, \beta) \mid g_\alpha \in \mathit{Level}_i \wedge \neg \mathit{TrueGates}_i(g_\alpha, \beta)\} \\ &\cup \{(g'_\alpha, g_\alpha) \mid g'_\alpha \in \mathit{Level}_{i+1} \wedge g_\alpha \in \mathit{Level}_i \wedge \mathit{Input}(g_\alpha, g'_\alpha)\} \end{aligned}$$

For OR gates g'_α where g' is on level i, $\mathit{TrueGates}_i(g'_\alpha, \beta)$ is true iff there exists some g_α where g is on level $i-1$ satisfying $\mathit{TrueGates}_{i-1}(g_\alpha, \beta)$ and $\mathit{TrueGates}_{i-1}(g'_\alpha, g_\alpha)$. This means g is satisfied by α, β and g is input to g'.

For AND gates g'_α where g' is on level i, $\mathit{TrueGates}_i(g'_\alpha, \beta)$ is false iff there exists some g_α where g is on level $i-1$ satisfying $\mathit{FalseGates}_{i-1}(g_\alpha, \beta)$ and $\mathit{FalseGates}_{i-1}(g'_\alpha, g_\alpha)$. This means g is falsified by α, β and g is input to g'.

Finally, the circuit is satisfiable iff $\exists g_\alpha \in \mathit{Level}_d \; \exists \beta \; \mathit{TrueGates}_d(g_\alpha, \beta)$.

Like the previous section, the total number of elements is $2^{N/2}\mathsf{poly}(M)$, and the total number of original relations is also $2^{N/2}\mathsf{poly}(M)$.

4.5 FO with Unary Function Symbols

Consider adding function symbols to first-order logic. Unary functions can be represented as arrays of linear size, but binary functions increase the input size to quadratic, and so on for higher arities. So to measure complexity in terms of the input size, we only consider unary functions. To

simulate higher arities, we could increase the universe to a Cartesian power and then have unary functions on this product space.

While we can simulate any function by a relation coding the graph of the function, $R_f(x,y) \iff f(x) = y$, to express, for example that $f(x_1) = f(x_2)$ we would need to write $\exists y, R_f(x_1,y) \wedge R_f(x_2,y)$. So the number of quantifiers in the translated forumlas would increase, possibly up to the number of function symbols appearing in the original. However, there is still a trivial $O(n^k)$ algorithm for model checking a k-quantifier formula with functions. So, assuming the OV Conjecture, the complexity grows by at most a linear amount over that for first-order without functions.

In this section, we show that this increase is necessary. Compared to the linear time baseline algorithm for the model checking of 2 quantifier formulas, when we introduce function symbols, a 2 quantifier formula may require quadratic time to solve.

Theorem 11. *The low-dimension OV conjecture implies that for any* $\varepsilon > 0$, FOPF_2 *cannot be decided in time* $O(m^{2-\varepsilon})$.

The low-dimension k-OV conjecture implies that for any $\varepsilon > 0$, FOPF_k *cannot be decided in time* $O(m^{k-\varepsilon})$.

Consider an OV instance where the dimension of vectors $d = c \log n$ for some constant c. We construct a hypergraph for model checking as follows.

Let each vector correspond to an element. Besides these elements, we create $O(\sqrt{n})$ extra elements, each corresponding to a distinct boolean vector of dimension $\frac{1}{2}\log n$. For all pairs of these short vectors that are orthogonal to each other, we create an edge of relation $R_\perp$ between them. There are at most $O((\sqrt{n})^2) = O(n)$ pairs of short vectors, so the relation $R_\perp$ is sparse.

Each vector of length $c \log n$ can be partitioned into $2c$ blocks of length $\frac{1}{2}\log n$. We define functions $f_1, \ldots, f_{2c}$ so that $f_i : \{0,1\}^{c \log n} \to \{0,1\}^{\frac{1}{2}\log n}$ be the mapping from a vector to its i-th block. Therefore, there exists a pair of orthogonal vectors x, y iff in FO with function symbols,

$$\varphi = \exists x \exists y \bigwedge_{i=1}^{2c} R_\perp(f_i(x), f_i(y))$$

is satisfied.

If we can decide such φ in time $O(m^{2-\varepsilon})$, then for any constant $c > 0$, OV of dimension $d = c \log n$ can be solved in time $O(n^{2-\varepsilon})$, contradicting the Low-Dimension OV Conjecture. Furthermore, the SETH of CNF-SAT will also be refuted.

The reduction from k-OV is similar, where we create a set of $O(\sqrt[k]{n})$ extra elements representing all boolean vectors of length $(1/k)\log n$, and create kc functions mapping vectors to its blocks of length $(1/k)\log n$. Here, the relation $R_\perp$ is k-ary, defined on the k-tuples of short vectors whose inner product is zero. The total number of these k-ary relations is still $(\sqrt[k]{n})^k = O(n)$.

4.6 FO with Comparison on Ordered Structures

In this section we will consider the case where elements are given a total pre-ordering, and there are three predicates expressing that an element is greater than, less than, or equivalent to another element in the ordering. The comparison relation is an implicit dense relation but can be represented in $O(n)$ space in the input, by giving a table indexed by element, giving the element's rank within the ordering, with equivalent elements given the same rank. (If we were not given this table, we could use any sorting algorithm to construct it in $O(n \log n)$ time.) Using this table, we can list the elements by this ordering in time $O(n)$, and given any two elements, we can compare them in time $O(1)$. The following theorem shows that adding comparison to first-order logic does not increase the fine-grained complexity because it is equivalent to first-order properties without comparison.

Theorem 12. *For non-decreasing* $2^{\Omega(\sqrt{\log m})} \leq s(m) \leq m$,

$$(\mathsf{FOP}_k(\leq), m^{k-1}/s(\mathsf{poly}(m))) \leq_{EC} (\mathsf{FOP}_3, m^2/s(m)).$$

We will show that $(\mathsf{FOP}_3(\leq), m^2/s(\mathsf{poly}(m))) \leq_{\mathrm{EC}} (\mathsf{FOP}_3, m^2/s(m))$. The reduction from $\mathsf{FOP}_k(\leq)$ follows from the quantifier-eliminating downward self-reduction.

Assume in time $T_{FO}(m)$ we can list all x satisfying any formula of form $\exists y Q_3 z \psi_{FO}(x,y,z)$, where ψ_{FO} is a quantifier-free first-order formula without ordering. By the grouping-reduction technique, it is reducible to FOP_3.

Let φ be in prenex normal form, and assume $\varphi = Q_1 x \exists y Q_3 z \psi(x,y,z)$, for otherwise if y is quantified by $\forall$, we will decide the negation of the formula. We show that we can list all x such that $\exists y Q_3 z \psi(x,y,z)$ holds.

Let g be a threshold value of elements' degree. First, we decide whether to include x on our list for all x of degree greater than g. There are at most m/g of them. For each large degree x, treating it as a constant we get a 2-quantifier problem. By the baseline algorithm (Lemma 4.9.1), it can be decided in time $O(m)$. So the total time spent is $O(m^2/g)$.

Next, for all y of degree greater than g, we list the set of x that cause the one-quantifier sub-formula to be true. For each large degree y, treating it as a constant we compute the problem: list all x such that $Q_3 z \psi(x,y,z)$. Using the baseline algorithm we can list all x satisfying $Q_3 z \psi(x,y,z)$ in time $O(m)$. Finally we merge all lists of x computed on each y. For any x in the lists, there must exist a y where $Q_3 z \psi(x,y,z)$ holds.

By triplicating elements, we can assume that the x, y, and z variables are quantified over disjoint domains X, Y, and Z. The ordering relation is defined on the union of all three sets. We remove from X and Y the elements of degree higher than g. We partition the whole universe $X \cup Y \cup Z$ into intervals where for each interval the total degree of elements in $X \cup Y \cup Z$ is between g and $2g$, the interval is a single element of Z of degree higher than g. Thus, there are $O(m/g)$ intervals.

Note that we allow elements to be equivalent in the pre-ordering. When we group the elements into intervals, we will keep all equivalent elements in the same interval, unless there are so many such elements that their total degree is more than g. If that is the case, we break the set of equivalent elements arbitrarily into groups of total degree between g and $2g$, and do not include any non-equivalent elements in these groups.

Let the elements of X, Y, Z in the i-th interval form sets X_i, Y_i, Z_i, respectively. For each pair

of groups X_i, Y_j, we need to create a list $L_{i,j}$ of those x in X_i for which there exists a y in Y_j such that the rest of the formula holds true. Our final output will be the union of all these lists.

We call a pair i, j *special* if either $i = j$ or there is some tuple in one of the non-ordering input relations involving elements from both intervals. There are at most $O(m)$ special pairs, since each tuple involves at most a constant number of intervals. We handle special and non-special pairs differently.

Note that each element z not in either of the two intervals Z_i or Z_j has the same ordering relationship with all $x_i \in X_i$ and the same ordering relationship with all $y_j \in Y_j$. We say two elements z_1, z_2 in Z are *indistinguishable* if they have the same ordering relationships to elements in X_i and Y_j, the same evaluation on unary relations, and have no non-ordering relationships involving any elements of X_i or Y_j. For any two indistinguishable z_1, z_2, for any $x \in X_i$, $y \in Y_j$ if the inner-most formula is true for x, y, z_1, then it is also true for x, y, z_2. Thus, the formula, including the innermost quantifier, is true for x, y if and only if it is true relative to a maximal set of distinguishable elements. There are only $O(g)$ elements z with some relation to the two intervals, $O(g)$ in the two intervals, and only constantly many equivalence classes of others under indistinguishability. As a preprocessing step, in linear time, we can for each boolean combination of unary predicates for z, create a sorted list of intervals with such elements. Then we can use binary search to see whether such an element exists in some interval before i, between i and j, or after j. We can find z's with a relation to either X_i or Y_j by searching all tuples involving such elements. Thus, we can construct in time $O(g + \log n)$ a maximal set of $O(g)$ distinguishable elements for a given pair of intervals.

For special intervals, we use the quadratic time baseline algorithm on them, taking time $O(g^2)$. So the total time is $O(m \cdot g^2)$.

For non-special intervals, we know there are no relations involving elements in the two intervals. Assume without loss of generality that $i \leq j$. We create unary predicates on z that say whether z comes from an interval before i, interval i, an interval between i and j, j, or after j. Call these five predicates $A_1(z), \ldots, A_5(z)$.

Recall that the formula is of the form: $\exists y Q_3 z \psi(x, y, z)$. We can further divide the sets X_i

and Y_j into constantly many subsets $X_{i,\alpha}$ and $Y_{j,\beta}$ according to the set of unary relations α and β that are true for x and for y respectively. If for each pair of α, β, we compute the list of $x \in X_{i,\alpha}$ so that $\exists(y \in Y_{j,\beta})Q_3 z\psi(x,y,z)$, the final output is the union of these lists. For each, we can replace all unary predicates of x and of y in ψ to get equivalent formulas $\psi_1(x,y,z), \ldots, \psi_5(x,y,z)$ when $A_i(z)$ is true on z.

We can divide the z's up into the five cases given by the new unary predicates $A_1, \ldots, A_5$. In each case, we will show that ψ can be simplified.

For the different cases, ψ_i will be simpler in different ways. ψ_1, ψ_3, and ψ_5 will have no occurrences of the ordering relation. ψ_2 will be a function only of x and z, and ψ_4 only of y and z, but may have ordering predicates.

If Q_3 is $\exists$, our output list is the union of the corresponding lists for the five cases above. (The list for the fourth case is either all x or none.) We will use the baseline algorithm to compute those for cases 2 and 4 in $O(g)$ time. The other three cases are size $O(g)$ instances of 3-quantifier unordered first-order statements, and so we can use the best algorithm for model checking on unordered structures for these.

If Q_3 is $\forall$, we only want to consider x, y so that all five conditions are true simultaneously. We can use the baseline algorithm to compute the subset X' of x so that the second case holds, and the subset Y' of y so that the fourth case holds. Then we restrict the universe to those subsets, and solve the other three cases: find the set of $x \in X'$ so that $\exists(y \in Y')\forall z[\wedge_{\ell=1,3,5}(A_\ell(z) \rightarrow \psi_\ell(x,y,z))]$

To finish the proof, we need to construct the simplified formulas φ_ℓ for the five cases. First note that in all cases, we have fixed all unary relations on x and y, and for each two distinct intervals, any x and y in the intervals have the same order, so we can also fix ordering relations between x and y.

For any z not in interval i or j, the ordering between z and any x in X_i is fixed, and the same for any $y \in Y_j$. (For some z that occur before i, we might have equivalence to elements in X_i, for some, they are strictly smaller, so what this fixed relationship is does depend on z.) We can introduce six new unary relations on z coding this ordering with respect to x and with respect to y.

So for cases 1, 3 and 5, we can replace any ordering relation between x and z or y and z with the corresponding unary predicate of z. This removes all ordering relations to obtain ψ_1, ψ_3 and ψ_5.

To create ψ_2, we are restricting to z in interval i. This fixes the ordering information between z and any $y \in Y_j$, but not between z and x. However, since the pair is not special, there are no relations that involve both y and z. Thus, we can replace those relations by *false* in ψ. Similarly, there are no relations that involve both x and y, so we can replace those relations by *False* as well. In addition, we have already fixed the unary predicates of y, and comparisons between y and x or z. Thus, the restricted formula now has no occurrences of y at all, so is a predicate $\psi_2(x,z)$.

The case for ψ_4 is symmetric.

In summary, fix a combination of α, β, then for a pair of non-special interval X_i, Y_j, we first decide ψ_2: list x so that a formula on x,z holds (in this case, y is indistinguishable to x and $z \in Z_i$ so the variable y is omitted). Next, decide ψ_4: list y so that a formula on y,z holds (here x is indistinguishable to y and $z \in Z_i$ so we omit the variable x). Here we get a list of x and a list of y, then on these two lists, we decide ψ_1, ψ_3, ψ_5.

Thus, the total time we spend on this sub-problem is $O(g)$ to solve the second and fourth cases, and then the best algorithm to solve a three quantifier unordered query on $O(g)$ sized inputs for the other three cases . So for each non-special pair of i, j, the time for φ_2 is $O(g) + T_{FO}(O(g))$. For special pairs, the time is $T_{FO}(g^2)$. There are at most $O(m)$ special pairs, and at most $O((m/g)^2)$ non-special pairs of i, j. We spent time $O(m^2/g)$ to solve the "high degree" case. So the total time is $O(m + m^2/g + (m/g)^2 \cdot T_{FO}(g) + m \cdot g^2)$.

If $s(m)$ is a polynomial improvement factor, i.e., $T_{FO}(m) = m^{2-\varepsilon}$, then by letting $g = m^{\frac{1}{2+\varepsilon}}$ the running time is $O(m^{2-\frac{\varepsilon}{2+\varepsilon}})$.

By the algorithm for OV in [AWY15, CW16] and the algorithm for FOP_k in Chapter 3, we get an improved algorithm for $\mathsf{FOP}_k(\leq)$ in time $m^k/2^{\Theta(\sqrt{\log m})}$.

4.7 FO with Transitive Closure on Symmetric Input Relations

Consider the model checking of a first-order formula with transitive closures, where the transitive closure operation can only be taken on symmetric input relations. In this case $TC_R(x,y)$ is true iff x and y are in the same connected component by edges of undirected edge set R. Thus the formula can have binary predicates about whether two variables are in the same connected component or not. Note that there can be more than one symmetric relations that the transitive closure operation can be taken on.

Theorem 13. *For non-decreasing* $2^{\Omega(\sqrt{\log m})} \leq s(m) \leq m$ *and* $k \geq 3$*, if* FOP_3 *is in time* $O(m^2/s(m))$*, then* $\mathsf{FOP}_k(TC_{sym})$ *is solvable in time* $O(m^{k-1}/s(\mathsf{poly}(m)))$.

Proof. Like the previous section, here we consider the case $k = 3$, and demonstrate a subquadratic time reduction to FOP_3.

Let there be t different TC relations, where t is a constant integer. Let $T_t(m)$ be the running time on instances with t TC relations. We will reduce an instance with t TC relations to instances with $t-1$ TC relations. Here we pick one of the TC relations, and will only deal with this TC relation. The goal is to reduce to instances without this TC relation.

Here we assume that in the input, each vertex has a "category", indicating which connected component it is in. The formula φ contains predicates of form $TC_R(x,y)$ to represent that x and y are in the same connected component by edges of relation R.

Let $\varphi = Q_1xQ_2yQ_3z\psi(x,y,z)$ Let the quantifier Q_2 be $\exists$, otherwise we will decide the negation of the formula.

ψ can be transformed into a disjunction of itself with $TC_R(x,y), TC_R(x,z)$ and $TC_R(y,z)$ replaced by all combinations of *true* and *false* respectively. Let C be the set of all the combinations (there are at most $2^3 = 8$ combinations).

So we consider the model checking of

$$\varphi = Q_1x\exists yQ_3z\,[\bigvee_{c\in C}(\psi_c \wedge (x,y,z \text{ satisfy } c))]$$

ψ_c is ψ with all TC predicates replaced by *true* or *false* according to c.

Let g be a threshold value. First of all, find out all x of degree greater than g. On each of the x, we solve a 2 quantifier problem in time $O(m)$ by the baseline algorithm. Next, find all y of degree greater than g. On each of the y, treating y as a constant we using the baseline algorithm we can list all satisfying x in time $O(m)$, and thus we can check for all x whether there exists a satisfiable y. There are at most $O(m/g)$ such large degree elements, so the total time to deal with these elements is $O(m^2/g)$.

Next, list all elements in $X \cup Y$ so that elements in the same category are listed consecutively. Furthermore, we list all categories of total degree greater than g before the other small categories. Partition the big list into $O(m/g)$ groups so that each has total degree at most g. We make sure that each category of total degree at least g is broken into groups that contain no elements from other categories. The small-degree categories are merged together to make groups of total degree as near to g as possible. In the i-th group, let the sets of elements in X, Y be X_i, Y_i respectively.

For each pair of sets (X_i, Y_j) and each truth value combination c, do the following case analysis.

- **Case 1:** If all elements of X_i and Y_j are from the same category, and c implies $x \in X_i, y \in Y_i$ are in the same category, then all pairs of x, y satisfy c. Next, find the set edges between X_i, Z and between Y_i, Z satisfying c, and make a query to the oracle solving $t-1$ TC relation problems. The running time is $T_{t-1}(g)$.

- **Case 2:** If X_i and Y_j have elements from the same categories and also elements from different categories, then i and j must be equal or adjacent integers. In this case we just query the baseline algorithm, so the time is $O(g^2)$.

- **Case 3:** If elements in X_i and elements in Y_j are from completely different categories, and c implies x, y should be in different categories, then all pairs of x, y satisfy c. This case is similar as the first case.

By some preprocessing, for any pair of i, j, it is easy to tell which of the above three cases the pair (X_i, Y_j) is in.

There are $O((m/g)^2)$ instances of time $T_{t-1}(g)$, and $O(m/g)$ instances of time $O(g^2)$. The total time is $O((m/g)^2 \cdot T_{t-1}(g) + (m/g) \cdot g^2 + m^2/g)$. If $T_{t-1}(m) = m^2/s_{t-1}(m)$ then by taking $g = m^{1/2}$ there is $T_t(m) = O(m^2/s_{t-1}(m^{1/2}))$. Letting $T_t(m) = m^2/s_t(m)$, we get $s_t(m) = O(s_{t-1}(m^{1/2})) = O(s_{t-2}((m^{1/2})^{1/2})) = \cdots = O(s(m^{1/2^t}))$. Thus $T_t(m) = O(m^2/s(m^{1/2^t}))$. □

4.8 Open problems

1. A very general type of open problem is to see whether other complexity classes have complete problems under fine-grained reductions, or give evidence that there are no such complete problems. To make sense, we need a stratification of the class where each layer in the stratification has a reasonable conjecture for its worst-case complexity. For example, we could look at $\mathsf{SPACE}(k \log n)$, with conjectured complexity n^k, if we restrict the tape alphabet to binary.

2. MC_φ where φ is a variable complexity k formula with only unary and binary predicates can be written in a straightline program whose intermediate relations have arity at most 2. Thus it can be solved in $\tilde{O}(n^{k-3+\omega})$ time, and when $k \geq 9$, it can be solved in time $n^{k-1+o(1)}$, by [Wil14a]. When the input is sparse, will there be better algorithms? From the space complexity point of view, it possible to succinctly represent the intermediate relations without explicitly listing $O(n^2)$ tuples?

3. What is the best algorithm for FO formulas with t function symbols, where t is a fixed small constant, such as 2 or 3? In this case, can a 2-quantifier problem be solved in time m^2 or faster? One idea is to use grouping-reduction to eliminate one function each time, but this time we cannot guarantee that on any pair of groups, the relations on their function values are still sparse compared to the group size.

4. The first-order logic can be extended to more expressive classes in many ways, including least fixed point, and temporal logic. It would be interesting if they can be studied in the fine-grained complexity context.

4.9 Baseline Algorithms

Lemma 4.9.1. *For any integer $k \geq 2$, FOP_k, $\mathsf{FOP}_{qr=k}$ and $\mathsf{FOP}_k(\leq)$ are in time $O(mn^{k-2})$. The model checking for k-quantifier formulas with transitive closure operations only on symmetric input binary relations is also in time $O(mn^{k-2})$.*

The baseline algorithm for FOP_k is proved in Section 3.9.1. For $\mathsf{FOP}_{\mathrm{qr}=k}$ and $\mathsf{FOP}_k(\leq)$, we only need to show the case where $k = 2$. The cases for $k > 2$ follows from the quantifier-eliminating downward self-reduction.

The linear time baseline algorithm for $\mathsf{FOP}_{\mathrm{qr}=2}$ is straightforward from the baseline algorithm for FOP_2. Let the variables in the outer scopes be x and the one in the inner scopes be y. For each variable named x and each variable named y, we can compute $\#(x) = |\{y \in y \mid \psi(x,y)\}|$ for any quantifier-free formula $\psi(x,y)$. Thus, for a variable x, we can list all elements in the domain of x satisfying any $\varphi(x)$ where φ has quantifier rank 1. Thus we can decide $\exists x\varphi(x)$ and $\forall x\varphi(x)$ for any variable named x.

The linear time baseline algorithm for $\mathsf{FOP}_2(\leq)$ is also adapted from the baseline algorithm for FOP_2. We prove that $\mathsf{FOP}_2(\leq) \subseteq \mathsf{TIME}(m)$. Let $\varphi = Q_1 x\, Q_2 y\, \psi(x,y)$, where

$$\psi(x,y) = ((x < y) \wedge \psi_<(x,y)) \vee ((x = y) \wedge \psi_=(x,y)) \vee ((x > y) \wedge \psi_>(x,y))$$

for each x, we let $Y_<, Y_=, Y_>$ be the subsets of Y less than x, equal to x or greater than x. These sets are disjoint. The proof for FOP_2 showed that each time taking one $x \in X$, we can compute $\#_<(x) = |\{y \in Y_< \mid \psi(x,y)\}|$ (and similarly $\#_=(x), \#_>(x)$ for $Y_=, Y_>$ respectively) in time linear to the degree of x. Thus for each x, if y is quantified by $\exists$ then we check whether $\#_<(x) + \#_=(x) + \#_>(x) > 0$.

If y is quantified by $\forall$ then we check whether $\#_<(x)+\#_=(x)+\#_>(x)=|Y|$.

For k-quantifier formulas with transitive closure operations only on symmetric input relations, it is easy to see that using the same method, for each x we can count the number of y satisfying a relation and also satisfying a constant number of predicates about whether x and y are in the same connected component by a certain undirected edge relation.

4.10 Baseline Algorithm for Variable Complexity k

4.10.1 Variable Complexity 2

Lemma 4.10.1. $\mathsf{FOP}_{vc=2} \subseteq \mathsf{TIME}(m)$

For the complement of a sparse relation, we call it a *co-sparse relation*. If a relation R is co-sparse, we can represent it by its complement $\bar{R}$, which takes only $O(m)$ space.

First, convert the variable complexity 2 formula φ into a constant number of first-order queries of at most 2 variables. Each query is one of the following forms.

1. $R(x,y) = R_i(x,y) \wedge R_j(x,y)$
2. $R(x,y) = R_i(x,y) \vee R_j(x,y)$
3. $R(x,y) = \neg R_i(x,y)$
4. $R(x) = \exists y R_i(x,y)$
5. $R(x) = \forall y R_i(x,y)$
6. $R = \exists x R_i(x)$
7. $R = \forall x R_i(x)$
8. $R = \neg R_i$

We will show that if we have already computed all of the previous queries in $O(m)$ time, then we can compute the current query (or its negation, if it is co-sparse) in $O(m)$ time.

1. Intersection

   - The intersection of sparse relation $R_i(x,y)$ and sparse relation $R_j(x,y)$ can be computed in time $O(\min(|R_i|,|R_j|))$, by going through the shorter list of R_i and R_j, and check if the tuple satisfies the other relation.
   - The intersection of sparse $R_i(x,y)$ and co-sparse $R_j(x,y)$ can be computed in time $O(|R_i|)$, by going through all tuples of R_i, and check if the tuple satisfies the other relation.
   - The intersection of co-sparse $R_i(x,y)$ and co-sparse $R_j(x,y)$ can be computed in time $O(|\bar{R}_i + \bar{R}_j|)$, by letting $\bar{R}_t$ be the union of $\bar{R}_i$ and $\bar{R}_j$.

2. Union

   It is reducible to the intersection of two relations, by De Morgan's Law.

3. Existential quantifier

   - $R(x) = \exists y R_i(x,y)$ for sparse $R_i(x,y)$ can be computed in time $O(|R_i|)$ by going through all tuples of R_i and list all x that appear in some tuple.
   - $R(x) = \exists y R_i(x,y)$ for co-sparse $R_i(x,y)$ can be computed by taking the complement of $\forall y \bar{R}_i(x,y)$.

4. Universal quantifier

   - $R(x) = \forall y R_i(x,y)$ for sparse $R_i(x,y)$ can be computed in time $O(|R_i|)$ by going through all tuples of R_i, for each x count how many tuples it is in, and finally list all x that appear in $|Y|$ tuples.
   - $R(x) = \forall y R_i(x,y)$ for co-sparse $R_i(x,y)$ can be computed by taking the complement of $\exists y \bar{R}_i(x,y)$.

4.10.2 3 and More Variables

Any formula of 3 quantifiers is can be solved in time $O(n^{\omega})$ [Wil14a]. By applying this algorithm on every line, we can decide a formula of variable complexity 3 in the amount of time. This can be generalized to any $k \geq 3$.

Theorem 14. *(Strengthening of Williams' algorithm) Any first-order property defined by a formula of variable complexity k is decidable in time $\tilde{O}(n^{k-3+\omega})$ for $k \geq 2$. For $k \geq 9$, it can be decided in $n^{k-1+o(1)}$ time.*

SETH implies k-OV for $k \geq 2$ requires time $n^{k-o(1)}$. Therefore, assuming SETH, for $2 \leq k < 8$, it is impossible to express k-OV in FO using only k variables, without blowing up the input size by a polynomial factor.

4.10.3 Case Analysis on FO with Three Variables

Now we consider formulas of variable complexity 3.

If a ternary intermediate relation is created in the straightline program in a line of form $R(x,y,z) = \psi_i(x,y,z)$, then ψ must be quantifier-free. Whenever this ternary intermediate relation is used in other places, we can just replace the relation by ψ_i. Thus we can without loss of generality assume that all intermediate relations are unary or binary.

We define three types of variable complexity 3 formulas.

1. **Strongly Succinct:** The formula is equivalent to a straightline program where all intermediate relations are unary. One example is to decide if there exists a length ℓ chain of orthogonal vectors: $v_1 \perp v_2, v_2 \perp v_3, \ldots, v_{\ell-1} \perp v_\ell$ for some constant ℓ.

   We call it succinct because the intermediate relations can be succinctly listed. Because in computing the straightline program, no dense relation is created, we can use the algorithm for "List-$\exists$-$\forall$" problems for each line. Thus, it is subquadratic time reducible to OV by the reduction in Chapter 3.

2. **Weakly Succinct:** The formula is equivalent to a straightline program where in each line there is at most one occurrence of an intermediate binary relation. Most natural problems of variable complexity 3 are in this case. The formula we have constructed for the constant-depth circuit satisfiability is weakly succinct. $\mathsf{FOP}_{\mathrm{qr}=3}$ problems are also weakly succinct.

   We call it weakly succinct because some of the intermediate relations can be succinctly listed. In this case, it can be solved in $O(mn)$ time for sparse graphs. The proof will be presented in the end of this section.

3. **Non-succinct:** The formula is not weakly succinct. Our best algorithm needs to explicitly construct all intermediate relations. In this case, it is solvable in matrix multiplication time, and in $O(n^3)$ using combinatorial algorithms.

   An example is that, in a sparse graph, decide whether there is a pair $(x \in X, y \in Y)$ such that for all $z \in Z$, z either has a neighbor not adjacent to x, or has a neighbor not adjacent to y. In FO, it is

$$\exists x \exists y \forall z (\exists y' \neg E(x,y') \wedge E(z,y')) \vee (\exists x' \neg E(y,x') \wedge E(z,x'))$$

   which is equivalent to the straightline program

$$\begin{aligned} R_1(y,z) &= \exists y' (\neg E(x,y') \wedge E(z,y')) \\ R_2(x,z) &= \exists x' (\neg E(y,x') \wedge E(z,x')) \\ \varphi &= \exists x \exists y \forall z (R_1(y,z) \vee R_2(x,z)) \end{aligned}$$

   An equivalent problem is: Given three families of sets, A, B, C, decide if for all $\forall S_1 \in A, \forall S_2 \in B$, there is a set $S_3 \in C$ contained in the intersection of A and B.

$$\forall S_1 \in A \forall S_2 \in B \exists S_3 \in C (S_3 \subseteq S_1 \cap S_2)$$

In FO,

$$\forall x \in A \forall y \in B \exists z \in C(\forall y'(y' \in x \vee \neg y' \in z) \wedge \forall x'(x' \in y \vee \neg x' \in z))$$

In matrix multiplication time, we can compute a list of all the pairs of S_1 and S_3 where S_3 is contained in S_1, and a list of all the pairs of S_2 and S_3 where S_3 is contained in S_2, then we can compute for all pairs of S_1 and S_2 whether there is a common S_3 using matrix multiplication. However, listing the pairs of S_1, S_3 and S_2, S_3 generates two dense relations that can be as large as n^2. So it is open whether non-succinct formulas can be decided in $O(m^2)$ time.

Proof for the $O(mn)$ upper bound for weakly succinct formulas.

Consider each line of the straightline program $R(x,y) = Q_z\psi(x,y,z)$. Without loss of generality assume Q_z is $\exists$, for otherwise we can compute the complement of R. Furthermore, we also assume $\psi(x,y,z)$ is a conjunction. For otherwise, we will write $\psi(x,y,z)$ in DNF, and consider each conjunction separately.

Case 1: The intermediate relation is on x,y.

Enumerate all x, for each x, we can list y such that $Q_z\psi(x,y,z)$ holds for x,y, using the $O(m)$ time baseline algorithm. The time is $O(mn)$.

Case 2: The intermediate relation is on y,z.

Case 2-1: Some binary predicate on x,z appears positively in the conjunction.

We enumerate the edges of the positive predicate, to get the pairs of (x,y), and for each of them, enumerate all z, and check if $\psi(x,y,z)$ holds. This takes time $O(mn)$.

Case 2-2: All binary predicates on x,z appear negatively in the conjunction.

Let ψ' be the subformula of the conjunction that contains all predicates on y,z, including the intermediate relation.

For each y, count how many z satisfy ψ' with y. Let the sum be $f(y)$. The total time is $O(n^2)$.

Enumerate all edges on x,z and enumerate all y. In this way we can count for each pair of (x,y) the number of z so that ψ' is satisfied on y,z and also there is an edge between x,z. Let the

sum be $g(x,y)$. The total time is $O(mn)$.

For each pair of x,y, the value $f(y) - g(x,y)$ is the number of z such that ψ' is true and also there are no edges between x,z.

Case 2-3: There are no binary predicates on x,z.

Then there must be binary predicates on x,y, or otherwise x is isolated from other variables in the formula, making it easier to decide. We do a similar counting argument as Case 2-2.

Case 3: The intermediate relation is on x,z.

This case is equivalent to Case 2, because we can switch variables x and y in the formula.

4.11 Acknowledgments

Chapter 4 contains material from "The Fine-Grained Complexity of Strengthenings of First-Order Logic", by Jiawei Gao and Russell Impagliazzo, which is currently in submission. The author of this dissertation was a principal author of this work. The authors sincerely thank Marco Carmosino and Antonina Kolokolova for comments on improving this paper.

Chapter 5

Reachability on Tree-Like DAGs, and Applications to Dynamic Programming Problems

5.1 Chapter Overview

5.1.1 Extending One-Dimensional Dynamic Programming to Graphs

The Least Weight Subsequence (LWS) is type of dynamic programming problems introduced by [HL87]: select a set of elements from a linearly ordered set so that the total cost incurred by the adjacent pairs of elements is optimized. It is defined as follows: Given elements $x_0, \ldots, x_n$, and an $n \times n$ matrix C of costs $c_{i,j}$, for all pairs of indices $i < j$, compute F on all elements, defined by

$$F(j) = \begin{cases} 0, \text{ for } j = 0 \\ \min_{0 \le i < j}[F(i) + c_{i,j}], \text{ for } j = 1, \ldots, n \end{cases}$$

$F(j)$ is the optimal cost value from the first element up to the j-th element. Because the cost matrix C is sometimes fixed by the problem, instead of given by the input, we use the notation LWS_C to define LWS with a specific cost matrix C. The Airplane Refueling problem [HL87] is a well

known example of LWS: Given the locations of airports on a line, find a subset of the airports for an airplane to add fuel, that minimizes the sum of the cost. The cost of flying from the i-th to the j-th airport is defined by $c_{i,j}$. Other LWS examples include finding a longest chain satisfying some property, such as Longest Increasing Subsequence [Fre75] and Longest Subset Chain [KPS17]; breaking a linear structure into blocks, such as Pretty Printing [KP81]; variations of Subset Sum such as the Coin Change problem, and the Knapsack problem. These problems have $O(n^2)$ time algorithms using dynamic programming, and in many special cases it can be improved: when the cost satisfies quadrangle inequality or some other properties, there are near linear time algorithms (e.g. [Yao80, Wil88, GP89]). But for the general LWS, it is not known whether these problems can be solved faster than $n^{2-o(1)}$ time.

A general approach to understanding the fine-grained complexity of these problems was initiated in [KPS17]. Many LWS problems have succinct representations of $c_{i,j}$. Taking problems defined in [KPS17] as examples, in LOWRANKLWS, $c_{i,j} = \langle \mu_i, \sigma_j \rangle$, where μ_i and σ_j are boolean vectors of length $d \ll n$ associated with each element. The CHAINLWS problem has costs $c_1, \ldots, c_n$ defined a property P so that $c_{i,j}$ equals c_j if $P(i,j)$ is true, and ∞ otherwise. P is computable by data associated with the pair (i,j). (For example, in LONGESTSUBSETCHAIN, $P(i,j)$ is true iff set S_i is contained in set S_j.) So the goal of the problem becomes finding a longest chain of elements so that adjacent elements that are to be selected satisfy property P. When C can be represented succinctly, we can ask whether there exist subquadratic time algorithms for these problems, or try to find subquadratic time reductions between problems. [KPS17] showed that in many LWS_C problems when C can be succinctly described in the input, in subquadratic time it is reducible to a corresponding problem, which is called a STATICLWS$_C$ problem. The problem STATICLWS$_C$ is: given elements $x_1, \ldots, x_{2n}$, a cost matrix C, and values $F(i)$ on all $i \in \{1, \ldots, n\}$, compute $F(j) = \min_{i \in \{1,\ldots,n\}}[F(i) + c_{i,j}]$ for all $j \in \{n+1, \ldots, 2n\}$. It is a parallel, batch version (with many values of j rather than a single one) of the update rule applied sequentially one index at a time in the standard DP algorithm. The reduction from LWS$_C$ to STATICLWS$_C$ implies that a highly sequential problem can be reducible to a highly

parallel one. If a STATICLWS$_C$ problem can be solved faster than quadratic time, so can the LWS$_C$ problem. Apart from one-directional reductions from LWS$_C$ to STATICLWS$_C$, [KPS17] also proved subquadratic time equivalence between some concrete problems: LOWRANKLWS is equivalent to MIN INNER PRODUCT, NESTED BOXES is equivalent to VECTOR DOMINATION, LONGEST SUBSET CHAIN is equivalent to ORTHOGONAL VECTORS, and CHAINLWS, which is a generalization of NESTED BOXES and LONGEST SUBSET CHAIN, is equivalent to SELECTION, a generalization of VECTOR DOMINATION and ORTHOGONAL VECTORS.

Some of the LWS problems can be naturally extended from lines to DAGs. For example, on a road map, we wish to find a path for a vehicle, along which we wish to find a sequence of cities where the vehicle can rest and add fuel so that the cost is minimized. The cost of traveling between cities x and y is defined by cost $c_{x,y}$. Connections between cities could be a general graph, not just a line. Works about algorithms for LWS problems on graphs include [AST94, Sch98, CWHL11, LjLW12].

Using a similar approach as [KPS17], this dissertation extend the Least Weight Subsequence problems to the Least Weight Subpath (LWSP$_C$) problem whose objective is to find a least weight subsequence on a path of a given directed acyclic graph $G = (V, E)$. Let there be a set V_0 containing vertices that can be the starting point of a sequence. The optimum value on each vertex is defined by:

$$F(v) = \begin{cases} \min(0, \min_{u \rightsquigarrow v}[F(u) + c_{u,v}]), \text{ for } v \in V_0 \\ \min_{u \rightsquigarrow v}[F(u) + c_{u,v}], \text{ for } v \notin v_0 \end{cases}$$

where $u \rightsquigarrow v$ means u is reachable to v. The goal of LWSP$_C$ is to compute $F(v)$ on all vertices $v \in V$. Examples of LWSP$_C$ problems will be given in Appendix A. LWSP$_C$ can be solved in time $O(|V| \cdot |E|)$ by doing reversed depth/breadth first search from each vertex, and update the F value on the vertex accordingly. It is not known whether it has faster algorithms, even for Longest Increasing Subsequence, which is an LWS$_C$ instance solvable in $O(n \log n)$ time on linear structures. If C is succinctly describable in similar ways as LOWRANKLWS, NESTED BOXES, SUBSET CHAIN or CHAINLWS, we wish to study if there are subquadratic time algorithms or subquadratic time

reductions between problems.

Because the cost matrix C is either fixed in the problem or given succinctly in the input, we consider that every vertex has some additional data so that $c_{x,y}$ can be computed by the data contained in x and y. Let the size of additional data associated with each vertex v be its weight $w(v)$. The weight of a vertex can be defined in different ways according to the problems. For example, in LOWRANKLWS, the weight of an element can be defined as the length of its associated vector; and in SUBSET CHAIN, the weight of an element is the size of its corresponding subset. We use $m = |E|$ as the number of graph edges. Let n be the number of vertices. Let the total weight of all vertices be N. We use $M = \max(m, N)$ as the size of the input. In this dissertation we will see that if we can improve the algorithm for STATICLWS$_C$ to be subquadratic time, then on some interesting classes of graphs we can solve LWSP$_C$ faster than $M^{2-o(1)}$ time.

5.1.2 Introducing Reachability to First-Order Model Checking

Introducing transitive closure to first-order logic is analogous to extending LWS$_C$ to paths in graphs, which makes parallel problems become sequential. The first-order property (or first-order model checking) problem is to decide whether an input structure satisfies a fixed first-order logic formula φ. Although model checking for input formulas is PSPACE-complete [Sto74, Var82], when φ is fixed by the problem, it is solvable in polynomial time. The sparse version of OV is one of these problems, defined by the formula $\exists u \exists v \forall i \in [d](\neg One(u,i) \vee (\neg One(v,i)))$, where relation $One(u,i)$ is true iff the i-th coordinate of vector u is one.

If φ has k quantifiers ($k \geq 2$), then on input structures of n elements and m tuples of relations, it can be solved in time $O(n^{k-2}m)$ [GIKW17]. On dense graphs where $k \geq 9$, it can be solved in time $O(n^{k-3+\omega})$, where ω is the matrix multiplication exponent [Wil14a]. Here we study the case where the input structure is sparse, i.e. $m = n^{1+o(1)}$, and ask whether a three-quantifier first-order formula can be model checked in time faster than $m^{2-o(1)}$. The *first-order property conjecture (FOPC)* states that there exists integer $k \geq 2$, so that first-order model checking for $(k+1)$-quantifier

formulas cannot be solved in time $O(m^{k-\varepsilon})$ for any $\varepsilon > 0$. This conjecture is equivalent to MDOVC, since OV is proven to be a complete problem in the class of first-order model checking problems; in other words, any model checking of 3 quantifier formulas on sparse graphs is subquadratic time reducible to OV [GIKW17]. This means from improved algorithms for OV we can get improved algorithms for first-order model checking.

The first-order property problems are highly parallelizable. If we introduce the transitive closure (TC) operation on the relations, then these problems will become sequential. The transitive closure of a binary relation E can be considered as the reachability relation by edges of E in a graph. In a sparse structure, the TC of a relation may be dense. So it can be considered as a dense relation succinctly described in the input. In finite model theory, adding transitive closure significantly adds to the expressive power of first-order logic (First discovered by Fagin in 1974 according to [Lib13], and then re-discovered by [AU79].) In fine-grained complexity, adding arbitrary transitive closure operations on the formulas strictly increases the hardness of the model checking problem. More precisely, [GI19] shows that SETH on constant depth circuits, which is a weaker conjecture than the SETH on k-CNF-SAT, implies the model checking for two-quantifier first-order formulas with transitive closure operations cannot be solved in time $O(m^{2-\varepsilon})$ for any $\varepsilon > 0$. This means this problem may stay hard even if the SETH on k-CNF-SAT is refuted.

However, we will see that for a class of three-quantifier formulas with transitive closure, model checking is no harder than OV under subquadratic time reductions.

We define problem $\textsc{Selection}_P$ to be the decision problem for whether an input structure satisfies $(\exists x \in X)(\exists y \in Y)P(x,y)$. $P(x,y)$ is a fixed property specified by the problem that can be decided in time $O(w(x)+w(y))$, where $w(x)$ is the size of additional data on element x. For example, OV is $\textsc{Selection}_P$ where $P(x,y)$ iff x and y are a pair of orthogonal vectors. In this case $w(x)$ is defined as the length of vector x. (If we work on the sparse version of OV, the weight $w(x)$ is defined by the Hamming weight of x.)

On a directed graph $G = (V,E)$, we define $\textsc{Path}_P$ to be the problem of deciding whether $(\exists x \in V)(\exists y \in V)[\mathrm{TC}_E(x,y) \wedge P(x,y)]$, where TC_E is the transitive closure of relation E and $P(x,y)$

is a property on x,y fixed by the problem. That is, whether there exist two vertices x,y not only satisfying property P but also x is reachable to y. We will give an example of PATH$_P$ in Appendix A. Also, we define LISTPATH$_P$ to be the problem of listing all $x \in V$ such that $(\exists y \in V)[\mathrm{TC}_E(x,y) \wedge P(x,y)]$.

Considering the model checking problems, we let PATHFO$_3$ and LISTPATHFO$_3$ denote the class of PATH$_P$ and LISTPATH$_P$ such that P is of form $\exists z\psi(x,y,z)$ or $\forall z\psi(x,y,z)$, where ψ is a quantifier-free formula in first-order logic. [1] Later we will see that problems in PATHFO$_3$ and LISTPATHFO$_3$ are no harder than OV. In these model checking problems, the weight of an element is the number of tuples in the structure that the element is contained in.

Trivially, SELECTION$_P$ on input size $|X| = N_1, |Y| = N_2$ can be decided in time $O(N_1N_2)$, where N_1 is the total weight of elements in X, and N_2 is the total weight of elements in Y. PATH$_P$ and LISTPATH$_P$ on input size M and total vertex weight N are solvable time $O(MN)$ by depth/breadth first search from each vertex, where M is defined to be the maximum of N and the number of edges m. In this dissertation we will show that on some graphs, if SELECTION$_P$ is in truly subquadratic time, so is PATH$_P$ and LISTPATH$_P$. Interestingly, by applying the same reduction techniques from PATH$_P$ to SELECTION$_P$, we can get a similar reduction from a dynamic programming problem on a graph to a static problem.

5.1.3 Main Results

This dissertation works on two classes of graphs, both having some similarities to trees. The first class is where the graph G is a multitree. A *multitree* is a directed acyclic graph where the set of vertices reachable from any vertex form a tree. Or equivalently a DAG is a multitree if and only if on all pairs of vertices u,v, there is at most one path from u to v. In different contexts, multitrees are also called *strongly unambiguous graphs*, *mangroves* or *diamond-free posets* [GLL12]. These graphs can be used to model computational paths in nondeterministic algorithms where there is at

[1] The formal definition of PATHFO$_3$ and LISTPATHFO$_3$ will be in the full version of this paper.

most one path connecting any two states [AL96]. The butterfly network, which is a widely-used model of the network topology in parallel computing, is an example of multitrees.

The second class of graphs is when we treat G as undirected by replacing all directed edges by undirected edges, the underlying graph has constant treewidth. *Treewidth* [RS84, RS86] is an important parameter of graphs that describes how similar they are to trees. [2] On these classes of graphs, we have the following theorems.

Theorem 15 (Reductions between decision problems.)**.** *Let $t(M) \geq 2^{\Omega(\sqrt{\log M})}$, and let the DAG $G = (V,E)$ satisfy one of the following conditions:*

- *G is a multitree, or*
- *G is a multitree of strongly connected components, or*
- *The underlying undirected graph of G has constant treewidth,*

then, the following statements are true:

- *If* SELECTION$_P$ *is in time $N_1N_2/t(\min(N_1,N_2))$, then* PATH$_P$ *is in time $M^2/t(\mathsf{poly}M)$.*[3]
- *If* PATH$_P$ *is in time $M^2/t(M)$, then* LISTPATH$_P$ *is in time $M^2/t(\mathsf{poly}M)$.*
- *When $P(x,y)$ is of form $\exists z\psi(x,y,z)$ or $\forall z\psi(x,y,z)$ where ψ is a quantifier-free first-order formula,* SELECTION$_P$ *is in time $N_1N_2/t(\min(N_1,N_2))$ iff* PATH$_P$ *is in time $M^2/t(\mathsf{poly}M)$ iff* LISTPATH$_P$ *is in time $M^2/t(\mathsf{poly}M)$.*

This theorem implies that OV is hard for classes PATHFO$_3$ and LISTPATHFO$_3$. By the improved algorithm for OV [AWY15, CW16], we get improved algorithms for PATHFO$_3$ and LISTPATHFO$_3$:

[2] Here we consider the undirected treewidth, where both the graph and the decomposition tree are undirected. It is different from *directed treewidth* defined for directed graphs by [JRST01].

[3] This reduction also applies to optimization versions of these two problems. Let Path_F be a problem to compute $\min_{x,y\in V, x\rightsquigarrow y} F(x,y)$ and $\mathsf{Selection}_F$ be a problem to compute $\min_{x\in X, y\in Y} F(x,y)$, where F is a function on x,y, instead of a boolean property. Then the same technique gives us a reduction from Path_F to $\mathsf{Selection}_F$. We will leave the details to the full version of the paper.

Corollary 5.1.1 (Improved algorithms.)**.** *Let the graph G be a multitree, or multitree of strongly connected components, or a DAG whose underlying undirected graph has constant treewidth. Then* PATHFO$_3$ *and* LISTPATHFO$_3$ *are in time* $M^2/2^{\Omega(\sqrt{\log M})}$.

Next, we consider the dynamic programming problems. If the cost matrix C in LWSP$_C$ is succinctly describable, we get the following reduction from LWSP$_C$ to STATICLWS$_C$. Except for the reduction from LONGESTSUBSETCHAIN to OV, we always assume that all vertices have the same weight.[4]

Theorem 16 (Reductions between optimization problems.)**.** *On a multitree graph, or a DAG whose underlying undirected graph has constant treewidth, let* $t(N) \geq 2^{\Omega(\sqrt{\log N})}$*, then,*

1. *if* STATICLWS$_C$ *of total weight N is in time* $N^2/t(N)$*, then* LWSP$_C$ *on input size M is in time* $M^2/t(\mathsf{poly}(M))$.
2. *if* LWSP$_C$ *is in time* $M^2/t(M)$*, then* LWS$_C$ *on input size N is in time* $N^2/t(\mathsf{poly}(N))$.

If there is a reduction from a concrete STATICLWS$_C$ problem to its corresponding LWS$_C$ problem (e.g. from MININNERPRODUCT to LOWRANKLWS, from VECTOR DOMINATION to NESTED BOXES and from OV to LONGEST SUBSET CHAIN [KPS17]), then the corresponding LWS$_C$, STATICLWS$_C$ and LWSP$_C$ problems are subquadratic-time equivalent. From the algorithm for OV [CW16] and SPARSE OV [GIKW17], we get an improved algorithm for problem LONGEST SUBSET CHAIN:

Corollary 5.1.2 (Improved algorithm)**.** *On a sparse multitree graph or a DAG whose underlying undirected graph has constant treewidth,* LONGEST SUBSET CHAIN *is in time* $M^2/2^{\Omega(\sqrt{\log M})}$.

The reduction uses a technique that decomposes multitrees into sub-structures where it is easy to decide whether vertices are reachable. So we also get reachability oracles using subquadratic space, that can answer reachability queries in sublinear time.

[4] In LONGESTSUBSETCHAIN, the subsets corresponding to different vertices can have different sizes.

Theorem 17 (Reachability oracle)**.** *On a multitree of strongly connected components, there exists a reachability oracle with subquadratic preprocessing time and space that has sublinear query time. On a multitree, the preprocessing time and space is $O(m^{5/3})$, and the query time is $O(m^{2/3})$.*

5.1.4 Organization of this Chapter

In Section 5.2 we prove the first part of Theorem 15, by reduction from PATH_P to SELECTION_P on multitrees. The case for bounded treewidth DAGs will be presented in Section 5.2.4. Section 5.3 proves Theorem 16 by presenting a reduction from LWSP_C to STATICLWS_C. Section 5.6 discusses about open problems. Appendix 5.4 proves the second part of Theorem 15 by reduction from LISTPATH_P to PATH_P. Appendix 5.5 proves the last part of Theorem 15, the subquadratic equivalence of SELECTION_P, PATH_P and LISTPATH_P when P is a first-order property. Appendix 5.7 talks about the reachability oracle for multitrees.

5.2 From Sequential Problems to Parallel Problems

This section will establish the first part of Theorem 15 by showing that if $t(M) \geq 2^{\Omega(\sqrt{\log M})}$, then $(\text{PATH}_P, M^2/t(\text{poly}M)) \leq_{\text{EC}} (\text{SELECTION}_P, N_1N_2/t(\min(N_1,N_2)))$. We will give the reduction for multitrees and multitrees of strongly connected components.

5.2.1 The Recursive Algorithm

In the algorithm we first remove high degree vertices, then follow a divide and conquer strategy. For simplicity of description, we will consider each strongly connected component as a single vertex, whose weight equals the total weight of the component. Thus we will be working on a multitree, instead of a multitree of strongly connected components. In the following algorithm, whenever querying SELECTION_P or exhaustively enumerating pairs of reachable vertices and testing P on them, we will extract all the vertices from a component. Testing P on a pair of vertices (or

strongly connected components) of weights N_1, N_2 is in time $O(N_1 N_2)$.

Let CUTPATH$_P$ be a variation of PATH$_P$. It is the property testing problem for $(\exists x \in S)(\exists y \in T)[TC_E(x,y) \wedge \varphi(x,y)]$, where (S,T) is a cut in the graph, such that all the edges between S and T are directed from S to T. CUTPATH$_P$ on input size M and total vertex weight N can be solved in time $O(MN)$ if $P(x,y)$ is decidable in time $O(w(x)+w(y))$: start from each vertex and do depth/breadth first search, and on each pair of reachable vertices decide if P is satisfied.

Lemma 5.2.1. *For $t(M) \geq 2^{\Omega(\sqrt{\log M})}$, if* SELECTION$_P(N_1,N_2)$ *is in time $N_1 N_2/t(\min(N_1,N_2))$ and* CUTPATH$_P(M)$ *is in time $M^2/t(M)$, then* PATH$_P(M)$ *is in time $M^2/t(\mathsf{poly}(M))$.*

Proof. Let γ be a constant satisfying $0 < \gamma \leq 1/4$. Let $T_\Pi(M)$ be the running time of problem Π on a structure of size M, and let $T_{\text{SELECTION}_P}(N_1,N_2)$ be the running time of SELECTION$_P$ on a pair of sets (X,Y) where the total vertex weight of X is N_1 and of Y is N_2.

We show that there exists a constant c where $0 < c < 1$ so that if $T_{\text{SELECTION}_P}(N_1,N_2) \leq N_1 N_2/t(\min(N_1,N_2))$ and $T_{\text{CUTPATH}_P}(M) \leq M^2/t(M)$, and $T_{\text{PATH}_P}(M')$ is at most $M'^2/t(M'^c)$ for all $M' < M$, then $T_{\text{PATH}_P}(M) \leq M^2/t(M^c)$. We run the recursive algorithm as shown in Algorithm 3. The intuition is to divide the graph into a cut S,T, recursively compute PATH$_P$ on S and T, and deal with paths from S to T. For large-weight vertices, we deal with them separately so that CUTPATH$_P$ will not deal with large-weight vertices.

For the vertices of weight more than M^γ, we deal with them separately as a first step. If a vertex has more than M^γ ancestors/descendants, and if SELECTION$_P$ on size (N_1,N_2) is in time $O(N_1 N_2/t(\min N_1,N_2))$, then the time to deal with a vertex of weight N_i is at most $O(MN_i/t(N_i)) \leq O(MN_i/t(M^\gamma))$. Because all N_i sum to at most M, the total time is $O(M^2/t(M^\gamma))$. If the vertex has less than M^γ ancestors/descendants, then the exhaustive search time on all such v and all their ancestors/descendants sums to at most $O(M \cdot M^\gamma)$. After the computation, the vertex becomes an "auxiliary" vertex. In the upcoming steps we will only use auxiliary vertices as intermediate points in the path, but will not include them in calls to SELECTION$_P$ or treat them as potential endpoints of a path and check P on them. This can be done by keeping a list of vertices to be ignored. There

Algorithm 3: $\textsc{Path}_P(G)$

```
// Reducing PATH_P to SELECTION_P and CUTPATH_P
if G has only one vertex then return false.
Let M be the size of the problem.
for each vertex v of weight ≥ M^γ do
    if v has at least M^γ ancestors then
        Compute SELECTION_P on the set of v's ancestors and v.
    else
        Exhaustively search all pairs of vertices on the set of v's ancestors and v, test P
          on all pairs. If P is true on any pair then return true.
    if v has at least M^γ descendants then
        Compute SELECTION_P on v and the set of v's descendants.
    else
        Exhaustively search all pairs of vertices on v the set of v's descendants, test P on
          all pairs. If P is true on any pair then return true.
    Replace v by an auxiliary vertex of weight 1.
Topological sort all vertices.
Keep adding vertices to S by topological order, until the total weight of S exceeds M/2.
  Let the rest of vertices be T.
Run PATH_P on the subgraph induced by S.
Run CUTPATH_P(S,T).
Run PATH_P on the subgraph induced by T.
if any one of the above three calls returns true then return true.
```

are $O(M^{1-\gamma})$ of vertices of weight more than M^γ, thus the total size of lists is $O(M \cdot M^{1-\gamma}) = M^{2-\gamma}$.

Let M_S and M_T be the sizes of sets S and T respectively. Assume $M_S \geq M_T$ and let $\Delta = M_S - M_T$. Then we have

$$\begin{aligned}
T_{\textsc{Path}_P}(M) &= T_{\textsc{Path}_P}(M_S) + T_{\textsc{Path}_P}(M_T) + T_{\textsc{CutPath}_P}(M) + O(M^2/t(M^\gamma)) \\
&= T_{\textsc{Path}_P}(M_T + \Delta) + T_{\textsc{Path}_P}(M_T) + T_{\textsc{CutPath}_P}(M) + O(M^2/t(M^\gamma)) \\
&\leq 2T_{\textsc{Path}_P}(M/2 + \Delta) + T_{\textsc{CutPath}_P}(M) + O(M^2/t(M^\gamma)) \\
&= 2(M/2+\Delta)^2/t((M/2+\Delta)^c) + M^2/t(M) + O(M^2/t(M^\gamma)).
\end{aligned}$$

Let d be the constant factor of term $O(M^2/t(M^\gamma))$. We can pick c to be small enough so that

$dt(M^c)/t(M^\gamma) = \varepsilon$. Thus the term $d \cdot M^2/t(M^\gamma) = \varepsilon M^2/t(M^c)$. The term $M^2/t(M)$ is less than $M^2/t(M^\gamma)$, so it is also less than $\varepsilon M^2/t(M^c)$. So the running time by the above formula yields to at most $2(M/2+\Delta)^2/t((M/2+\Delta)^c)+2\varepsilon M^2/t(M^c)$. Because the function $M^2/t(M^c)$ is monotonically increasing, the formula is upper bounded by $2(M/2+\Delta)^2/t((M/2+\Delta)^c)+2\varepsilon(M/2+\Delta)^2/t((M/2+\Delta)^c) = (2+2\varepsilon)(M/2+\Delta)^2/t((M/2+\Delta)^c)$. When $\Delta \leq M^\gamma$ for $\gamma < 1/4$ and when M is large enough, $M^\gamma \ll M$ so $M/2+\Delta = (1+o(1))M/2$. So $(2+2\varepsilon)(M/2+\Delta)^2$ can be significantly less than M^2. Moreover, we can make $(2+2\varepsilon)(M/2+\Delta)^2/M^2$ less than $t((M/2+\Delta)^c)/t(M^c)$ because t grows very slowly. Thus we get $(2+2\varepsilon)(M/2+\Delta)^2/t((M/2+\Delta)^c) \leq M^2/t(M^c)$. □

5.2.2 A Special Case that Can Be Exhaustively Searched

The following lemma shows that if no vertex has both a lot of ancestors and a lot of descendants, then the total number of reachable pairs of vertices is subquadratic to m. This lemma holds for any DAG, not just for multitrees. We will use this lemma in the next subsection to show that in a subgraph where all vertices have few ancestors and descendants, we can test property P on all pairs of reachable vertices by brute force.

Lemma 5.2.2. *If in a DAG $G = (V,E)$ of m edges, every vertex has either at most n_1 ancestors or at most n_2 descendants, then there are at most $(m \cdot n_1 \cdot n_2)$ pairs of vertices s,t such that s is reachable to t.*

In a DAG $G = (V,E)$ of m edges, let S,T be two disjoint sets of vertices where edges between S and T only direct from S to T. If every vertex has either at most n_1 ancestors in S or at most n_2 descendants in T, then there are at most $(m \cdot n_1 \cdot n_2)$ pairs of vertices $s \in S$ and $t \in T$ such that s is reachable to t.

Proof. We define the ancestors of an edge $e \in E$ to be the ancestors (or ancestors in S) of its incoming vertex, and its descendants to be the descendants (or descendants in T) of its outgoing vertex. Let the number of its ancestors and descendants be denoted by $anc(e)$ and $des(e)$ respectively.

For each edge e, it belongs to exactly one of the following three types:

Type A: If $anc(e) \leq n_1$ but $des(e) > n_2$, then let $count(e)$ be $anc(e)$.

Type B: If $des(e) \leq n_2$ but $anc(e) > n_1$, then let $count(e)$ be $des(e)$.

Type C: If $anc(e) \leq n_1$ and $des(e) \leq n_2$, then let $count(e)$ be $anc(e) \cdot des(e)$.

$\sum_{e \in E} count(e) \leq m \cdot n_1 \cdot n_2$ because the *count* value on each edge is bounded by $n_1 \cdot n_2$. We will prove that this value upper bounds the number of reachable pairs of vertices.

For each pair of reachable vertices (u,v) (or (u,v) s.t. $u \in S$ and $v \in T$), let $(e_1, \ldots, e_p)$ be the path from u to v. Along the path, *anc* does not decrease, and *dec* does not increase. A path belongs to exactly one of the following three types:

Type a: Along the path $anc(e_1) \leq anc(e_2) \leq \cdots \leq anc(e_p) \leq n_1$, and $des(e_1) \geq des(e_2) \geq \cdots \geq des(e_p) > n_2$. That is, all the edges are Type A.

Type b: Along the path $des(e_p) \leq des(e_{p-1}) \leq \cdots \leq des(e_1) \leq n_2$, and $anc(e_p) \geq anc(e_{p-1}) \geq \cdots \geq anc(e_1) > n_1$. That is, all the edges are Type B.

Type c: Along the path there is some edge e_i so that $anc(e_i) \leq n_1$ and $des(e_i) \leq n_2$. That is, it has at least one Type C edge.

There will not be other cases, for otherwise if a Type A edge directly connects to a Type B edge without a Type C edge in the middle, then the vertex joining these two edges would have more than n_1 ancestors and more than n_2 descendants.

If a path from u to v is Type a, then its last edge e_p is Type A. If it is Type b, then its first edge e_1 is Type B. If it is Type c, then there is some edge e_i in the path that is Type C. This means:

1. For each Type A edge e, $count(e)$ is at least the number of all Type a pairs (u,v) whose path has e as its last edge.

2. For each Type B edge e, $count(e)$ is at least the number of all Type b pairs (u,v) whose path has e as its first edge.

3. For each Type C edge e, $count(e)$ is at least the number of all Type c pairs (u,v) whose path contains e.

Therefore each path is counted at least once by the $count(e)$ of some edge e. □

5.2.3 Subroutine: Reachability Across a Cut

Now we will show the reduction from CUTPATH$_P$ to SELECTION$_P$, where the graph does not contain vertices of weight greater than M^γ. The high level idea of CUTPATH$_P$ is that we think of the reachability relation on $S \times T$ as an $|S| \times |T|$ boolean matrix whose one-entries correspond to reachable pairs of vertices. If we could partition the matrix into all-one combinatorial rectangles, then we can decide all entries within these rectangles by a query to SELECTION$_P$, because in the same rectangle, all pairs are reachable.

Claim 5.2.1. *Consider the reachability matrix of on sets S and T. Let M_S and M_T be the sizes of S and T. If there is a way to partition the matrix into non-overlapping combinatorial rectangles $(S_1,T_1),\ldots,(S_k,T_k)$ of sizes $(r_1,c_1),\ldots,(r_k,c_k)$, and if there is some t so that computing each subproblem of size (r_i,c_i) takes time $r_i \cdot c_i/t(\min(r_i,c_i))$, and each r_i and c_i are at least ℓ, then all the computation takes total time $O(M_S \cdot M_T/t(\ell))$.*

Proof. Let the minimum of all r_i be r_{min} and the minimum of all c_i be c_{min}. Then the factor of time saved for computing each combinatorial rectangle is at least $t(\min(r_{min},c_{min}))$, greater than $t(\ell)$. So the time spent on all rectangles is at most $O((\sum_{i=1}^t c_i)(\sum_{i=1}^t r_i)/t(\ell))$, also we have $(\sum_{i=1}^t c_i)(\sum_{i=1}^t r_i) \leq M_S \cdot M_T$ because the rectangles are contained inside the matrix of size $M_S \cdot M_T$ and they do not overlap. So the total time is $O(M_S \cdot M_T/t(\ell))$. □

The algorithm CUTPATH$_P(S,T)$ is shown in Algorithm 4. It tries to cover the one-entries of the reachability matrix by combinatorial rectangles as many as possible. Finally, for the one-entries not covered, we go through them by exhaustive search, which takes less than quadratic time.

Algorithm 4: CUTPATH$_P(S,T)$ on a multitree

Count the number of ancestors $anc(v)$ and descendants $des(v)$ for all vertices.
Insert all vertices with at least M^α ancestors and M^α descendants into linked list L.
while there exists a vertex $v \in L$ **do**
    // we call v a *pivot vertex*
    Let A be the set of ancestors of v in S.
    Let B be the set of descendants of v in T.
    Add v to A if $v \in S$, otherwise add v to B.
    Run SELECTION$_P$ on (A,B). If it returns true then **return** true.
    for each $a \in A$ **do**
        let $des(a) = des(a) - |B|$.
        if $des(a) < M^\alpha$ and $a \in L$ **then** remove a from L.
    for each $b \in B$ **do**
        let $anc(b) = anc(b) - |A|$.
        if $anc(b) < M^\alpha$ and $b \in L$ **then** remove b from L.
    Remove v from the graph.
for each edge (s,t) crossing the cut(S,T) **do**
    Let A be the set of ancestors of s (including s) in S.
    Let B be the set of descendants of t (including t) in T.
    On all pairs of vertices (a,b) where $a \in A, b \in B$, check property P. If P is true on any pair of (a,b) then **return** true.

In the beginning, we can count the number of ancestors (or descendants) of all vertices in the DAG in $O(M)$ time by going through all vertices by topological order (or reversed topological order).

In each query to SELECTION$_P(A,B)$, all vertices in A can reach all vertices in B, because they all go through v. For any pair of reachable vertices $s \in S, t \in T$, if they go through any pivot vertex, then the pair is queried to SELECTION$_P$. Otherwise it is left to the end, and checked by exhaustive search on all pairs of reachable vertices.

The calls to SELECTION$_P$ correspond to non-overlapping all-one combinatorial rectangles in the reachability matrix. This is because the graph G is a multitree. For each call to SELECTION$_P$, the rectangle size is at least $M^\alpha \times M^\alpha$. Thus the total time for all the $\exists\exists$P calls is $O(M^2/t(M^\alpha))$ by Claim 5.2.1.

Each time we remove a pivot vertex v, there will be no more paths from set A to set B, for

otherwise there would be two distinct paths connecting the same pair of vertices. Thus, removing a v decreases the total number of pairs of reachable vertices by at least $M^\alpha \cdot M^\alpha = M^{2\alpha}$. There are M^2 pairs of vertices, so the total number of pivot vertices like v is at most $M^2/M^{2\alpha} = M^{2-2\alpha}$.

Each time we find a pivot vertex v, we update the number of descendants for all its ancestors, and update the number of ancestors for all its descendants. Because it has at least M^α ancestors and M^α descendants, the value decrease on each affected vertex is at least M^α. So each vertex has decreased its ancestors/descendants values for at most $M/M^\alpha = M^{1-\alpha}$ times. In other words, each vertex can be an ancestor/descendant of at most $M^{1-\alpha}$ pivot vertices. The total time to deal with all ancestors/descendants of all pivot vertices in the while loop is in $O(M \cdot M^{1-\alpha}) = O(M^{2-\alpha})$.

Finally, after the while loop, there are no vertices with both more than M^α ancestors and M^α descendants. In this case, by Lemma 5.2.2 in Section 5.2.2, the total number of reachable vertices is bounded by $M \cdot M^\alpha \cdot M^\alpha = M^{1+2\alpha}$. Each vertex has weight at most M^γ. So the total time to deal with these paths is $O(M^{1+2\alpha} \cdot M^\gamma \cdot M^\gamma) = O(M^{1+2\alpha+2\gamma})$.

Thus the total running time is $O(M^2/t(M^\alpha) + M^{2-\alpha} + M^{1+2\alpha+2\gamma})$. By choosing α and γ to be appropriate constants, we get subquadratic running time.

If $t(M) = M^\varepsilon$, then by choosing $\alpha = \gamma = 1/(4+\varepsilon)$, we get running time $M^{2-\varepsilon/(4+\varepsilon)}$.

5.2.4 CUTPATH$_P$ for Bounded-Treewidth DAGs

We prove the first part of Theorem 15 on DAGs whose underlying undirected graphs have constant treewidth. The algorithm PATH$_P$ for constant treewidth graphs is the same as the one for multitrees. In this section we will show the reduction algorithm CUTPATH$_P$ for constant treewidth graphs on a cut (S,T).

Let $\mathcal{T}$ be the decomposition tree of a graph G. Recall that by the definition of tree decomposition, each node z of the tree corresponds to a set $\mathcal{B}(z)$ which is a subset of vertices of G. Because the treewidth is constant, each set $\mathcal{B}(z)$ has a constant number of vertices. Every vertex of G appears in at least one set of a tree node. Also, for every edge of G, there is at least one tree node whose set

contains both its endpoints. And if a vertex v appears both in $\mathcal{B}(z_1)$ and $\mathcal{B}(z_2)$, then along the path from z_1 to z_2, v must appear in all the sets of the tree nodes. Here we consider the decomposition tree as rooted, where all edges are directed from the root to leaves.

We use a similar reduction idea as Section 5.2.3. In the decomposition tree, each time we find a node z to split the tree into two connected components. We first deal with all the paths that go through the vertices in $\mathcal{B}(z)$. Any other path in the graph must be completely contained in one of the connected components we have created. In the end, all connected components are so small that we can go through all pairs of reachable vertices by exhaustive search. The algorithm is defined in Algorithm 5.

The following claim uses a $1/3-2/3$ trick on trees:

Claim 5.2.2. *In a rooted tree of size n, we can find a connected subgraph of size between* $(1/3)n$ *and* $(2/3)n$ *in* $O(n)$ *time.*

Proof. For each node z in the tree, we will compute the size of the subtree rooted at z, denoted by $f(z)$. We compute $f(z)$ from the leaves up to the root, by a reversed topological order. If z is a leaf then let $size(z) \leftarrow 1$.

On each parent node p, we initially let $f(p) \leftarrow 1$, and then for each child c_i of p, add the value $f(c_i)$ to $f(c_i)$. If before we add the $f(c_i)$ of certain child c_i to $f(p)$, $f(p) < (1/3)n$, and after we add $f(c_i)$ to $f(p)$, $f(p) \geq (1/3)n$, then there are two cases:

If $f(p) \leq (2/3)n$, then the subgraph formed by p and its subtrees $c_1, \ldots, c_i$ is the connected subgraph we want.

If $f(p) > (2/3)n$, then it must be $f(c_i) \geq (2/3)n - (1/3)n = (1/3)n$. That is, the subtree rooted at c_i has size between $(1/3)n$ and $(2/3)n$. But then we should have already returned the subtree rooted c_i instead. So this case would not happen.

After we have added the sizes of all the children of p to $f(p)$, we have finished computing $f(p)$. If $f(p)$ is still less than $1/3$, we will continue to let the next vertex by the reversed topological order be the current parent. □

Algorithm 5: CUTPATH$_P(S,T)$ on constant treewidth DAG

Compute $\mathcal{T}$, the tree decomposition of the underlying undirected graph.
for each z in $\mathcal{T}$ **do**
    Let $size(z)$ be the number of nodes of $\mathcal{T}$.
while there exists a tree node z in $\mathcal{T}$ so that there is a connected subgraph of $\mathcal{T}$ rooted at z with size between $(1/3)size(z)$ and $(2/3)size(z)$ **do**
    `// z can be found in time` $O(size(z))$ `by Claim 5.2.2.`
    for each $v \in \mathcal{B}(z)$ **do**
        `// Deal with all paths going through` v.
        Let A be the set of ancestors of v in S.
        Let B be the set of descendants of v in T.
        Add v to A if $v \in S$, otherwise add v to B.
        if both A and B have at least M^α vertices **then**
            Run SELECTION$_P$ on (A,B). If it returns true then **return** true.
        else
            Exhaustively check P on all pairs of $a \in A$ and $b \in B$. If P is true on any (a,b) then **return** true.
        Remove v from the graph, and from the sets of all the tree nodes.
    Remove z from $\mathcal{T}$.
    for each tree node z' who was originally in the same connected component with z **do**
        Update $size(z')$ to be the new size of the connected component z' is in.
for each edge (s,t) crossing the cut(S,T), **do**
    Let A be the set of ancestors of s (including s) in S.
    Let B be the set of descendants of t (including t) in T.
    On all pairs of vertices (a,b) where $a \in A, b \in B$, check property P. If P is true on any pair of (a,b) then **return** true.

Next we will analyze the reduction algorithm. First, if a the treewidth of a graph is constant, then the corresponding decomposition tree can be computed in linear time [Bod96].

Unlike multitree graphs, here the calls to SELECTION$_P$ are not non-overlapping rectangles: different v from the same $\mathcal{B}(z)$ may share the same ancestors or descendants. However, each time after removing a z, the connected components of the decomposition tree correspond to non-overlapping rectangles in the reachability matrix, and will not overlap with the rectangles corresponding to the ancestors and descendants for any $v \in \mathcal{B}(z)$. Thus, the overlapping only happens when dealing with the ancestors and descendants of different v from the same $\mathcal{B}(z)$,

and these SELECTION$_P$ rectangles will not overlap with other SELECTION$_P$ rectangles after z is removed. Because in each non-overlapping rectangle corresponding to a connected component, we only computed the SELECTION$_P$ for $|\mathcal{B}(z)|$ times, which is a constant. So by Claim 5.2.1, the total time spent on all the calls to SELECTION$_P$ is still $O(M^2/t(M^\alpha))$.

When we remove all vertices $v \in \mathcal{B}(z)$, the graph vertices from sets of different connected components of the decomposition tree are not reachable to each other. Because any path from one connected component to another must go through some vertex in $\mathcal{B}(z)$.

Unlike multitree graphs, this time some vertex v in $\mathcal{B}(z)$ may have fewer than M^α ancestors or descendants. If so, then we do exhaustive search on the sets of v's ancestors and descendants, since calling SELECTION$_P$ will not save time. Each time we find a v, the connected component of the decomposition tree that v belongs to loses at least $(1/3)size(v)$ of its vertices, thus each vertex can be the ancestor/descendants of at most $O(\log_{3/2} M)$ such v's. There are at most M vertices in the graph, each of which can take part in at most M^α such paths going through each such v. So the total time is $O(M \cdot \log_{3/2} M \cdot M^\alpha) = O(M^{1+\alpha} \cdot \log_{3/2} M)$.

Also, because each vertex can be the ancestor/descendants of at most $O(\log_{3/2} M)$ such v's, the total time for updating *size* for all of them is also bounded by $O(M \cdot \log_{3/2} M)$.

In the end, each remaining vertex has $O(M^\alpha)$ ancestors and $O(M^\alpha)$ descendants. The total running time for the exhaustive search is $O(M \cdot M^\alpha \cdot M^\alpha \cdot M^{2\gamma}) = O(M^{1+2\alpha+2\gamma})$ by Lemma 5.2.2.

The overall running time is $O(M^2/t(M^\alpha) + M^{1+\alpha} \cdot \log_{3/2} M + M^{1+2\alpha+2\gamma})$. By choosing α and γ to be appropriate small constants, we get subquadratic running time.

5.3 Application to Least Weight Subpath

In this section we will prove Theorem 16. The reduction from LWSP$_C$ to STATICLWS$_C$ uses the same structure as the reduction from PATH$_P$ to SELECTION$_P$ in the proof of Theorem 15 shown in Section 5.2.

Process LWSP$_C(G, F_0)$ computes values of F on initial values F_0 defined on all vertices of

G. On a given LWSP_C problem, we will reduce it to an asymmetric variation of STATICLWS_C. Process $\text{STATICLWS}_C(A,B,F_A)$ computes all the values of function F_B defined on domain B, given all the values of F_A defined on domain A, such that $F_B(b) = \min_{a\in A}[F_A(s)+c_{a,b}]$. Let N_A and N_B be the total weight of A and B respectively. It is easy to see that if STATICLWS_C on $|N_A| = |N_B|$ is in time $N_A^2/t(N_A)$, then STATICLWS_C on general A,B is in time $O(N_A \cdot N_B/t(\min(N_A,N_B)))$.

We also define process $\text{CUTLWSP}_C(S,T,F_S)$, which computes all the values of F_T defined on domain T, given all the values of F_S on domain S, where $F_T(t) = \min_{s\in S, s\rightsquigarrow t}[F_S(s)+c_{s,t}]$.

The reduction algorithm is adapted from the reduction from PATH_P to SELECTION_P. LWSP_C is analogous to PATH_P, STATICLWS_C is analogous to SELECTION_P, and CUTLWSP_C is analogous to CUTPATH_P. In PATH_P, we divide the graph into two halves, recursively call PATH_P on the subgraphs, and use CUTPATH_P to deal with paths from one side of the graph to the other side. Similarly in LWSP_C, we divide the graph into two halves, recursively compute function F on the source side of the graph, then based on these values we call CUTPATH_P to compute the initial values of function F on the sink side of the graph, and finally we recursively call LWSP_C on the sink side of the graph. In CUTPATH_P, we first identify large all-one rectangles in the reachability matrix, and then use SELECTION_P to solve them, and finally we go through all reachable pairs of vertices that are not covered by these rectangles. Similarly, in LWSP_C, we will use the similar method to identify large all-one rectangles in the reachability matrix and use STATICLWS_C to solve them, and finally we go through all reachable pairs of vertices and update F on each of them.

The algorithm LWSP_C is similar as PATH_P, and is defined in Algorithm 6. Initially, we let $F(v) \leftarrow 0$ for all $v \in V_0$, and let $F(v) \leftarrow +\infty$ for all $v \notin V_0$. We run $\text{LWSP}_C(G,F_0)$ on the whole graph. Here we only consider the case where all vertices have the same weight. (For SUBSETCHAIN the subset associated with each vertex can have different sizes. But by the universe-shrinking self reduction in [GIKW17] we can transform the universe of the sets to be as small as $2^{\Theta(\sqrt{\log n})}$ for problems with n subsets. By expressing the set using a vector of length equal to the size of the small universe, we will make all vertices have the same weight.)

The algorithm $\text{CUTLWSP}_C(S,T,F_S)$ is adapted from CUTPATH_P, with the following changes:

Algorithm 6: $\mathrm{LWSP}_C(G=(V,E,V_0),F_0)$

if G has only one vertex v **then**
  if $v \in V_0$ **then**
    return $\min(0, F_0(v))$.
  return F_0 on v.
Let M be the size of the problem.
Topological sort all vertices.
Keep adding vertices to S by topological order, until the total weight of S exceeds $M/2$. Let the rest of vertices be T.
Compute F on domain S, by $F \leftarrow \mathrm{LWSP}_C(G_S, F_0)$, where G_S is the subgraph of G induced by S.
Let $F_T \leftarrow \textsc{CutLWSP}_C(S,T,F)$.
For each $t \in T$, let $F_0(t) \leftarrow \min(F_0(t), F_T(t))$.
Compute F on domain T, by $F \leftarrow \mathrm{LWSP}_C(G_T, F_0)$, where G_T is the subgraph of G induced by T.
return F on domain V.

1. In the beginning, $F_T(t)$ is initialized to ∞ for all $t \in T$.

2. Each query to $\textsc{Selection}_P(A,B)$ in $\textsc{CutPath}_P$ is replaced by

   (a) Compute F_B on domain B by $\textsc{StaticLWS}_C(A,B,F_S)$.

   (b) For each vertex b in B, let $F_T(b)$ be the minimum of the original $F_T(b)$ and $F_B(b)$.

3. Whenever processing a pair of vertices s,t such that s is reachable to t in either the preprocessing phase or the final exhaustive search phase, we let $F_T(t) \leftarrow F_S(s) + c_{s,t}$ if $F_S(s) + c_{s,t} < F_T(t)$.

4. In the end, the process returns F_T, the target function on domain T.

Correctness of $\textsc{CutLWSP}_C$.

The correctness of $\textsc{CutLWSP}_C$ follows from the correctness of $\textsc{CutPath}_P$. We claim that after running $\textsc{CutLWSP}_C(S,T,F_S)$, for any vertex $t \in T$, there is $F_T(t) = \min_{s \in S, s \rightsquigarrow t}[F_S(s) + c_{s,t}]$. Because for any pair $s \in S$, $t \in T$, such that s reachable to t, they are either processed in a query to $\textsc{StaticLWS}_C(A,B)$ where $s \in A, t \in B$, or computed separately thus $F_T(t) \leftarrow \min(F_T(t), F(s) + c_{s,t})$.

Correctness of LWSP$_C$.

The LWSP$_C$ algorithm has the following facts:

1. Whenever a process LWSP$_C$ on domain $V_1 \subseteq V$ returns, the values of F on V_1 are fixed and will not be changed henceforth.

2. Whenever there is an edge from u to v, then the value of F on u is always fixed before the value on v. So the final values of function F on all vertices are fixed by topological order.

3. Each time we call LWSP$_C$ on a subset of vertices $V_1 \subseteq V$, the F values on all ancestors of any vertex in V_1 that are not in V_1 have been fixed by some previous calls to LWSP$_C$.

Assume that when we call LWSP$_C$ on subgraph with cut (S,T), initially there is

$$F_0(v) = \begin{cases} \min_{u \in R(v) \setminus (S \cup T), u \rightsquigarrow v}[F(u) + c_{u,v}], \text{ if } v \notin V_0 \\ \min(0, \min_{u \in R(v) \setminus (S \cup T), u \rightsquigarrow v}[F(u) + c_{u,v}]), \text{ if } v \in V_0 \end{cases} \tag{5.1}$$

where $R(v)$ is the set of vertices reachable to v. Then, if LWSP$_C(S, F_0)$ is correct, after running LWSP$_C(S, F_0)$, for any $s \in S \setminus V_0$, there is $F(s) = \min_{u \in R(s) \setminus T, u \rightsquigarrow s}[F(u) + c_{u,s}]$. And after running CUTLWSP$_C(S,T,F)$, we have $F_T(t) = \min_{s \in S, s \rightsquigarrow t}[F(s) + c_{s,t}]$. Then after taking $F_0(t) = \min(F_0(t), F_T(t))$ on all t, for any $t \in T \setminus V_0$, we get $F_0(t) = \min_{u \in R(t) \setminus T, u \rightsquigarrow t}[F(u) + c_{u,t}]$. Similarly for any $t \in T \cap V_0$, $F_0(t)$ gets the the minimum of this value and 0. Therefore, on each call of LWSP$_C(V_1, F_0)$ on a subset $V_1 \subset V$ with initial values F_0, F_0 keeps the invariant in formula (5.1).

The time complexity of this reduction algorithm follows from the argument of Section 5.2. Here because all vertices have the same weight and we are dealing with DAGs so there are no strongly connected components. And in PATH$_P$, there will not be the term $M^2/t(M^\gamma)$. The rest of the time analysis is the same as Section 5.2.

5.4 From Listing Problems to Decision Problems

In this section we prove the second part of Theorem 15, that LISTPATH$_P$ is reducible to PATH$_P$.

Consider a star graph, which is a graph with its vertex set partitioned in X, Y and another single vertex c. Every $x \in X$ is connected to c, and c is connected to every $y \in Y$. Let problem FINDX$_P$ be the following problem: on a star graph, find an $x \in X$ satisfying $(\exists y \in Y)P(x,y)$. We will prove that LISTPATH$_P$ is reducible to FINDX$_P$ and FINDX$_P$ is reducible to PATH$_P$.

Lemma 5.4.1. *Let* $t(M) \geq 2^{\Omega(\sqrt{\log M})}$. $(\text{LISTPATH}_P, M^2/(t(\text{poly}M))) \leq_{EC} (\text{FINDX}_P, M^2/t(M)))$

Proof. We use a grouping reduction technique similar as the trick in [WW10] and [AWW16].

We modify the algorithm for PATH$_P$ in Section 5.2 to get the algorithm for LISTPATH$_P$. That is, we divide the graph into two subgraphs and call LISTPATH$_P$ recursively in a similar wa as PATH$_P$. PATH$_P$ needs to call SELECTION$_P$ as queries, and in the counterpart of LISTPATH$_P$ we will call FINDX$_P$ as queries.

Whenever we need to call SELECTION$_P(X,Y)$, we partition X and Y into groups of size at most $\sqrt{M}$. Thus there are $O((|X|/\sqrt{M}) \times (|Y|/\sqrt{M}))$ groups. For each pair of group X_i, Y_j, we construct a star graph and call FINDX$_P$ on it. The star graph is constructed as follows: Connect every $x \in X_i$ to a dummy vertex c, and connect c to every $y \in Y_j$. Thus if there exist some satisfying x in X_i, FINDX$_P$ will find a satisfying x.

Every time a satisfying vertex x in X_i is found by FINDX$_P$, we mark it and add it into the list of satisfying x, and then call the FINDX$_P$ on the same star again with x removed from the graph. We keep calling FINDX$_P$ on this graph, ignoring all marked vertices, until either all elements in X_i are marked and removed, or FINDX$_P$ cannot find a satisfying x.

Because there are at most M vertices that can be listed, there are at most M calls to FINDX$_P$ that returns a satisfying x. Each call has instance size $\sqrt{M}$. The running time is $O(M \cdot (\sqrt{M})^2/t(\sqrt{M}))$. The total time spent on the rest of the algorithm is the same as the running time of PATH$_P$. □

Lemma 5.4.2. *Let* $t(M) \geq 2^{\Omega(\sqrt{\log M})}$. $(\text{FINDX}_P, M^2/(t(\text{poly}M))) \leq_{EC} (\text{PATH}_P, M^2/t(M)))$

Proof. First, we pick an arbitrary element $x_1 \in X$, and construct a graph by letting x_1 connect to all y in Y. Then we call PATH_P on this graph. If it returns yes, then we return x_1.

Otherwise, on the star graph we will replace the center vertex c by x_1, remove the original x_1, and call PATH_P on this graph. After each call to PATH_P, if it returns yes, we divide X in two halves and call PATH_P again. Using binary search and shrinking the size of X by half each time, we will finally find a satisfying x. □

Lemmas 5.4.1 and 5.4.2 imply the reduction

$$(\text{LISTPATH}_P, M^2/(t(\text{poly}M))) \leq_{\text{EC}} (\text{PATH}_P, M^2/t(M))) \text{ for } t(M) \geq 2^{\Omega(\sqrt{\log M})}.$$

5.5 From Parallel Problems to Sequential Problems

We prove the third part of Theorem 15, the other direction of the reduction. The reduction from PATH_P to LISTPATH_P is straightforward.

To reduce from SELECTION_P to PATH_P, we can construct a graph with dummy vertex c in the middle, such that each x in set X is connected to c, and c is connected to every y in set Y. If P is expressible by first-order logic, then we will let c act like one of the y's when computing $R(x,c)$, and act like of the x's when computing predicates on $P(c,y)$. Let x_1 be an arbitrary element in X, and y_1 be an arbitrary element in Y. We create c by merging x_1 and y_1 into a single element. c has all the relations x_1 and y_1 have. Thus, on any $x \in X, x \neq x_1$, the value of $P(x,c)$ is the same as $P(x,y_1)$. Symmetrically on any $y \in Y, y \neq y_1$, the value of $P(c,y)$ is the same as $P(x_1,y)$. Therefore, there exists x,y such that $P(x,y)$ is true iff SELECTION_P on this graph returns true.

In general, if we are allowed to define another property P' such that $P'(x,y) \leftarrow (P(x,y) \wedge (x \neq c) \wedge (y \neq c))$, we have a reduction from SELECTION_P to $\text{Path}_{P'}$.

5.6 Open problems

One open problem is to extend PATH_P and LWSP_C to general DAGs and find subquadratic time reductions and equivalences. Also, we would like to consider the case where the graph is not sparse, where we use $O(MN)$ as the baseline time complexity instead of $O(M^2)$.

It would also be desirable to study the fine-grained complexity of the DAG versions of other quadratic time solvable dynamic programming problems, e.g. the Longest Common Subsequence problem.

5.7 Reachability Oracle

This section presents a proof of Theorem 17. A reachability oracle on a graph takes in a pair of vertices u, v in the graph, and answers whether v is reachable from u. A naive approach is to use $O(n^2)$ space to store the reachability of all pairs of vertices. By adapting the PATH_P algorithm on multitrees, we get sublinear time reachability oracles for multitrees using subquadratic space and subquadratic preprocessing time. If the graph is a multitree of strongly connected components, we can first treat each strongly connected component as a single vertex, whose weight is the total weight of all vertices in the component.

The reachability oracle for multitrees can be adapted directly from the PATH_P algorithm. In the recursion tree of calling PATH_P, we take down the final subproblem that each vertex belongs to, and when querying a pair of vertex, we find the PATH_P instance corresponding to the least common ancestor of the two final subproblems corresponding to these vertices, and consider the CUTPATH_P process called by this PATH_P instance.

Next we modify the CUTPATH_P algorithm. Among all pivot vertices, we call the ones who have no other pivot vertices as descendants "sink pivot vertices". After computing the number of ancestors and descendants for all vertices, we can decide if a vertex is a sink in time linear to the degree of the vertex.

We create another graph G'. Similar to CUTPATH$_P$, we keep finding pivot vertices who have at least M^α ancestors and M^α descendants in the remaining graph, and then remove them. Whenever finding a pivot vertex v, we create edges from all its ancestors to v, and from v to all its descendants in G'.

Thus, when querying a pair of vertices a, b, if a can reach b, there are three cases:

- Case 1: b is a pivot vertex. Then there is an edge from a to b in G'.
- Case 2: The path from a to b goes through at least a pivot vertex. In this case, it must go through a sink pivot vertex. We decide if there is a sink pivot vertex v adjacent to a in G' who is also adjacent to b. Each vertex can be an ancestors of at most M/M^α sink pivot vertices, because each sink pivot vertex has more than M^α descendants, and different sink pivot vertices have disjoint set of descendants. So this checking can be done in time M/M^α.
- Case 3: The path from a to b does not go through any pivot vertex. Then we can do a DFS starting from a that traverses through at most M^α of a's descendants in the remaining graph G to find b. The time taken is $O(M^\alpha)$.

Thus, the query time is $O(M/M^\alpha + M^\alpha)$, which is sublinear to M.

If the graph is a multitree, not a multitree of strongly connected components, then every vertex has unit weight. In this case, the modified CUTPATH$_P$ process runs in time $O(m^2/m^\alpha + m^{2-\alpha} + m^{1+2\alpha})$, because now we do not use SELECTION$_P$ thus the function $t(m)$ is $O(m)$, and there are no large-weight vertices thus we can pick $\gamma = 0$. If we choose $\alpha = 1/3$, then the running time is $O(m^{5/3})$. The modified PATH$_P$ algorithm has running time satisfying the recursion $T(m) = 2T(m/2) + O(m^{5/3})$, which is $O(m^{5/3})$. So the preprocessing time and space is $O(m^{5/3})$, and the query time is $O(m^{2/3})$.

5.8 Acknowledgments

Chapter 5 contains material from "On the Fine-grained Complexity of Least Weight Subsequence in Multitrees and Bounded Treewidth DAGs", by Jiawei Gao, to appear in International Symposium on Parameterized and Exact Computation (IPEC 2019). The author of this dissertation was a principal author of this publication. The author would like to thank Russell Impagliazzo for his guidance and advice on this paper, and thank Marco Carmosino, Anant Dhayal and Jessica Sorrell for helpful comments.

Bibliography

[ABDN18] Amir Abboud, Karl Bringmann, Holger Dell, and Jesper Nederlof. More consequences of falsifying SETH and the orthogonal vectors conjecture. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 253–266. ACM, 2018.

[ABW15] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*, pages 59–78. IEEE, 2015.

[AGW14] Amir Abboud, Fabrizio Grandoni, and Virginia Vassilevska Williams. Subcubic equivalences between graph centrality problems, apsp and diameter. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 1681–1697. SIAM, 2014.

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of databases: the logical level*. Addison-Wesley Longman Publishing Co., Inc., 1995.

[AHWW16] Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. Simulating branching programs with edit distance and friends: Or: A polylog shaved is a lower bound made. In *Proc. STOC*, pages 375–388. ACM, 2016.

[Ajt83] Miklós Ajtai. Σ^1_1-formulae on finite structures. *Annals of pure and applied logic*, 24(1):1–48, 1983.

[AL96] Eric Allender and Klaus-Jörn Lange. $StUSPACE(\log n) \subseteq DSPACE(\log^2 n/\log\log n)$. In *International Symposium on Algorithms and Computation*, pages 193–202. Springer, 1996.

[AR18] Udit Agarwal and Vijaya Ramachandran. Fine-grained complexity for sparse graphs. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2018, pages 239–252, 2018.

[AST94] Alok Aggarwal, Baruch Schieber, and Takeshi Tokuyama. Finding a minimum-weightk-link path in graphs with the concave monge property and applications. *Discrete & Computational Geometry*, 12(3):263–280, 1994.

[AU79] Alfred V Aho and Jeffrey D Ullman. Universality of data retrieval languages. In *Proceedings of the 6th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 110–119. ACM, 1979.

[AW85] Miklos Ajtai and Avi Wigderson. Deterministic simulation of probabilistic constant depth circuits. In *Foundations of Computer Science, 1985., 26th Annual Symposium on*, pages 11–19. IEEE, 1985.

[AWW14] Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster alignment of sequences. In *Automata, Languages, and Programming*, pages 39–51. Springer, 2014.

[AWW16] Amir Abboud, Virginia Vassilevska Williams, and Joshua Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete Algorithms*, pages 377–391. SIAM, 2016.

[AWY15] Amir Abboud, Ryan Williams, and Huacheng Yu. More applications of the polynomial method to algorithm design. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 218–230. SIAM, 2015.

[AYZ95] Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM (JACM)*, 42(4):844–856, 1995.

[BCH16] Michele Borassi, Pierluigi Crescenzi, and Michel Habib. Into the square: On the complexity of some quadratic-time solvable problems. *Electronic Notes in Theoretical Computer Science*, 322:51–67, 2016.

[BFNW93] László Babai, Lance Fortnow, Noam Nisan, and Avi Wigderson. BPP has subexponential time simulations unless EXPTIME has publishable proofs. *Computational Complexity*, 3:307–318, 1993.

[BI13] Christopher Beck and Russell Impagliazzo. Strong ETH holds for regular resolution. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 487–494, 2013.

[BI15] Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 51–58. ACM, 2015.

[BIS90] David Mix Barrington, Neil Immerman, and Howard Straubing. On uniformity within NC^1. *Journal of Computer and System Sciences*, 41(3):274 – 306, 1990.

[BK15] Karl Bringmann and Marvin Kunnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*, pages 79–97. IEEE, 2015.

[Bod96] Hans L Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on computing*, 25(6):1305–1317, 1996.

[Bri14] Karl Bringmann. Why walking the dog takes time: Fréchet distance has no strongly subquadratic algorithms unless SETH fails. In *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on*, pages 661–670. IEEE, 2014.

[BRS+18] Arturs Backurs, Liam Roditty, Gilad Segal, Virginia Vassilevska Williams, and Nicole Wein. Towards tight approximation bounds for graph diameter and eccentricities. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 267–280. ACM, 2018.

[CGI+16] Marco L Carmosino, Jiawei Gao, Russell Impagliazzo, Ivan Mihajlin, Ramamohan Paturi, and Stefan Schneider. Nondeterministic extensions of the strong exponential time hypothesis and consequences for non-reducibility. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, pages 261–270. ACM, 2016.

[CIKK16] Marco L Carmosino, Russell Impagliazzo, Valentine Kabanets, and Antonina Kolokolova. Learning algorithms from natural proofs. In *31st Conference on Computational Complexity*, 2016.

[CIP02] Moses Charikar, Piotr Indyk, and Rina Panigrahy. New algorithms for subset query, partial match, orthogonal range searching, and related problems. In *Automata, Languages and Programming*, pages 451–462. Springer, 2002.

[CW16] Timothy M Chan and Ryan Williams. Deterministic APSP, Orthogonal Vectors, and More: Quickly derandomizing Razborov-Smolensky. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1246–1255. SIAM, 2016.

[CWHL11] S.C. Chen, J.Y. Wu, G.S. Huang, and R.C.T. Lee. Finding a longest increasing subsequence on a galled tree. In *the 28th Workshop on Combinatorial Mathematics and Computation Theory, Penghu, Taiwan*, 2011.

[DF92] Rodney G Downey and Michael R Fellows. Fixed-parameter intractability. In *Structure in Complexity Theory Conference, 1992., Proceedings of the Seventh Annual*, pages 36–49. IEEE, 1992.

[DF95] Rod G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness i: Basic results. *SIAM J. Comput.*, 24(4):873–921, August 1995.

[Fag76] Ronald Fagin. Probabilities on finite models. *The Journal of Symbolic Logic*, 41(01):50–58, 1976.

[FG06] Jörg Flum and Martin Grohe. Parameterized complexity theory, volume xiv of texts in theoretical computer science. an eatcs series, 2006.

[FJ56] Lester R Ford Jr. Network flow theory. Technical report, DTIC Document, 1956.

[Fre75] Michael L Fredman. On computing the length of longest increasing subsequences. *Discrete Mathematics*, 11(1):29–35, 1975.

[FSS84] Merrick Furst, James B Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17(1):13–27, 1984.

[GI19] Jiawei Gao and Russell Impagliazzo. The fine-grained complexity of strengthenings of first-order logic. *Electronic Colloquium on Computational Complexity (ECCC)*, 26:9, 2019.

[GIKW17] Jiawei Gao, Russell Impagliazzo, Antonina Kolokolova, and Ryan Williams. Completeness for first-order properties on sparse structures with algorithmic applications. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '17, pages 2162–2181, 2017.

[GLL12] Jerrold R Griggs, Wei-Tian Li, and Linyuan Lu. Diamond-free families. *Journal of Combinatorial Theory, Series A*, 119(2):310–322, 2012.

[GO95] Anka Gajentaan and Mark H Overmars. On a class of $O(n^2)$ problems in computational geometry. *Computational geometry*, 5(3):165–185, 1995.

[GP89] Zvi Galil and Kunsoo Park. A linear-time algorithm for concave one-dimensional dynamic programming. 1989.

[HL87] Daniel S Hirschberg and Lawrence L Larmore. The least weight subsequence problem. *SIAM Journal on Computing*, 16(4):628–638, 1987.

[IKW02] Russell Impagliazzo, Valentine Kabanets, and Avi Wigderson. In search of an easy witness: Exponential versus probabilistic time. *Journal of Computer and System Sciences*, 65(69):672–694, 2002.

[IP99] Russell Impagliazzo and Ramamohan Paturi. Complexity of k-SAT. In *Computational Complexity, 1999. Proceedings. Fourteenth Annual IEEE Conference on*, pages 237–240. IEEE, 1999.

[IPZ98] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? In *Foundations of Computer Science, 1998. Proceedings. 39th Annual Symposium on*, pages 653–662. IEEE, 1998.

[JMV15] Hamid Jahanjou, Eric Miles, and Emanuele Viola. Local reductions. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, pages 749–760, 2015.

[JRST01] Thor Johnson, Neil Robertson, Paul D Seymour, and Robin Thomas. Directed tree-width. *Journal of Combinatorial Theory, Series B*, 82(1):138–154, 2001.

[JS99] David S Johnson and Mario Szegedy. What are the least tractable instances of max independent set? In *Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, pages 927–928. Society for Industrial and Applied Mathematics, 1999.

[KI04] Valentine Kabanets and Russell Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Computational Complexity*, 13(1–2):1–46, 2004.

[KP81] Donald E Knuth and Michael F Plass. Breaking paragraphs into lines. *Software: Practice and Experience*, 11(11):1119–1184, 1981.

[KPS17] Marvin Künnemann, Ramamohan Paturi, and Stefan Schneider. On the Fine-Grained Complexity of One-Dimensional Dynamic Programming. In *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 21:1–21:15, 2017.

[Lib13] Leonid Libkin. *Elements of finite model theory*. Springer Science & Business Media, 2013.

[LjLW12] Guan-Yu Lin, Jia jie Liu, and Yue-Li Wang. Finding a longest increasing subsequence from the paths in a complete bipartite graph. 2012.

[LMN93] Nathan Linial, Yishay Mansour, and Noam Nisan. Constant Depth Circuits, Fourier Transform, and Learnability. *J. ACM*, 40(3):607–620, 1993.

[LWW18] Andrea Lincoln, Virginia Vassilevska Williams, and Ryan Williams. Tight hardness for shortest cycles and paths in sparse graphs. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1236–1252. Society for Industrial and Applied Mathematics, 2018.

[MPS16] Daniel Moeller, Ramamohan Paturi, and Stefan Schneider. Subquadratic algorithms for succinct stable matching. In *International Computer Science Symposium in Russia*, pages 294–308. Springer, 2016.

[NW94] Noam Nisan and Avi Wigderson. Hardness vs. Randomness. *Journal of Computer and System Sciences*, 49:149–167, 1994.

[PI00] Pavel Pudlak and Russell Impagliazzo. A lower bound for dll algorithms for k-sat. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2000, San Francisco, CA, USA, January 9-11, 2000*, pages 128–136, 2000.

[RS84] Neil Robertson and Paul D Seymour. Graph minors. iii. planar tree-width. *Journal of Combinatorial Theory, Series B*, 36(1):49–64, 1984.

[RS86] Neil Robertson and P.D Seymour. Graph minors. ii. algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309 – 322, 1986.

[Sch98] Baruch Schieber. Computing a minimum weightk-link path in graphs with the concave monge property. *Journal of Algorithms*, 29(2):204–222, 1998.

[SHI90] Richard Edwin Stearns and Harry B Hunt III. Power indices and easier hard problems. *Mathematical Systems Theory*, 23(1):209–225, 1990.

[Sto74] Larry Joseph Stockmeyer. *The complexity of decision problems in automata theory and logic*. PhD thesis, Massachusetts Institute of Technology, 1974.

[Var82] Moshe Y Vardi. The complexity of relational query languages. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 137–146. ACM, 1982.

[Wil88] Robert Wilber. The concave least-weight subsequence problem revisited. *Journal of Algorithms*, 9(3):418–425, 1988.

[Wil05] Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2):357–365, 2005.

[Wil13] Ryan Williams. Improving exhaustive search implies superpolynomial lower bounds. *SIAM Journal on Computing*, 42(3):1218–1244, 2013.

[Wil14a] Ryan Williams. Faster decision of first-order graph properties. In *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, page 80. ACM, 2014.

[Wil14b] Ryan Williams. Nonuniform ACC Circuit Lower Bounds. *J. ACM*, 61(1):2:1–2:32, 2014.

[Wil16] Virginia Vassilevska Williams. CS267 lecture 1, algorithms for fixed subgraph isomorphism. `http://theory.stanford.edu/~virgi/cs267/lecture1.pdf`, 2016.

[Wil18] Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the ICM*, 2018.

[WW10] Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on*, pages 645–654. IEEE, 2010.

[Yao80] F Frances Yao. Efficient dynamic programming using quadrangle inequalities. In *Proceedings of the twelfth annual ACM symposium on Theory of computing*, pages 429–435. ACM, 1980.

[Yao82] Andrew C Yao. Theory and applications of trapdoor functions. In *Foundations of Computer Science (FOCS)*, pages 80–91, 1982.

Appendix A

Examples of Problems

A.1 Model Checking Problems

Below we list some problems studied in fine-grained complexity, with their first-order definitions on structures with unary and binary relations.

- **Graph problems.** The input structure is $G = (V, E)$ with a universe V and a binary relation E.

  1. *Diameter*-2: $(\forall x_1)(\forall x_2)(\exists x_3)\,[E(x_1,x_3) \wedge E(x_3,x_2)]$.

  2. *Radius*-2: $(\exists x_1)(\forall x_2)(\exists x_3)\,[E(x_1,x_3) \wedge E(x_3,x_2)]$.

  3. *k-Clique*: $(\exists x_1)\ldots(\exists x_k)\left[\bigwedge_{i,j\in\{1,\ldots,k\},i\neq j} E(x_i,x_j)\right]$. More generally, for a fixed graph H of k vertices, deciding if H is a subgraph or induced subgraph of the input graph G (e.g., the k-Independent Set problem) can be expressed in a similar way.

  4. *k-Dominating Set*:$(\exists x_1)\ldots(\exists x_k)(\forall x_{k+1})\left[\bigvee_{i=1}^{k} E(x_i,x_{k+1})\right]$.

- **Set problems.** The inputs are set families $\mathcal{S}$ or $\mathcal{S}_1,\ldots,\mathcal{S}_k$ over a universe U. Here, all sets are given explicitly and represented by first-order variables. These structures contain a single binary predicate $\in$.

1. *Hitting Set*: [1] $(\exists H \in \mathcal{S})(\forall S \in \mathcal{S})(\exists x)\,[(x \in H) \wedge (x \in S)]$.

2. *k-Set Packing:* $(\exists S_1 \in \mathcal{S}) \ldots (\exists S_k \in \mathcal{S})\ (\forall x)\left[\bigvee_{i=1}^{k}\left((x \in S_i) \rightarrow \bigwedge_{j \neq i}(x \notin S_j)\right)\right]$.

3. *k-Empty Intersection* (*k*-OV): $(\exists S_1 \in \mathcal{S}_1) \ldots (\exists S_k \in \mathcal{S}_k)(\forall u \in U)\left[\bigvee_{i=1}^{k} \neg(u \in S_i)\right]$.

4. *k-Set Cover*: $(\exists S_1 \in \mathcal{S}_1) \ldots (\exists S_k \in \mathcal{S}_k)(\forall u \in U)\left[\bigvee_{i=1}^{k}(u \in S_i)\right]$.

5. *Set Containment*: $(\exists S_1 \in \mathcal{S}_1)(\exists S_2 \in \mathcal{S}_2)\ (\forall u \in U)\,[(\neg(u \in S_1)) \vee (u \in S_2)]$.

k-Empty Intersection is equivalent to *k*-OV, and Set Containment is equivalent to Sperner Family problem. See Section 1.3 for definitions and conjectures for variants of the Orthogonal Vectors problem.

A.2 Problems about Reachability

We give a list of problems that can be considered as instances of LWSP_C or PATH_P.

Trip Planning (LWSP version of Airplane Refueling)

On a DAG where vertices represent cities and edges are roads, we wish to find a path for a vehicle, along which we wish to find a sequence of cities where the vehicle can rest and add fuel so that the cost is minimized. The cost of traveling between cities x and y is defined by cost $c_{x,y}$. $c_{x,y}$ can be defined in multiple ways, e.g. $c_{x,y}$ is $cost(y)$ if $dist(x,y) \leq M$ and ∞ otherwise. $dist(x,y)$ is the distance between x, y that can be computed by the positions of x, y. M is the maximal distance the vehicle can travel without resting. $cost(y)$ is the cost for resting at position y.

Longest Subset Chain on graphs (LWSP version of Longest Subset Chain)

On a DAG where each vertex corresponds to a set, we want to find a longest chain in a path of the graph such that each set is a subset of its successor. Here $c_{x,y}$ is -1 if S_x is a subset of S_y, and ∞ otherwise.

Multi-currency Coin Change (LWSP version of Coin Change)

[1] Other versions of Hitting Set where the sets are not given explicitly are second-order logic problems. Our definition here is consistent with the version in the Hitting Set Conjecture.

Consider there are two different currencies, so there are two sets of coins. We need to find a way to get value V_1 for currency #1 and value V_2 for currency #2, so that the total weight of coins is minimized. Each pair of values $v_1 \in \{0,\ldots,V_1\}$ and $v_2 \in \{0,\ldots,V_2\}$ can be considered as a vertex. We connect vertex (v_1,v_2) to (v_1',v_2') iff $v_1' = v_1+1$ or $v_2' = v_2'+1$. The whole graph is a grid, and we wish to find a subsequence of a path from $(0,0)$ to (V_1,V_2) so that the cost is minimized. The cost is defined by $C_{(v_1,v_2),(v_1',v_2)} = w_{1,v_1'-v_1}$ and $C_{(v_1,v_2),(v_1,v_2')} = w_{2,v_2'-v_2}$, where $w_{i,j}$ is the weight of a coin of value j from currency #i.

Pretty Printing with alternative expressions (LWSP version of Pretty Printing)

The Pretty Printing problem is to break a paragraph into lines, so that each line have roughly the same length. If a line is too long or too short, then there is some cost depending on the line length. The goal of the problem is to minimize the cost.

For some text, it is hard to print prettily. For example, if there are long formulas in the text, then sometimes its line gets too wide, but if we move the formula into the next line, the original line has too few words. One solution for this issue is to use alternate wording for the sentence, to rephrase a part of a sentence to its synonym. These sentences have different lengths, and formulas in some of them will be displayed better than others. These different ways can be considered as different paths in a graph, and we wish to find one sentence that has the minimal Pretty Printing cost.

A $\textsc{Path}_P$ instance

Say we have a set of words, and we want to find a word chain (a chain of words so that the last letter of the previous word is the same as the first letter of the next word) so that the first word and the last word satisfy some properties, e.g. they do not have similar meanings, they have the same length, they don't have the same letters on the same positions, etc. Each word corresponds to a vertex in the graph. For words that can be consecutive in a word chain, we add an edge to the words.