Completeness for First-Order Properties on Sparse Structures with Algorithmic Applications

Jiawei Gao * † Russell Impagliazzo * † Antonina Kolokolova ‡ § Ryan Williams † ‡

Abstract

Properties definable in first-order logic are algorithmically interesting for both theoretical and pragmatic reasons. Many of the most studied algorithmic problems, such as Hitting Set and Orthogonal Vectors, are first-order, and the first-order properties naturally arise as relational database queries. A relatively straightforward deterministic algorithm for evaluating a property with \( k \) quantifiers takes time \( O(m^k) \) and, assuming the Strong Exponential Time Hypothesis (SETH), some such properties require \( O(m^{k+\epsilon}) \) time for any \( \epsilon > 0 \). (Here, \( m \) represents the size of the input structure, i.e. the number of tuples in all relations.)

We give randomized algorithms for every first-order property that improves this upper bound to \( m^k O^{\sqrt{\log m}} \), 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 randomized fine-grained reductions. (To obtain improved algorithms, we apply the improved Orthogonal Vectors algorithm of [32].) 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 \( \mathbf{P} \) and beyond, this is the first such completeness result for a standard complexity class.

1 Introduction

Fine-grained complexity 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 ([20, 18]), studies of the exact complexity of \( \mathbf{NP} \)-complete problems ([29, 23, 24, 22]), and algorithmic consequences of circuit lower bounds ([6, 37, 26, 28, 8, 21, 25]), the full power of this approach has emerged only recently. This approach has given us new circuit lower bounds ([32, 34]), surprising algorithmic improvements using circuit lower bound techniques ([3, 31, 11, 15]), and many new insights into the relative difficulty of substantially improving known algorithms for a variety of problems both within and beyond polynomial time.

One of the strengths of this approach also makes it seemingly more complicated. Fine-grained reductions often cut across traditional complexity hierarchies; for example, many results use a now-standard reduction from the \( \mathbf{NP} \)-complete SAT problem down to the first-order definable (aka, uniform \( \mathbf{AC}^0 \)) orthogonal vectors problem. (Counter-intuitively, this reduces a very hard problem to a problem in an extremely simple complexity class.) On the other hand, different complete problems for the same complexity class can have different time complexities, meaning there may not be fine-grained reductions between them (or at least, that such reductions can be highly non-trivial.) Thus far, fine-grained complexity has remained focused on specific problems, rather than organizing problems into classes as in traditional complexity. 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 ([34, 35, 9, 11, 12, 11, 13, 27]).

Here, we give the first results in fine-grained complexity that apply to an entire complexity class, namely the class of first-order definable properties a.k.a. the uniform version of \( \mathbf{AC}^0 \). This class is both algorithmically natural in that it contains many standard problems considered before (such as Hitting Set and Orthogonal Vectors), and motivated by its importance in logic and database theory. For a first-order property with \( k + 1 \) quantifiers, on a structure with \( m \) records (also called tuples or hyperedges), the relatively straightforward deterministic
algorithm takes $O(m^k)$ time, and if Strong Exponential Time Hypothesis (SETH) is true, there are such properties that require $m^{k-o(1)}$ time to decide. For $k = 1$, this is linear time and so cannot be improved. For each such problem with $k \geq 2$, we give a probabilistic algorithm that solves it in $m^k/2^{O(\sqrt{\log m})}$ time. (This 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 of) the Orthogonal Vectors problem is complete under fine-grained reduction for the class of all first-order properties. This is the first completeness result for a previously studied complexity class under fine-grained reducibility. To obtain the algorithmic results, we then apply the counter-intuitive algorithm for the Orthogonal Vectors problem of [3], which uses techniques from circuit lower bounds. (This algorithm was derandomized by [10].)

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 [9], Longest Common Subsequence [5 11 13], Dynamic Time Warping [11 13], Fréchet Distance [12], Succinct Stable Matching [27], 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 ([11]), 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 1. (See 2.3 for definitions of problems.)

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. First-order properties are also extensively studied in complexity, logic (especially finite model theory and theory of databases) and combinatorics. Roughly speaking, first-order properties are essentially the uniform version of $AC^0$ in the complexity literature [10]. Algorithms for model-checking first-order properties are essential in databases (the core of the relational database language SQL is equivalent to first-order properties).

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 have hard instances that are already sparse. Secondly, the complexity of first-order problems in the dense model is somewhat unclear, at least for numbers of quantifiers between 3 and 7 ([33]). Third, the sparse model is more relevant for first-order model checking, as the input to database problems is given as a list of tuples.

1.1 First-order properties The problem of deciding whether a structure satisfies a logical formula is called the model checking problem. 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 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 [39]. Moreover, these problems are also major topics in parameterized complexity theory. [11] organizes parameterized first-order model checking problems (many of which are graph problems) into hierarchical classes based on their quantifier structures. Our work will study generic model checking problems in a fine-grained manner.
More specifically, let \( \varphi \) be a fixed first-order sentence containing free predicates of arbitrary constant arity (and no other free variables). For example, the Sparse \( k \)-OV problem can be expressed by \( \varphi = (\exists v_1 \in A_1) \ldots (\exists v_k \in A_k) (\forall i) \left[ \bigvee_{j=1}^n \neg (v_j[i] = 1) \right] \). The model-checking problem for \( \varphi \), denoted \( MC_\varphi \), is deciding whether \( \varphi \) is true on a given input structure interpreting predicates in \( \varphi \) (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 use notation \( MC(\Phi) \) for a class of model-checking problems for \( \varphi \in \Phi \), with the main focus on classes of \( (k+1) \)-quantifier \( \varphi \) with \( k \geq 1 \) (denoted \( MC(k+1) \)) and restrictions of this class to specific quantifier prefixes (e.g., \( MC(\exists \forall) \) for 3-quantifier \( \varphi \) with quantifier prefix \( \exists \forall \) when written in prenex normal form). For a formal description of the first-order properties and more examples, see Sections 2.2 and 2.3.

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 \( \forall \epsilon > 0 \), there is a \( (k+1) \)-quantifier first-order property that cannot be decided by any algorithm in \( O(m^{k-\epsilon}) \) time.

### 1.2 Orthogonal Vectors

In the **Orthogonal Vectors (OV)** problem, we are 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 \( O(n^2d) \), and the best known algorithm runs in \( n^{\omega(1)} \) time [3,10].

In this paper 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 Conjectures, based on the size of \( d \). In all three conjectures the complexity is measured in the word RAM model with \( O(\log n) \) bit words.

**Low-dimension OV (LDOVC):** \( \forall \epsilon > 0 \), there is no \( O(n^{2-\epsilon}) \) time algorithm for OV with dimension \( d = \omega(\log n) \).

**Moderate-dimension OV (MDOVC):** \( \forall \epsilon > 0 \), there is no \( O(n^{2-\epsilon} \cdot \text{poly}(d)) \) time algorithm that
solves OV with dimension $d$. \footnote{Although dimension $d$ is not restricted, we call it “moderate dimension” because such an algorithm only improves on the naive algorithm if $d = n^{O(\epsilon)}$.}

**Sparse OVC (SOVC):** For any integer $k \geq 2$: given a set $A_1, \ldots, A_k$ 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 $i$, $\prod_{j=1}^k v_i[j] = 0$ (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.

### 1.3 Main Results

**Completeness results.** 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 Corollary 5.1 to Lemma 5.2 in Section 5.1)

**Lemma 1.1.** For any integer $k \geq 2$, there exist $\delta, \epsilon > 0$ and an $O(n^{k-\epsilon})$ time algorithm solving $k$-OV with dimension $d = n^\delta$, if and only if there is an $\epsilon' > 0$ and an $O(m^{k-\epsilon'})$ time algorithm solving Sparse $k$-OV with $m$ being 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 1.1.** The following two propositions are equivalent:

(A) There exist $\delta, \epsilon > 0$ so that OV of dimension $d = n^\delta$ can be solved in time $O(n^{2-\epsilon})$. (i.e., MDOVC is false)

(B) For any integer $k \geq 2$, for any first-order property $L$ expressible with $k + 1$ quantifiers, there exists $\epsilon > 0$ so that $L$ can be decided in time $O(m^{k-\epsilon'})$ (i.e., FOPC is false).

This paper 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 $MC(\exists \forall^{k-1} \forall)$ (and its negation form $MC(\exists^k \forall \exists)$) under fine-grained reductions.

**Theorem 1.2.** If Sparse $k$-OV with total Hamming weight $m$ can be solved in randomized time $O(m^{k-\epsilon})$ for some $\epsilon > 0$, then all problems in $MC(\exists^k \forall)$, $MC(\forall^k \exists)$, $MC(\exists \forall^{k-1} \forall)$ and $MC(\exists^k \forall \exists)$ are solvable in randomized time $O(m^{k-\epsilon'})$ for some $\epsilon' > 0$.

$MC(\exists^k \forall)$ and $MC(\forall^k \exists)$ are interesting sub-classes of $MC(k + 1)$: if Nondeterministic SETH is true, then all SETH-hard problems in $MC(k + 1)$ are contained in $MC(\exists^k \forall)$ or $MC(\forall^k \exists)$ (\cite{13}).

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

**Algorithmic results.** Combining our reductions with the surprisingly fast algorithm for Orthogonal Vectors by \cite{3} and \cite{10}, we obtain improved algorithms for every problem representable as a $(k + 1)$-quantifier first-order property.

**Theorem 1.3.** There is an algorithm solving $MC(k + 1)$ 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 \cite{33}, 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 $MC(k + 1)$ on graphs when $k$ is a sufficiently large constant, and showed that $MC(k + 1)$ 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 $MC(k + 1)$ (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.

### 1.4 Organization of this paper

Section 1 introduced the motivation, some definitions and statements of the main results. Section 2 we give the formal definition of fine-grained reductions and exact complexity reductions, as well as detailed definitions of first-order properties including example first-order...
representations of common problems in fine-grained complexity. We present a general outline of the proofs in Section 3 and high-level explanation of key techniques in Section 4.

The full proof starts with the reduction from $MC(\exists \forall)$ to $k$-OV (Section 5.1), with randomized universe-shrinking self-reduction described in Section 5.2, which can be derandomized in Section 5.3. Then we present the extensions to hypergraphs in Section A and the reduction from $MC(\forall \exists \forall \forall)$ to $k$-OV in Section 7. In Section 9 we talk about open problems.

Section 6 shows the derandomization of the algorithm in Section 5. The last part of the appendix presents algorithms for $MC(k+1)$. In particular, Section B.1 gives a baseline algorithm for $MC(k+1)$ with time complexity $O(n^{k-1}m)$ and Section B.2 gives an improved algorithm with $m^k \frac{1}{2^k} O(\sqrt{\log m})$ time. Finally, Section B.3 contains algorithms for easy cases of $MC(k+1)$, based on analysis of 3-quantifier cases.

2 Preliminaries

2.1 Fine-grained reductions and exact complexity reductions To establish the relationship between complexities of different problems, we use the notion of fine-grained reductions as defined in [30]. These reductions establish conditional hardness results of the form “If one problem has substantially faster algorithms, so does another problem”. We will also use what we call exact complexity reductions (see definition 2.2), which strengthens the above claim to “if one problem has algorithms improved by a factor $s(m)$, then another problem can be improved by a factor $s'(m)$” 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.

DEFINITION 2.1. (Fine-grained reduction ($\leq_{FGR}$)) Assume that $L_1$ and $L_2$ are languages and $T_1$ and $T_2$ are their conjectured running time lower bounds, respectively. Then we say $(L_1, T_1) \leq_{FGR} (L_2, T_2)$ if for every $\epsilon > 0$, there exists $\epsilon' > 0$, and an algorithm $A_{L_1}$ for $L_1$ which runs in time $T_1(n)^{1-\epsilon} \cdot T_2(n_1)^{1-\epsilon'}$ on inputs of length $n$, making $q$ calls to an oracle for $L_2$ with query lengths $n_1, \ldots, n_q$, where $\sum_{i=1}^q T_2(n_i) \leq (T_1(n))^1 - \epsilon'$. That is, if $L_2$ has an algorithm substantially faster than $T_2$, $L_1$ can be solved substantially faster than $T_1$.

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 2.2. (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 $q$ calls to oracle of $L_2$ with query lengths $n_1, \ldots, n_q$, where $\sum_{i=1}^q T_2(n_i) \leq T_1(n)$. That is, if $L_2$ is solvable in time $T_2(n)$, then $A_{L_1}$ solves $L_1$ in time $T_1(n)$.

2.2 Model checking for first-order logic 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 $(v_1, \ldots, v_{\ell_1})$ such that $R_i(v_1, \ldots, v_{\ell_i})$ holds hyperedges (labeled $R_i$). A set of all $R_i$-labeled hyperedges in a given hypergraph is denoted by $E_{R}$, 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 $\varphi$ 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 $\varphi$.
Without changing $k$, we can write $\varphi$ in prenex form. The model-checking problem for a first-order property $\varphi$, $MC_\varphi$, is: given a structure (hypergraph) $G$, determine whether $\varphi$ holds on $G$ (denoted by $G \models \varphi$). For a class of formulas $\Phi$, we use the notation $MC(\Phi)$ for a class of model-checking problems for $\varphi \in \Phi$, with shortcuts $MC(k)$ for $\Phi = k$-quantifier first-order formulas in prenex form, and $MC(Q_1 \ldots Q_k)$ for first-order prenex formulas with quantifier prefix $Q_1 \ldots Q_k$, with a shortcut $Q^c_k$ denoting $c$ consecutive occurrences of $Q$ (e.g., $MC(3^k \forall)$).

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 $\varphi$, 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. $R(x_2, x_1)$ vs. $R(x_4, x_1)$) as different predicates, and partition corresponding edge sets appropriately.

The focus of this paper is on sparse structures, that is, the case when $m \leq O(n^{1+\gamma})$ for some $\gamma$ such that $0 \leq \gamma < 1$. In particular, each $E_i$ 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 $m$.

### 2.3 Common problems and conjectures

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):** For every $\epsilon > 0$, there exists a $k \geq 2$ so that $k$-CNF-SAT is not in $\text{TIME}[2^{n(1-\epsilon)}]$.

Below we list some problems studied in fine-grained complexity, with their first-order definitions. The problems are grouped by the type of input.

- **Graph problems.**
  1. Diameter-2: $(\forall x_1)(\forall x_2)(\exists x_3) [E(x_1, x_3) \land E(x_3, x_2)]$.
  2. Radius-2: $(\exists x_1)(\forall x_2)(\exists x_3) [E(x_1, x_3) \land E(x_3, x_2)]$.
  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.

- **Set problems.** The inputs are set families $S_1, \ldots, S_k$ over a universe $U$. We use “$\in$” as a binary predicate:
  1. Hitting Set, where all the sets are given explicitly in a set family $S$: $(\exists H \in S) (\forall S \in S) (\exists x) [(x \in H) \land (x \in S)]$. (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.)
  2. $k$-Set Packing, where all the sets are given explicitly in a set family $S$: $(\exists S_1 \in S) \ldots (\exists S_k \in S) (\forall x) \left[ \bigvee_{i=1}^k \left( (x \in S_i) \rightarrow \bigwedge_{j \neq i} (x \notin S_j) \right) \right]$.
  3. $k$-Empty Intersection (equivalent to $k$-OV): $(\exists S_1 \in S_1) \ldots (\exists S_k \in S_k) (\forall u \in U) \left[ \bigvee_{i=1}^k \neg(u \in S_i) \right]$.
  4. $k$-Set Cover: $(\exists S_1 \in S_1) \ldots (\exists S_k \in S_k) (\forall u \in U) \left[ \bigvee_{i=1}^k (u \in S_i) \right]$.
  5. Set Containment (equivalent to Sperner Family): $(\exists S_1 \subseteq S_1)(\exists S_2 \subseteq S_2)(\forall u \in U) \left[ [(\neg(u \in S_1)) \lor (u \in S_2)] \right]$.

See Section 1.2 for conjectures about variants of the Orthogonal Vectors problem.

### 3 Overview

The main technical part of this paper is in the proof of Theorem 1.2 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 randomized 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 $n^k$ and proving Lemma 1.1. Converting an $MC(∃∃∀)$ 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, $MC(∀∃∃)$ is reduced to $MC(∃∃∀)$.

In Theorem 1.1 and Theorem 1.3, we consider the class of all $k+1$-quantifier first-order properties $MC(k+1)$, 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 $∃∀∀$, with $2$-OV and other basic problems (to be defined in Section 5.1), is the hardest ($∃∀∀$ and their complements are easy, and the rest reduce to $∃∃∀$). Appealing to lemmas in the proof of Theorem 1.2 with $k=2$ completes the proof of Theorem 1.1 (see figure 2 for details), and applying the OV algorithm in [8] gives Theorem 1.3.

3.1 Reduction from $MC(k+1)$ to OV The following outlines the reduction from any arbitrary problem in $MC(k+1)$ to OV for any integer $k \geq 2$, thus proving the direction from (A) to (B) in Theorem 1.1. The direction from (B) to (A) is straightforward, because sparse OV is in $MC(3)$.

1. Using the quantifier-eliminating downward reduction of Lemma B.1, we reduce from the $(k+1)$-quantifier problem $MC_ϕ$ down to a $3$-quantifier problem $MC_ϕ′$. Thus, improving the $O(m^2)$ algorithm for $MC_ϕ$ implies improving the $O(m^k)$ algorithm for $MC_ϕ′$.

2. If $MC_ϕ′$ belongs to one of these classes: $MC(∃∃∃), MC(∀∀∀), MC(∀∃∃), MC(∃∀∀)$, we solve it directly in time $O(m^{3/2})$, using the algorithms from Lemma 3.2.

3. If $ϕ′$ has the quantifier structure $∀∃∀$ (or its negated form $∃∀∃$), we reduce $MC_ϕ′$ to $MC_ϕ″$, where $ϕ″$ has quantifier structure $∃∀∀$, using Lemma 7.1. If not, $ϕ′$ is already of the form $∀∃∀$.

4. We reduce a general model checking problem for $ϕ″$ of the quantifier structure $∃∀∀$ to a graph property problem of the same quantifier structure, using Lemma A.1.

5. Using Lemma 5.5, we reduce formulas of form $∃∀∀$ to a “hybrid” problem, which by Lemma 5.4 can be reduced to a combination of Sparse OV Set Containment and 2-Set Cover (which we call Basic Problems).

6. We use a probabilistic “universe-shrinking” technique (Lemma 5.2) on each of the Basic Problems, to transform a sparse instance into an equivalent one of small dimension.

7. After applying this to the hybrid problem, we can complement edge relations as needed to transform all Basic Problems into OV (Lemma 5.3).

8. By applying the [8] algorithm to the instance of low-dimension OV, we get an improved algorithm.

Figure 2 shows a diagram of the above reduction process.

Moreover, Lemmas 7.1, 5.5, 5.4 and 5.1 also work for any constant $k \geq 2$. So for a problem in $MC(∃k∀)$ or $MC(∀k−1∀)$, we can reduce it to $k$-OV as follows:

1. If the problem belongs to $MC(∀k−1∀)$, reduce it to $MC(∃k∀)$ using Lemma 7.1.

2. Eliminate hyperedges using Lemma A.1 then reduce the $MC(∃k∀)$ problem to the Hybrid problem, using Lemma 5.5.

3. Reduce from the Hybrid Problem to a combination of $2$ Basic Problems, using Lemma 5.4.

4. Reduce all Basic Problems to $k$-OV, using Lemma 5.1.

This completes the proof of Theorem 1.2.

4 The building blocks of hard instances

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 1.1, 1.2 and 1.3. Because of Lemma 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.

4.1 Complementing sparse relations The sparse $k$-OV problem can be reformulated as the $k$-Empty Intersection ($k$-EI) problem, where sets correspond to vectors and elements correspond to dimensions:

**Problem: $k$-Empty Intersection ($k$-EI) (Equivalent to $k$-OV.)**

**Input:** A universe $U$ of size $n_u$, and $k$ families of sets $S_1 \ldots S_k$ on $U$, of size $n_1 \ldots n_k$.

**Output:** Whether there exist $S_1 \in S_1, \ldots, S_k \in S_k$ such that $\bigcap_{i=1}^{k} S_i = \emptyset$.

**Logical expression:** $ϕ = (∀S_1 \in S_1) \ldots (∀S_k \in S_k)(∀u \in U) \left[ ∨_{i=1}^{k} \neg(u \in S_i)\right]$. We also call 2-EI the Set Disjointness problem.

Next, we introduce two similar problems that act
as important intermediate problems in our reduction process.

**Problem: Set Containment** (Equivalent \(^4\) to Sperner Family.)

**Input:** A universe \(U\) of size \(n_u\), and two families of sets \(S_1, S_2\) on \(U\), of size \(n_1, n_2\).

**Output:** Whether there exist \(S_1 \subseteq S_2\).

**Logical expression:** \(\varphi = (\exists S_1 \in S_1)(\exists S_2 \in S_2)(\forall u \in U)[(\neg(u \in S_1)) \lor (u \in S_2)]\).

**Problem:** \(k\)-Set Cover (Equivalent to \(k\)-Dominating Set.)

**Input:** A universe \(U\) of size \(n_u\), and \(k\) families of sets \(S_1, \ldots, S_k\) on \(U\), of size \(n_1, \ldots, n_k\).

**Output:** Whether there exist \(S_1 \subseteq S_2 \subseteq \ldots \subseteq S_k\) such that \(\bigcup_{i=1}^{k} S_i = U\).

**Logical expression:** \(\varphi = (\exists S_1 \in S_1) \ldots (\exists S_k \in S_k)(\forall u \in U)\left[\bigvee_{i=1}^{k} (u \in S_i)\right]\).

All these problems are first-order properties: we can use unary relations to partition the vertex set into \((S_1, \ldots, S_k, U)\), and consider the relation “\(\subset\)” as a binary relation. We will use the context of hypergraphs to describe the input structure, as in Appendix 2.2. 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. We call 2-Set Cover, Set Containment and OV (or equivalently, Set Disjointness) the Basic Problems; the general notion of a “Basic Problem” is formally defined in Section 5.1. Borassi et al. \(\square\) 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 \(S_1\), or \(S_2\), or both. It is easy to see that \(S_1 \cap S_2 = \emptyset \iff S_1^c \cup S_2^c = U \iff S_1 \subseteq S_2^c \iff S_2 \subseteq S_1^c\). 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 like \((u \in S_1)\), we say its complement, like \((u \notin S_1)\), 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 \(S_1\), 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 con-
struct truly subquadratic time reduction algorithms that preserve the improved factor in running time. Using this technique which we call \textit{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 \cite{11}. 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. Here, each set in the original instance is mapped to a new set defined on the smaller universe, and solutions of the original instance are mapped to solutions to the new instance with high probability given that all the sets are small. From this reduction, the claim follows:

\textbf{Claim 4.1.} If any one of OV, 2-Set Cover and Set Containment (or Sperner Family) has truly subquadratic time randomized algorithms, then the other two are also solvable in randomized subquadratic time. Thus the three problems are all hard for MC(3).

Claim 4.1 is itself an interesting result: in \cite{11}, 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 4.1 is a special case of Lemma 5.1 in Appendix 5, we give a much more general reduction. Claim 4.1 also shows that an improved algorithm for any of these three problems implies improved algorithms for the other two.

\subsection{Sparse and co-sparse relations}

Having shown how to reduce any two Basic Problems with the same $k$, 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 the paper.

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 \textit{query processing}. A set of such tuples (pairs) $(x_i, x_j)$ can be considered a result of a \textit{first-order query} defined by an intermediate formula $\varphi'$ 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, like $[R_1(x_1, x_2) \land \neg R_2(x_1, x_2)]$, we can iterate over them (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 $[\neg R_1(x_1, x_2) \land \neg R_2(x_1, x_2)]$, 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 either $R_1(x_1, x_2)$ 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 $[\neg R_1(x_1, x_2) \lor \neg R_2(x_1, x_2)]$. For a pair $(x_1, x_2)$, to make the formula true, predicates $R_1, R_2$ can be assigned values from \{\{True, False\}, \{False, True\}, \{False, False\}\}. In the first two cases, the pairs $(x_1, x_2)$ satisfying $[R_1(x_1, x_2) \land \neg R_2(x_1, x_2)]$ and $[\neg R_1(x_1, x_2) \land R_2(x_1, x_2)]$ are sparse, while in the last case, the pairs satisfying $[\neg R_1(x_1, x_2) \land \neg R_2(x_1, x_2)]$ are 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 Appendix 5.2 and for the baseline algorithm presented in Appendix 5.1.

Now, we would like to reduce $MC_\varphi$ 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 $\psi$ 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 $u(x, 0^r), \ldots, u(x, 1^r)$ for each
z, where $r$ is the number of different predicates in $\psi$, 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 $u_{(z,a)}$ if the assignment $a$ falsifies $\psi$ and the relations between $x$ (or $y$) and $z$ agree with $a$. In this way, a pair of 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 $\psi$. Then, if there exists a pair of disjoint sets $S_x$ and $S_y$, the corresponding $x$ and $y$ satisfy that for all $z$, $\psi$ 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. That is, we introduce a Hybrid Problem that is a combination of Basic Problems. Depending on the four combinations of sparsity or co-sparsity on the relations between variables $x, z, y, z$, we reduce $MC_\varphi$ not only to $OV$ but a combination of $OV, Set$ Containment, reversed Set Containment and 2-Set Cover in each of the 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 $MC_\varphi$.

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 5.5 for the proof that covers more complicated cases.

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

This section will prove the completeness of $k$-OV in $MC(\exists^k\forall)$ problems. Because we will deal with hypergraph inputs in Section A, here we only consider the input structures that are graphs, i.e. where all relations are either unary or binary. First, we introduce a class of Basic Problems, and prove 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).
corresponding set families. However, complementing the sets in $S_i$ takes time $O(n \cdot m_d)$, 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 $\delta$, if the size of $U'$ is $n_d' = O(m^\delta)$, then complementing all sets in $S_i$ only takes time and space $m \cdot O(m^\delta) = O(m^{1+\delta})$.

**Lemma 5.2.** (Universe-shrinking self-reductions of Basic Problems)

Let label $\ell$ 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 $O(m \cdot s(m)^\delta)$ on universe $U'$ whose size is $n'_u = O(s(m)^\delta)$, 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 OV to a moderate-dimension version of OV, implying Lemma 1.1 (see corollary below). The other direction (moderate-dimension OV to Sparse OV) is easy since if the dimension $d = n^\delta$, then maximal possible $m = d \cdot n = n^{1+\delta}$, as required.

**Corollary 5.1.** Reverse direction of Lemma 1.1

Suppose that for any $k \geq 2$ there exists $\delta, \epsilon > 0$ and an $O(n^{k-\epsilon})$ algorithm solving $k$-OV on dimension $d = n^\delta$. Then there is an $\epsilon' > 0$ and $O(m^{k-\epsilon'})$ time algorithm solving Sparse $k$-OV.

**Proof.** The algorithm converts an instance of Sparse $k$-OV to an instance of $k$-OV of dimension $n^\delta$ using universe-shrinking self-reduction (Lemma 5.2) and then applies assumed $O(n^{k-\epsilon})$ time algorithm to the reduced instance. More specifically, let $m = O(n^{1+\gamma})$, where $n$ is the number of vectors. Choosing $s(m) = O(m^{\delta/(1+\gamma)})$ for some $\delta > 0$ creates an instance of OV with dimension $n'_u = O(s(m)^\delta) = O(n^\delta)$, and size $m' = O(n^{1+\delta})$; number of vectors $n$ remains unchanged. Now, the reduction takes time $O(m^k/s(m)^\delta) = O(m^{k-\delta/(1+\gamma)})$, and running the $O(n^{k-\epsilon})$ time algorithm on the reduced instance takes $O(n^{k-\epsilon}) \leq O(m^{k-\epsilon/(1+\gamma)})$ time. Setting $\epsilon' = \min\{\delta/(1+\gamma), \epsilon/(1+\gamma)\}$ completes the proof.

We will present the randomizations for the above problems $BP[\ell]$ s.t. $\ell \neq 1^k$ in Section 5.1.1. For $BP[1^k]$, we will prove that it is either easy to solve or easy to complement in Section 5.1.2. In Appendix 6, we will derandomize these reductions.

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 5.3.** Reduction between different Basic Problems

For two different labels $\ell_1, \ell_2 \in \{0,1\}^k$, given set families $S_1, \ldots, S_k$, let $S'_1, \ldots, S'_k$ be defined such that

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

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

The proof of correctness is straightforward.

Pick $s'(m) = s(m)^{1/(6k)}$. Using Lemma 5.2 we shrink the universe to size $n'_u = s'(m)^\delta$. So the time complexity in this step is bounded by $O(m \cdot s'(m)^\delta)$, which is significantly less than $m^k/s(m)$ even if $k = 2$.

Let new instance size be $m'$. So $m' = m \cdot s'(m)^\delta$. Given that the constructed instance has an algorithm running in time $m^k/s(m')$, we get $m^k/s(m') < (m \cdot s(m)^{1/(6k)})^5/k/s(m) < m^k/s(m)^{1/6}$. Thus, by the two-step fine-grained mapping reductions given by Lemma 5.2 and Lemma 5.3 we have an exact complexity reduction between any two Basic Problems, completing the proof for Lemma 5.1.

When $k = 2$, Orthogonal Vectors ($BP[00]$), Set Containment ($BP[01]$ and $BP[10]$) and 2-Set Cover ($BP[11]$) are reducible to each other in subquadratic time. Thus Claim 4.1 follows.

### 5.1.1 Randomized universe-shrinking self-reduction of $BP[\ell]$ where $\ell \neq 1^k$

This section proves part of Lemma 5.2 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 $U$. (17 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 $U'$, so the error probability of the reduction algorithm is small.

**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 $S_1, \ldots, S_k$. If their sizes are pre-computed, we can do the enumeration in $O(m/s(m))$. Assume the current large set is $S_i \in S_i$. 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 $S_j$ after-interchanging), we create a new formula $\psi_{S_i}$ on variables $S_2, \ldots, S_k, u$ from formula $\psi$, by replacing $u \in S_1 (u \notin S_1)$ by a unary relation on $u$. Then, we decide if the graph induced by $S_2, \ldots, S_k$ 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 $h : U \to U^n$ 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 $S_i$, we compute new sets $h(S_i)$ for all $S_i \in S_i$. Then, we decide whether the new sets satisfy the following property, which is another $BP[\ell]$ problem:

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

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 illustrates an example of the universe-shrinking self-reduction for problem $BP[01]$, where we look for $S_1, S_2$ so that $S_1 \subseteq S_2$. If they exist, then after 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 problem $BP[00]$ where we decide whether there exist $S_i$ and $S_j$ so that they are disjoint, a false negative may occur when there are two disjoint sets but some elements in $S_i \cap S_j$ 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 $\ell$ containing $i$ ($i \geq 1$) zeros and $k - i$ ones, we can assume $BP[\ell]$ is defined by

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

or equivalently,

$$(\exists S_1) \ldots (\exists S_k)(\forall u)(\bigvee_{j=1}^{k-1} (u \notin S_j)) \vee (\bigvee_{j=i+1}^{k} (u \in S_j))$$

Figure 3: The universe-shrinking process. $S_i = \{a, b\}$ and $S_j = \{a', b', c, d\}$. After the mapping $h$, the new sets are $h(S_i) = \{a', b', c', d'\}$ and $h(S_j) = \{a', b', c', d', e'\}$.
"∀(S_1, ..., S_k), A' ∉ B'" is even smaller.

5.1.2 Deterministic universe-shrinking self-reduction of BP[1]^k

This section proves the remaining part of Lemma 5.2 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 nu is small, the sets are easy to complement; when nu is large, the problem is easy to solve.

- **Case 1:** Large universe. If nu > s(m), then in a solution of this problem, at least one set has size at least nu/k. There are at most m/(k/nu) = 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 5.1.1, 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)) · O(m^{k-1}) = O(m^k/s(m)).

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

5.2 Hybrid Problem

Next we reduce general MC(∃^k∀) 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(∃^k∀) and OV.

Let k ≥ 2. The input to the Hybrid Problem includes four parts:

1. Set families S_1, ..., S_k defined on universe U, where U is partitioned into 2^k disjoint sub-universes: U = ∪_{r ∈ [0,1]} U_r.
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 ∈ \{1, ..., k\} and i < j.

The goal of the problem is to decide if there exist S_i ∈ S_1, ..., S_k ∈ S_k such that both of the following constraints are true:

- **(A)** For each \( l \in \{0,1\}^k \), (S_1, ..., S_k) is a solution of BP[l] defined on sub-universe U_r.
- **(B)** For all pairs of indices i, j \( \in \{1, ..., k\} \), i < j, we have that R(S_i, S_j) = true iff type[i, j] = 1.

We let n be the sum of |S_1|, ..., |S_k| and U, and let m be the number of 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 ≤ n^{1+o(1)}). Figure shows a solution to a Hybrid Problem instance when k = 2.

Intuition behind the Hybrid Problem. We mentioned in Section 5 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 ≤ i ≤ k can be either sparse or co-sparse. Because there are k of such variables \(x_i\), there are \(2^k\) cases for a combination ((x_1, x_{k+1}), ..., (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 ≤ i < j ≤ 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 we expect a co-sparse relation.

5.2.1 Reduction to Basic Problems

**Lemma 5.4.** Let s(m) be a non-decreasing function such that 2^{Ω(√log m)} ≤ s(m) < m^{1/5}. Then, (Hybrid Problem, m^k/(s(m))^{1/6}) ≤ EC ((k-OF, 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 ≤ i < j ≤ 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 ∈ S_i, S_j ∈ 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 ∈ S_i and S_j ∈ S_j, we add an extra element u_{S_i, S_j} in U_ℓ and let u_{S_i, S_j} ∈ S_i, u_{S_i, S_j} ∈ S_j. Thus, S_i' ∈ S_i and S_j' ∈ S_j can both appear in the solution only when for all u_{S_i, S_j} (u_{S_i, S_j} \notin S_i') \lor (u_{S_i, S_j} \notin S_j'), and it holds iff R(S_i', S_j') = 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 edge R(S_i, S_j), we let u_{S_i} ∈ U_ℓ and let u_{S_i} ∈ S_i. For each edge R(S_i, S_j), we let u_{S_j} ∈ S_j. Thus, S_i' ∈ S_i and S_j' ∈ S_j can both appear in the solution only when for all u_{S_i, S_j}, (u_{S_i, S_j} \notin S_i') \lor (u_{S_j, S_j} \notin S_j'), and it holds iff
5.2 Hybrid Problem

After the above construction, we can drop the constraint (B) of the Hybrid Problem. We will ignore the relation $R$ and $\text{type}$ in the Hybrid Problem. The problem now is to decide whether there exists a tuple $(S_1, \ldots, S_k)$ being a solution to all $2^k$ Basic Problems. Then we can use Lemma 5.1 to reduce all these Basic Problems to $BP[0^k]$. Let $U'_{i}$ 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 $U'_{i}$, 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.

5.2.2 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 5.4.

**Lemma 5.5.** For any integer $k \geq 2$, any problem in $MC(\exists^k \forall)$ is linear-time Turing reducible to the Hybrid Problem, namely, $(MC(\exists^k \forall), T(m)) \leq_{EC} (\text{Hybrid Problem}, T(O(m)))$.

Consider the problem $MC_\varphi$ 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 the original vertex set.

WLOG, we assume that for each binary predicate $R_i(x_i, x_j), i \leq j$. Let $P_{k+1}$ be the set of unary and binary predicates in $\psi$ that involve variable $x_{k+1}$, and let $P_{k+1}$ denote the set of the other predicates not including $x_{k+1}$. A partial interpretation $\alpha$ for $P_{k+1}$ is a binary string of length $|P_{k+1}|$, that encodes the truth values assigned to all predicates in $P_{k+1}$. For each $i$ s.t. $1 \leq i \leq |P_{k+1}|$, if the $i$-th predicate in $P_{k+1}$ is assigned to true, then we set the $i$-th bit of $\alpha$ to one, otherwise we set it to zero. For a tuple $(v_1, \ldots, v_k)$, we say it implies $\alpha$ (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 $P_{k+1}$ are the same as the values specified by $\alpha$.

For each $\alpha \in \{0, 1\}^{P_{k+1}}$, we create a distinct Hybrid Problem instance $H_\alpha$. If any of the Hybrid Problems accepts, we accept. Let $\psi|_\alpha(x_1, \ldots, x_{k+1})$ be $\psi$ after replacing all occurrences of predicates in $P_{k+1}$ by their corresponding truth values specified by $\alpha$. The following steps show how to create $H_\alpha$ from $\alpha$ 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 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 \((u_1, \ldots, u_{k+1})\) 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 \(S\). 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

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

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 the sub-universe labeled by \(C\) where the \(i\)-th bit of \(C\) is 1, which is 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 \((v_i, v_{k+1}) \not\models c_i = 0^{|P_i|}\).

**Step 2:** Construction of relation \(R\) and string type.

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

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

- **Case 1:** At least one predicate on \((x_i, x_j)\) in \(\alpha\) 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 \(i < j \leq k\), if \((v_i, v_j)\) agrees with \(\alpha\), then we create edge \(R(S_{v_i}, S_{v_j})\). Finally we make \(\text{type}[i, j] = 1\) in the Hybrid Problem \(H_{\alpha}\).

- **Case 2:** All predicates on \((x_i, x_j)\) in \(\alpha\) 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 \(i < j \leq k\), if \((v_i, v_j)\) does not agree with \(\alpha\), then we create edge \(R(S_{v_i}, S_{v_j})\). Finally we make \(\text{type}[i, j] = 0\) in the Hybrid Problem \(H_{\alpha}\).

The analysis of correctness will be left to the full version of this paper.

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 5.5 follows.

6 Derandomization

We derandomize the reduction in Section 5 for the \(k = 2\) case, so that the whole proof of Theorems 1.1 and 1.3 is deterministic. 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 [28].

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 \( HP \) to Hybrid Problem.

**Lemma 6.1.** For any \( 2^{Ω(\sqrt{\log n})} \leq s < m^{1/3} \), there is a deterministic universe-shrinking self-reduction for \( SC \) such that
\[
(\text{SC}(\sqrt{n}), \frac{m^2}{s}) \leq_{\text{EC}} (\text{SC}(\frac{s^2 \log^2 n}{\log^3 s}, \frac{m^2}{s^3 \log^3 s})).
\]

**Lemma 6.2.** For any \( 2^{Ω(\sqrt{\log n})} \leq s < m^{1/3} \), there is a deterministic universe-shrinking self-reduction for \( SD \) such that
\[
(\text{SD}(\sqrt{n}), \frac{m^2}{s}) \leq_{\text{EC}} (\text{SD}(\frac{s^2 \log^2 n}{\log^3 s}, \frac{m^2}{s^3 \log^3 s})).
\]

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

**Lemma 6.3.** For any \( 2^{Ω(\sqrt{\log n})} \leq s < m^{1/3} \), where \( m \) is the input size to the Hybrid Problem, there is a deterministic reduction algorithm such that
\[
(\text{HP}, \frac{m^2}{s}) \leq_{\text{EC}} (\text{SD}(\frac{s^2 \log^2 n}{\log^3 s}, \frac{m^2}{s^3 \log^3 s})).
\]

**6.1 Proof of Lemma 6.1** This section presents the derandomization of the universe-shrinking self-reduction in Sections 5.1.1 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 \( s^\ell < q \) and \( q^\ell > n \). By Bertrand’s postulate, we can find such a \( q \) in time \( O(s \log n / \log s) \).

First, we use the algorithm in Section 5.1.1 to decide if there is a solution containing a size 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 \( q^\ell \). 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 \( \ell \). The number of different polynomials is \( q^\ell \). 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 \( h(u) = \{ i, p_u(i) \mid i \in GF(q) \} \) of size \( q \). For set \( S \subseteq U \), define \( h(S) = \bigcup_{u \in S} h(u) \). Finally, \( S'_1 = \{ h(S) \mid S \subseteq S_1 \} \), and \( S'_2 \) is constructed similarly. Then we decide the \( SC(q^\ell) \) instance that takes \( S'_1 \) and \( S'_2 \) as input.

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

If \( S_1 \not\subseteq S_2 \) for all sets, we need to show that for each element \( u_1 \in S_1 \setminus S_2 \), \( |h(u_1) \cap h(S_2)| < q \). 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 \( \ell \), 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 \( S_1 \not\subseteq S_2 \) in the original instance if there exist \( h(S_1) \not\subseteq h(S_2) \) in the constructed instance.

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

**6.2 Proof of Lemma 6.2** This section presents the derandomization of the universe-shrinking self-reduction in Section 5.1.1 for the Basic Problem \( BP[00] \), i.e. when the corresponding Basic Problem is the Set Disjointness problem, which is equivalent to Sparse OV.

First, we use the algorithm in Section 5.1.1 to decide if there is a solution containing a size of size at least \( s \), which takes time \( O(m^2/s) \). So next we only consider sets of size smaller than \( s \).

Let \( \ell = \log n / \log s \), and let \( q \) be a prime above \( s^\ell \), thus \( q = O(s^{2\ell}) = O(s^2 \log n / \log s) \). So \( q^\ell \geq n \). By Bertrand’s postulate, we can find such a \( q \) in time \( O(s^2 \log n / \log s) \). 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 \( \log n \leq \log q^\ell = \ell \).

Let \( a \) be an element in group \( GF(q) \). For each element \( u \) in \( U \), we let hash function \( h_a(u) = p_a(u) \). 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 sets. The algorithm makes \( q \) queries to \( SD(q) \) instances of input size \( m \), each taking time \( T(m) = m^2/s^3 \log^2 n = m^2/sq \), 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 element \( a \) in \( GF(q) \) so that \( p_a(u) = p_a(v) \) is at most \( \log n \), the degree of the polynomial. Suppose \( S_1 \in S_1 \) and \( S_2 \in S_2 \) 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 $S_1 \in S_1$ and $S_2 \in S_2$, $h(S_1 \cap S_2) \subseteq h(S_1) \cap h(S_2)$, 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.

### 6.3 Hybrid Problem

In this section we combine the above two deterministic reductions so solve the Hybrid Problem, which yields a deterministic reduction for Theorem 1.1 and Theorem 1.3. Here we use a similar version of Hybrid Problem as defined in Section 5.2 but without the relation $R$ and the string $type$. More formally, we consider the Hybrid Problem defined as follows:

**Problem HP**

**Input:** $S_1, S_2$, each a set family of sets $S_1 = A_1 \cup B_1 \cup C_1 \cup D_1$, $S_2 = A_2 \cup B_2 \cup C_2 \cup D_2$.

**Output:** Whether there exist $S_1 \in S_1$ and $S_2 \in S_2$ so that

1. $A_1 \cap A_2 = \emptyset$ (Set Disjointness)
2. $B_1 \subseteq B_2$ (Set Containment)
3. $C_1 \supseteq C_2$ (Set Containment reversed)
4. $D_1 \cup D_2 = U_D$ (2-Set Cover)

From the results in Section 5.2, the model checking for first-order sentences of form $\exists \forall$ can be reduced to the Hybrid Problem. More precisely, $(MC(\exists \forall), T(O(m))) \leq (HP, T(m))$.

**Proof of Lemma 6.3**

First, we decide if there is a solution containing a size of at least $s$, as described in the previous sub-sections, using time $O(m^2/s)$. So we next 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 $U_D \setminus D_i$ and $U_D \setminus D_j$. So $D_i \cup D_j = U_D$ iff $(U_D \setminus D_i) \cap (U_D \setminus D_j) = \emptyset$. The resulting instance size is $O(ms)$.

Then, we use the universe-shrinking self reduction algorithms 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 $O(s^{2 \log^2 n / \log^2 s})$. For each $j$, we create set $U_B \setminus B_j'$, so $B_i \subseteq B_j'$ if $B_i' \subseteq B_j' \land (U_B \setminus B_j') = \emptyset$. Similarly for each $i$ we create $U_C \setminus C_i'$, so $C_i \supseteq C_j'$ if $C_i' \supseteq C_j' \land (U_C \setminus C_i') \cap C_j' = \emptyset$. The resulting instance size is $O(m \cdot s^{2 \log^2 n / \log^2 s})$.

Finally, we use the universe-shrinking self reduction algorithms 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 $O(s^{2 \log^2 n / \log^2 s})$. For each $i$ and each $j$, we create sets $S_i' = A_i' \cup B_i' \cup (U_C \setminus C_i') \cup (U_D \setminus D_i)$ and $S_j' = A_j' \cup B_j' \cup C_j' \cup (U_D \setminus D_j)$.

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 $O(m \cdot s^{2 \log^2 n / \log^2 s})$.

In the reduction we make $s^{2 \log^2 n / \log^2 s}$ calls to the algorithm for Set Disjointness on small universe. Thus if $SD(O(s^{2 \log^2 n / \log^2 s}))$ has algorithms in time $m^2/s^{2 \log^2 n / \log^2 s}$, we get running time $s^{2 \log^2 n / \log^2 s} \cdot O((m \cdot s^{2 \log^2 n / \log^2 s})^2 / s^{7 \log^2 n / \log^2 s}) = O(m^2/s)$. \hfill $\Box$

This gives a reduction from the graph hitting problem to the Hybrid Problem.

### 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 $MC(\exists \forall)$ 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 \cite{7} to sparse structures.

**Lemma 7.1.** For $k \geq 2$ and $s(m)$ a non-decreasing function such that $2^{d((\log m)/s(m))} \leq s(m) < m^{1/5}$, let $\varphi' = (\exists x_2)\ldots(\exists x_{k+1})(\forall x_1)(\psi(x_1, \ldots, x_{k+1})$. There is an exact complexity reduction

$$(MC(\forall x_1)\varphi', \frac{m^k}{s^{(\sqrt m)})}) \leq EC(MC(\exists x_1)\varphi', \frac{m^k}{s(m)})$$

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 $V_2, \ldots, V_{k+1}$. 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 $V_1 \ldots V_k$ 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 $\psi_{v_1}$ on variables $x_2, \ldots, x_{k+1}$ from formula $\psi$, by replacing occurrences of unary
predicates in $\psi$ 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 $(\exists x_2) \cdots (\exists x_k)(\forall x_{k+1}) \psi_0(x_2, \ldots, x_{k+1})$ 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 $v_i \in V_i$ with degree less than $t$ in the outermost loop. For each such $v_i$, 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.

  o **Case 1**: $k > 2$. Because variables $x_2$ through $x_k$ are all quantified by $3$, we interchange their order so that the variable $x_1$ becomes the second-outmost variable $x_3$ (and thus the current $v_1$ becomes $v_2$). Next, for each $v_1$ and $v_2$ we construct a new formula $\psi(v_1, v_2)$ on variables $x_1, \ldots, x_{k+1}$, by regarding $x_1$ and $x_2$ as fixed values $v_1$ and $v_2$, and then modify $\psi$ 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 $(\exists x_3) \cdots (\exists x_{k+1}) \psi(v_1, v_2)(x_3, \ldots, x_{k+1})$, using time $O(m^{k-2})$. If the formula is satisfied, we mark the current $v_1$ as “good”. The total time complexity is $O(m \cdot m^{1/k}) \cdot O(m^{k-2}) = O(m^{k-1+1/k})$.

  o **Case 2**: $k = 2$. For each vertex $v_2$, we mark all the $v_1$’s satisfying $\forall x_3 \psi(x_1, x_2, x_3)$ 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 proceed 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 $\varphi'$. 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 $O(m^k/\log(m'))$, then the running time for $MC(\exists x_1)\varphi'$ is $O(m \log m) \cdot O(m^k/\log(m')) = O((m^{k-1}/k)^k/s(m') \cdot \log m) \leq O(m^{k}/s(\sqrt{m}) \cdot \log m)$. The exponent of $m$ is less than $k$. Thus this is a fine-grained Turing reduction. Lemma 7.1 follows.

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

### 8 Improved algorithms

In this section we present an algorithm solving Sparse OV in time $m^2/2^{\Theta(\sqrt{\log m})}$. The algorithm is adapted from the following result of dense OV $[3, 16]$. For vectors of dimension $d$, there is an algorithm solving Orthogonal Vectors in $n^2\cdot\Theta(d/\log d/\log n)$.

Consider the universe-shrinking self-reduction for Sparse OV (aka $BP[00]$), deciding if there are two sets that are disjoint) in Section 5.1. 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 $s(m)^{\varphi'}$ (in which case $k = 2$). Using the algorithm in the above theorem, we can solve it in time $n^2 - \Theta(1/\log(s(m)^{\varphi'})) < m^2 - \Theta(1/\log(s(m)^{\varphi'})) < m^2/2^{\Theta(\sqrt{\log m})}$. So the total running time is bounded by $m^2/2^{\Theta(\sqrt{\log m})}$.

By the above argument and Section 6 all the Basic Problems have algorithms in $m^2/2^{\Theta(\sqrt{\log m})}$, so does any other problem in $MC(\exists x \forall y)$. The reduction from $MC(\forall x \exists y)$ to $MC(\forall y)$ in Section 7 gives $2^{\Theta(\sqrt{\log m})}$ savings for $MC(\forall y)$ problems. Finally using the quantifier-eliminating downward reduction from Section 5.1, we get Theorem 1.3 that states all $MC(k+1)$ problems can be solved in $m^k/2^{\Theta(\sqrt{\log m})}$.
time by randomized algorithms.

9 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?

Our results raise the possibility that many other classes have complete problems under fine-grained reducibility, and that this will be a general method for establishing the plausibility of conjectures on the fine-grained complexity of problems. There is a number of candidates for such classes. We could drop the restriction that the formula has $k$ quantifiers in all, and look at formulas with quantifier depth $\leq k$. We could also stratify the first-order formulas by variable complexity, the number of distinct variable names in a formula, rather than number of quantifiers. (Variable complexity arises naturally in database theory, because the variable complexity determines the arity of some relation in any way of expressing the query as a sequence of sub-queries.) First-order logic is rather limited, so we could look at augmentations that increase its reach, such as allowing a total ordering on elements, or allowing the logic to take transitive closures of relations (e.g., to talk about the reachability relation in a sparse directed graph), or more generally, introduce monotone fixed point operations. Alternatively, rather than varying the types of formulas we could restrict the types of structures, for example considering structures of bounded treewidth.

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.

Acknowledgments

First of all, we 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, Ramya Ramyaa and Stefan Schneider for many useful discussions.

References

Appendix A Extending algorithms and hardness results to hypergraphs

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

**Lemma A.1.** If $MC(\exists \forall)$ on graphs is solvable in time $T(m)$, then $MC(\exists \forall)$ 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.

Define \(N(x,y)\) be a new relation such that \(N(x,y) = true\) iff there exists some \(z\) such that there is a hyperedge \(R_i(x,y,z) = true\) (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 \((\exists x)(\exists y)(\forall z)[N(x,y) \land \psi(x,y,z)] \lor (\exists x)(\exists y)(\forall z)[\neg N(x,y) \land \psi^*(x,y,z)]\). We can decide \((\exists x)(\exists y)(\forall z)[\neg N(x,y) \land \psi^*(x,y,z)]\) using the algorithm for graphs, because all relations are binary. To decide \((\exists x)(\exists y)(\forall z)[N(x,y) \land \psi(x,y,z)]\), we consider three types of \(x\)'s and \(y\)'s.

- **Type 1:** \(deg(x) \geq \sqrt{m}\). It is similar to deciding “large sets” for Basic Problems in Section 5.1.1. In the outer loop, enumerate all such \(x\)'s. For each \(x\), we modify the model checking problem to an instance of MC(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 MC(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 \(\psi\). 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 \((\forall z)[\neg N(x,y) \land \psi^*(x,y,z)]\).

Similarly, for \(MC(3)\) problems where \(\varphi = (\exists x_1)\ldots(\exists x_k)(\forall x_{k+1})[N(x_1,x_j) \land \psi(x_1,\ldots,x_{k+1})]\), 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

\[
\varphi = (\exists x_1)\ldots(\exists x_k)(\forall x_{k+1}) \left[ \bigvee_{i,j \in \{1,\ldots,k\}} (N(x_i,x_j) \land \psi(x_1,\ldots,x_{k+1}) \right] \lor \left[ \left( \bigwedge_{i,j \in \{1,\ldots,k\}} \neg N(x_i,x_j) \right) \land \psi^*(x_1,\ldots,x_{k+1}) \right]
\]

To decide \((\exists x_1)\ldots(\exists x_k)(\forall x_{k+1})[N(x_i,x_j) \land \psi(x_1,\ldots,x_{k+1})]\), 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

\[(\exists x_1) \ldots (\exists x_k)(\forall x_{k+1})
\left(\bigwedge_{i,j \in \{1, \ldots, k\} \atop i \neq j} \neg N(x_i, x_j)\right) \land \psi^*(x_1, \ldots, x_{k+1})\]

, we can use the algorithm for MC(∃k∀) problems on graphs, because the new formula has only unary and binary relations.

**Appendix B Baseline and improved algorithms**

In this section, we first present a baseline algorithm for MC(k + 1) that runs in time O(nk−1m), which also implicitly gives us a quantifier-eliminating downward reduction from any MC(k + 1) problem to MC(k) problems for k ≥ 2. Then, we show how to get an improved algorithm in time \(m^k / 2^{\Theta(\sqrt{k \log m})}\) using our reductions and the result by [3, 16]. 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.

**B.1 Baseline algorithm for first-order properties**

This section gives an O(nk−1m) time algorithm solving MC(k + 1) with any quantifier structure for k ≥ 1, thus proving Lemma [3, 16].

**LEMMA B.1.** (Quantifier-eliminating downward reduction for MC(k + 1))

*Let the running time of MC(k + 1) on graphs of n vertices and m edges be \(T_k(n, m)\). We have the recurrence*

\[
T_k(n, m) \leq n \cdot T_{k-1}(n, O(m)) + O(m), \quad \text{for } k \geq 2.
\]

\[
T_1(n, m) = O(m).
\]

By this lemma, if all problems in MC(k) have algorithms in time \(T(n, m)\), then any problem in MC(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 \(v_2\) 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 \(\tilde{\psi}(v_1, v_2)\) 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 \(\tilde{\psi}(v_1, v_2)\). Let

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

(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 \(\tilde{\psi}\) there are no occurrences of negated binary predicates, we can compute

\[
#\tilde{\psi}(v_1) = \text{The number of } v_2\text{ s.t. } \tilde{\psi}(v_1, v_2)\text{ holds}
\]

by first enumerating all tuples \((v_1, v_2)\) connected by at least one edge and checking if \(\tilde{\psi}(v_1, v_2)\) 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 \(\tilde{\psi}(v_1, v_2)\) true. Finally, let \(\#(v_1) = \#\psi(v_1) + \#\tilde{\psi}(v_1)\). 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, [\(\psi(v_1, v_2) = true\)] equals one and [\(\tilde{\psi}(v_1, v_2) = true\)] equals its contribution to \(\#\psi(v_1)\). 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 \(\#\tilde{\psi}(v_1)\) 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 \(v_2\). So when we enumerate all edges, [\(\psi(v_1, v_2) = true\)] equals zero and [\(\tilde{\psi}(v_1, v_2) = true\)] equals its contribution to \(\#\tilde{\psi}(v_1)\). 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 \(\#\psi(v_1)\) are both 0.

**Inductive Step.** For \(k \geq 2\), we give a quantifier-eliminating downward reduction, thus proving the recurrence relation. Assume \(\varphi = (Q_1x_1)\ldots(Q_kx_k+1)\psi(x_1, \ldots, x_{k+1})\) For each \(v_1 \in V_1\), create new formula \(\varphi_{v_1} = (Q_2x_2)\ldots(Q_k+1x_k+1)\tilde{\psi}(x_2, \ldots, x_{k+1})\), and in \(\psi\) 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 $R_i(x_1, x_2)$ (or $R_i(x_1, x_2)$) with unary predicate $R'_i(x_2)$ whose value equals $R_i(v_1, x_2)$ (or $R_i(v_1, x_2)$), 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 $\varphi_{v_1}$. If $x_1$ is quantified by $\exists$, we accept iff any of them accepts. Otherwise we accept iff all of them accepts. The construction of $\varphi_{v_1}$ 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 a $MC(k+1)$ problem to a $MC(k)$ problem. It makes $O(m)$ queries, each of size $O(m)$. Then if problems in $MC(k)$ are solvable in time $O(m^{k-1-\epsilon})$, then problems in $MC(k+1)$ are solvable in time $m \cdot O(m^{k-1-\epsilon}) = O(m^{k-\epsilon})$. This quantifier-eliminating downward reduction implies that if all $MC(k)$ have $T(n, m)$ time algorithms, then all $MC(k+1)$ 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 $MC(k+1)$ to $MC(3)$ in Lemma B.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.

### B.2 Improved algorithm for first-order properties

In this section we present an algorithm solving Sparse OV in time $m^2/2^{\Theta(\sqrt{\log m})}$. The algorithm is adapted from the following result of dense OV. [3].

**Theorem B.1.** ([5], [10]) For vectors of dimension $d$, there is an algorithm solving Orthogonal Vectors in $n^{2-\Omega(1/\log(d/\log n))}$.

Consider the universe-shrinking self-reduction for Sparse OV (aka BP[00], deciding if there are two sets that are disjoint) in Section 5.1. 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 $s(m)^{\frac{2}{3}}$ (in which case $k = 2$). Using the algorithm in the above theorem, we can solve it in time $n^{2-\Theta(1/\log(s(m)^{\frac{2}{3}}))} \leq m^2/2^{\Theta(\sqrt{\log m})} = m^2/2^{\Theta(\sqrt{\log m})}$. So the total running time is bounded by $m^2/2^{\Theta(\sqrt{\log m})}$.

By the above argument and Section 5, all the Basic Problems have algorithms in $m^2/2^{\Theta(\sqrt{\log m})}$, so does any other problem in $MC(\exists\forall)$. The reduction from $MC(\exists\forall)$ to $MC(\exists\forall)$ in Section 5 gives $2^{\Theta(\sqrt{\log m})}$ savings for $MC(\exists\forall)$ problems. Finally using the quantifier-eliminating downward reduction from Section B.1 we get Theorem 1.3 that states all $MC(k+1)$ problems can be solved in $m^k/2^{\Theta(\sqrt{\log m})}$ time by randomized algorithms.

### B.3 Algorithms for easy cases

In this section we show that any $(k+1)$-quantifier problem with a quantifier sequence ending with $\exists$ or $\forall$ 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 $MC(3)$ problem. Then from the next subsections we see that these problems are solvable in $O(m^{3/2})$. [5] shows improved algorithms that run in time $O(m^{41/41})$ for detecting triangles and detecting induced paths of length 2, which are special cases of $MC(\exists\exists\exists)$.

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

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.

#### B.3.1 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 $MC(3)$ problems. Let the Atomic Problem labeled by $\ell$ be $MC(\exists x \in X)(\exists y \in Y)(\exists z \in Z) \psi(x, y, z)$, and referred to as $\Delta[\ell]$. It is defined on a tripartite graph on vertex sets $(X, Y, Z)$, whose edge sets are $E_{X Y}, E_{Y Z}, E_{X Z}$ defined on $(X, Y), (Y, Z), (X, Z)$ respectively. The graph is undirected, i.e., $E_{X Y}, E_{Y Z}$ and $E_{X Z}$ 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 $E_{X Y}, E_{Y Z}, E_{X Z}$ 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 $\psi_t$ for all Atomic Problems are defined in Table 1. For problem $MC_\varphi$ where $\varphi = (\exists x \in X)(\exists y \in Y)(\exists z \in Z) \psi(x, y, z)$, we write $\psi$ as a DNF, and split the terms. Then we decide if 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 \( E(x, y) = \bigwedge_R \) is a positive binary predicate in \( R(X, Y) \) and \( \bigwedge_R \) is a negative binary predicate in \( \overline{R}(X, Y) \), 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

\[
\#_\ell(x) = \left| \{(y, z) \mid x, y, z \text{ satisfy } \psi_t \} \right|.
\]

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 3 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 there are at most \( \sqrt{m} \) sets with degree at least \( \sqrt{m} \). So in all, the running time is \( O(m^{3/2}) \).

Algorithm B.1 \( \Delta[2^+] \)

\[
\text{for all } (x, y) \in E_{XY} \text{ do } \quad \triangleright \text{ Small degree } y \\
\text{if } \deg_Z(y) \leq \sqrt{m} \text{ then} \\
\text{for all } z \text{ s.t. } (y, z) \in E_{YZ} \text{ do} \\
\text{if } (x, z) \in E_{XZ} \text{ then} \\
\#_{2^+}(x) \leftarrow \#_{2^+}(x) + 1 \\
\text{end if} \\
\text{end for} \\
\text{end for} \\
\text{end if} \\
\text{end for}
\]

Algorithm B.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 \( O(m^{3/2}) \).

Algorithm B.2 \( \Delta[1^+] \)

\[
\text{for all } (x, y) \in E_{XY} \text{ do } \quad \triangleright \text{ Small degree } y \\
\text{if } \deg_Z(y) \leq \sqrt{m} \text{ then} \\
\text{for all } z \text{ s.t. } (y, z) \in E_{YZ} \text{ do} \\
\text{if } (x, z) \notin E_{XZ} \text{ then} \\
\#_{1^+}(x) \leftarrow \#_{1^+}(x) + 1 \\
\text{end if} \\
\text{end for} \\
\text{end if} \\
\text{end for}
\]

\[
\text{for all } (x, y) \in E_{XY} \text{ do } \quad \triangleright \text{ Large degree } y \\
\text{if } \deg_Z(y) \geq \sqrt{m} \text{ then} \\
\#_{1^+}(x) \leftarrow \#_{1^+}(x) + \deg_Z(y) \\
\text{end if} \\
\text{end for}
\]

\[
\text{for all } y \text{ s.t. } \deg_Z(y) > \sqrt{m} \text{ do} \\
\text{for all } (x, z) \in E_{XZ} \text{ do} \\
\text{if } (x, y) \in E_{XY} \text{ and } (y, z) \in E_{YZ} \text{ then} \\
\#_{1^+}(x) \leftarrow \#_{1^+}(x) - 1 \\
\text{end if} \\
\text{end for} \\
\text{end for} \\
\text{end for}
\]
if we just over-counted the pair \((y,z)\), then we exclude the pair by subtracting one.

```latex
\begin{align*}
\text{if } \#_{1+}(x) > 0 \text{ for some } x \in X \text{ then Accept} \\
\text{else Reject}
\end{align*}
```

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 \(\#_{0+}(x) \leftarrow |E_{YZ}| - (\#_{2+}(x) + \#_{1+}(x) + \#_{1+}'(x))\).

### B.3.3 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 \(\psi\) in DNF and split each term to a separate \(\exists \exists \exists\) problem. Then for each term \(\psi_t\), we decide if there exist \(x_1,x_2,x_3\) satisfying it.

Let \(\psi_{t1}\) be the part of the conjunction containing all ternary predicates in \(\psi_t\), and \(\psi_{t2}\) be the rest of term \(\psi_t\). Thus \(\psi_t = \psi_{t1} \land \psi_{t2}\).

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

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

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