The Parameterized Complexity of Learning Monadic Second-Order Logic

Steffen van Bergerem
Humboldt-Universität zu Berlin, Germany

Martin Grohe
RWTH Aachen University, Germany

Nina Runde
RWTH Aachen University, Germany

Abstract

Within the model-theoretic framework for supervised learning introduced by Grohe and Turán (TOCS 2004), we study the parameterized complexity of learning concepts definable in monadic second-order logic (MSO). We show that the problem of learning an MSO-definable concept from a training sequence of labeled examples is fixed-parameter tractable on graphs of bounded clique-width, and that it is hard for the parameterized complexity class para-NP on general graphs.

It turns out that an important distinction to be made is between 1-dimensional and higher-dimensional concepts, where the instances of a k-dimensional concept are k-tuples of vertices of a graph. For the higher-dimensional case, we give a learning algorithm that is fixed-parameter tractable in the size of the graph, but not in the size of the training sequence, and we give a hardness result showing that this is optimal. By comparison, in the 1-dimensional case, we obtain an algorithm that is fixed-parameter tractable in both.

2012 ACM Subject Classification Theory of computation → Logic; Theory of computation → Complexity theory and logic; Theory of computation → Fixed parameter tractability; Computing methodologies → Logical and relational learning; Computing methodologies → Supervised learning

Keywords and phrases monadic second-order definable concept learning, agnostic probably approximately correct learning, parameterized complexity, clique-width, fixed-parameter tractable, Boolean classification, supervised learning, monadic second-order logic

Digital Object Identifier 10.4230/LIPIcs.CSL.2025.8

Related Version *Full Version*: `https://arxiv.org/abs/2309.10489` [10]

Funding *Steffen van Bergerem*: This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – project number 431183758 (gefördert durch die Deutsche Forschungsgemeinschaft (DFG) – Projektnummer 431183758).
Martin Grohe: Funded by the European Union (ERC, SymSim, 101054974). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.
Nina Runde: This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – project number 453349072 (gefördert durch die Deutsche Forschungsgemeinschaft (DFG) – Projektnummer 453349072).

1 Introduction

We study abstract machine-learning problems in a logical framework with a declarative view on learning, where the (logical) specification of concepts is separated from the choice of specific machine-learning models and algorithms (such as neural networks). Here we are concerned with the computational complexity of learning problems in this logical learning framework, that is, the *descriptive complexity of learning* [8].

Specifically, we consider Boolean classification problems that can be specified in monadic second-order logic (MSO). The input elements for the classification task come from a set $\mathbb{X}$, the *instance space*. A *classifier* on $\mathbb{X}$ is a function $c\colon \mathbb{X} \to \{+,-\}$. Given a *training sequence* S of labeled examples $(x,\lambda) \in \mathbb{X} \times \{+,-\}$, we want to find a classifier, called a *hypothesis*, that explains the labels given in S and that can also be used to predict the labels of elements from $\mathbb{X}$ not given as examples. In the logical setting, the instance space $\mathbb{X}$ is a set of tuples from a (relational) structure, called the *background structure*, and classifiers are described by formulas of some logic, in our case MSO, using parameters from the background structure. This model-theoretic learning framework was introduced by Grohe and Turán [33] and further studied in [30, 32, 31, 7, 9, 11, 8].

We study these problems within the following well-known settings from computational learning theory. In the *consistent-learning* model, the examples are assumed to be generated using an unknown classifier, the *target concept*, from a known *concept class*. The task is to find a hypothesis that is consistent with the training sequence S, i.e. a function $h\colon \mathbb{X} \to \{+,-\}$ such that $h(x) = \lambda$ for all $(x,\lambda) \in S$. In Haussler's model of *agnostic probably approximately correct (PAC) learning* [35], a generalization of Valiant's *PAC learning* model [50], an (unknown) probability distribution $\mathscr{D}$ on $\mathbb{X} \times \{+,-\}$ is assumed, and training examples are drawn independently from this distribution. The goal is to find a hypothesis that generalizes well, i.e. one is interested in algorithms that return with high probability a hypothesis with a small expected error on new instances drawn from the same distribution. For more background on PAC learning, we refer to [37, 41, 46]. In both settings, we require our algorithms to return a hypothesis from a predefined *hypothesis class*.

Our Contributions

In this paper, we study the parameterized complexity of the consistent-learning problem MSO-Consistent-Learn and the PAC-learning problem MSO-PAC-Learn. In both problems, we are given a graph G (the background structure) and a sequence of labeled training examples of the form $(\bar{v},\lambda)$, where $\bar{v}$ is a k-tuple of vertices from G and $\lambda \in \{+,-\}$. The goal is to find a hypothesis of the form $h_{\varphi,\bar{w}}$ for an MSO formula $\varphi(\bar{x};\bar{y})$ and a tuple $\bar{w}$ with $h_{\varphi,\bar{w}}(\bar{v}) := +$ if $G \models \varphi(\bar{v};\bar{w})$ and $h_{\varphi,\bar{w}}(\bar{v}) := -$ otherwise. For MSO-Consistent-Learn, this hypothesis should be consistent with the given training examples. For MSO-PAC-Learn, the hypothesis should generalize well. We restrict the complexity of allowed hypotheses by giving a bound q on the quantifier rank of φ and a bound ℓ on the length of $\bar{w}$. Both q and ℓ as well as the dimension k of the problem, that is, the length of the tuples to classify, are part of the parameterization of the problems. A detailed description of MSO-Consistent-Learn is given in Section 3. The problem MSO-PAC-Learn is formally introduced in Section 5.

▶ **Example 1.1.** Assume we are given the graph G depicted in Figure 1, the training sequence $S = ((v_1,+),(v_3,+),(v_4,-),(v_5,-))$, and $k = 1$, $\ell = 1$, $q = 3$. Note that $k = 1$ indicates that the instances are vertices of the input graph G. Furthermore, $\ell = 1$ indicates that the specification may involve one vertex of the input graph as a parameter. Finally, $q = 3$ indicates that the formula specifying the hypothesis must have quantifier rank at most 3.

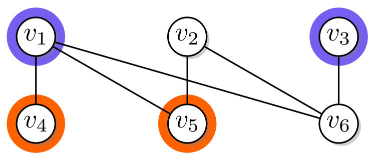

Figure 1 Graph G for Example 1.1. Positive examples are shown in purple, and negative examples are shown in orange.

Our choice of a hypothesis $h\colon V(G) \to \{+,-\}$ consistent with S says that "there is a bipartite partition of the graph such that all positive instances x are on the same side as v_2 and all negative examples are on the other side." This hypothesis can be formally specified in MSO as $h_{\varphi,\bar{w}}$ for the MSO formula $\varphi(x;y) = \exists Z\big(\psi_{\text{bipartite}}(Z) \wedge Z(x) \wedge Z(y)\big)$ and parameter setting $\bar{w} = (v_2)$, where $\psi_{\text{bipartite}}(Z) = \forall z_1 \forall z_2 \Big(E(z_1,z_2) \to \neg\big(Z(z_1) \leftrightarrow Z(z_2)\big)\Big)$.

For the 1-dimensional case of MSO-Consistent-Learn, called 1D-MSO-Consistent-Learn, [31, 30] gave algorithms that are sublinear in the background structures after a linear-time pre-processing stage for the case that the background structure is a string or a tree. This directly implies that 1D-MSO-Consistent-Learn can be solved in time $f(\ell, q) \cdot n$ for some function f, that is, in fixed-parameter linear time, if the background structure is a string or a tree. Here n is the size of the background structure and ℓ, q are the parameters of the learning problem described above. We generalize the results to labeled graphs of bounded clique-width. Graphs of clique-width c can be described by a c-*expression*, that is, an expression in a certain graph grammar that only uses c labels (see Section 2.1 for details). In our algorithmic results for graphs of bounded clique-width, we always assume that the graphs are given in the form of a c-expression. We treat c as just another parameter of our algorithms. By the results of Oum and Seymour [44], we can always compute a $2^{\mathcal{O}(c)}$-expression for a graph of clique-width c by a fixed-parameter tractable algorithm.

▶ **Theorem 1.2.** *Let $\mathscr{C}$ be a class of labeled graphs of bounded clique-width. Then* 1D-MSO-Consistent-Learn *is fixed-parameter linear on $\mathscr{C}$.*

Since graphs of bounded tree-width also have bounded clique-width, our result directly implies fixed-parameter linearity on graph classes of bounded tree-width.

Our proof for Theorem 1.2 relies on the model-checking techniques due to Courcelle, Makowsky, and Rotics for graph classes of bounded clique-width [23]. To make use of them, we encode the training examples into the graph as new labels. While this construction works for $k = 1$, it fails for higher dimensions if there are too many examples to encode.

As far as we are aware, all previous results for learning MSO formulas are restricted to the one-dimensional case of the problem. We give the first results for $k > 1$, presenting two different approaches that yield tractability results in higher dimensions.

As we discuss in Section 5, for the PAC-learning problem MSO-PAC-Learn in higher dimensions, we can restrict the number of examples to consider to a constant. In this way, we obtain fixed-parameter tractability results for learning MSO-definable concepts in higher dimensions, similar to results for first-order logic on nowhere dense classes [9, 8].

▶ **Theorem 1.3.** *Let $\mathscr{C}$ be a class of labeled graphs of bounded clique-width. Then* MSO-PAC-Learn *is fixed-parameter linear on $\mathscr{C}$.*

In the second approach to higher-dimensional tractability, and as the main result of this paper, we show in Section 6 that a consistent hypothesis can be learned on graphs of bounded clique-width with a quadratic running time in terms of the size of the graph.

▶ **Theorem 1.4.** *There is a function $g\colon \mathbb{N}^5 \to \mathbb{N}$ such that, for a Λ-labeled graph G of clique-width $\mathsf{cw}(G) \leq c$ and a training sequence S of size $|S| = m$, the problem* MSO-Consistent-Learn *can be solved in time*

$$\mathcal{O}\big((m+1)^{g(c,|\Lambda|,q,k,\ell)}|V(G)|^2\big).$$

While this is not strictly a fixed-parameter tractability result, since we usually do not consider m to be part of the parameterization, we show in Section 7 that this bound is optimal. Technically, this result is much more challenging than Theorems 1.3 and 1.2. While we still use an overall dynamic-programming strategy that involves computing MSO types, here we need to consider MSO types over sequences of tuples. The number of such sequence types is not constantly bounded, but exponential in the length of the sequence. The core of our argument is to prove that the number of relevant types can be polynomially bounded. This fundamentally distinguishes our approach from typical MSO/automata arguments, where types are from a bounded set (and they correspond to the states of a finite automaton).

Lastly, we study MSO-Consistent-Learn on arbitrary classes of labeled graphs. Analogously to the hardness of learning FO-definable concepts and the relation to the FO-model-checking problem discussed in [9], we are interested specifically in the relation of MSO-Consistent-Learn to the MSO-model-checking problem MSO-Mc. We show that MSO-Mc can already be reduced to the 1-dimensional case of MSO-Consistent-Learn, even with a training sequence of size two. This yields the following hardness result that we prove in Section 4.

▶ **Theorem 1.5.** 1D-MSO-Consistent-Learn *is para-NP-hard under fpt Turing reductions.*

Related Work

The model-theoretic learning framework studied in this paper was introduced in [33]. There, the authors give information-theoretic learnability results for hypothesis classes that can be defined using first-order and monadic second-order logic on restricted classes of structures.

Algorithmic aspects of the framework were first studied in [32], where it was proved that concepts definable in first-order logic can be learned in time polynomial in the degree of the background structure and the number of labeled examples the algorithm receives as input, independently of the size of the background structure. This was generalized to first-order logic with counting [7] and with weight aggregation [11]. On structures of polylogarithmic degree, the results yield learning algorithms running in time sublinear in the size of the background structure. It was shown in [31, 7] that sublinear-time learning is no longer possible if the degree is unrestricted. To address this issue, in [31], it was proposed to introduce a preprocessing phase where, before seeing any labeled examples, the background structure is converted to a data structure that supports sublinear-time learning later. This model was applied to monadic second-order logic on strings [31] and trees [30].

The parameterized complexity of learning first-order logic was first studied in [9]. Via a reduction from the model-checking problem, the authors show that on arbitrary relational structures, learning hypotheses definable in FO is AW[$*$]-hard. In contrast to this, they show that the problem is fixed-parameter tractable on nowhere dense graph classes. This result has been extended to nowhere dense structure classes in [8]. Although not stated as fpt results, the results in [31, 30] yield fixed-parameter tractability for learning MSO-definable concepts on strings and trees if the problem is restricted to the 1-dimensional case where the tuples to classify are single vertices.

The logical learning framework is related to, but different from the framework of *inductive logic programming* (see, e. g., [21, 42, 43]), which may be viewed as the classical logic-learning framework. In the database literature, there are various approaches to learning queries from examples [6, 5, 34, 36, 38, 13, 49, 1, 2, 14, 48, 17]. Many of these are concerned with active learning scenarios, whereas we are in a statistical learning setting. Moreover, most of the results are concerned with conjunctive queries or queries outside the relational database model, whereas we focus on monadic second-order logic. Another related subject in the database literature is the problem of learning schema mappings from examples [3, 15, 18, 19, 29]. In formal verification, related logical learning frameworks [20, 25, 28, 40, 52] have been studied as well. In algorithmic learning theory, related works study the parameterized complexity of several learning problems [4, 39] including, quite recently, learning propositional CNF and DNF formulas and learning solutions to graph problems in the PAC setting [16].

2 Preliminaries

We let $\mathbb{N}$ denote the set of non-negative integers. For $m, n \in \mathbb{N}$, we let $[m, n] := \{\ell \in \mathbb{N} \mid m \le \ell \le n\}$ and $[n] := [1, n]$. For a set V, we let $2^V := \{V' \mid V' \subseteq V\}$.

2.1 Clique-Width

In this paper, graphs are always undirected and simple (no loops or parallel edges); we view them as $\{E\}$-structures for a binary relation symbol E, and we denote the set of vertices of a graph G by $V(G)$. A *label set* is a set Λ of unary relation symbols, and a Λ-*graph* or Λ-*labeled graph* is the expansion of a graph to the vocabulary $\{E\} \cup \Lambda$. A *labeled graph* is a Λ-graph for any label set Λ.

In the following, we define *expressions* to represent labeled graphs. A *base graph* is a labeled graph of order 1. For every base graph G, we introduce a *base expression* β that *represents* G. Moreover, we have the following operations.

Disjoint union: For disjoint Λ-graphs G_1, G_2, we define $G_1 \uplus G_2$ to be the union of G_1 and G_2. If G_1 and G_2 are not disjoint, then $G_1 \uplus G_2$ is undefined.

Adding edges: For a Λ-graph G and unary relation symbols $P, Q \in \Lambda$ with $P \neq Q$, we let $\eta_{P,Q}(G)$ be the Λ-graph obtained from G by adding an edge between every pair of distinct vertices $v \in P(G)$, $w \in Q(G)$. That is, $E\big(\eta_{P,Q}(G)\big) := E(G) \cup \big\{(v, w), (w, v) \mid v \in P(G), w \in Q(G), v \neq w\big\}$.

Relabeling: For a Λ-graph G and unary relation symbols $P, Q \in \Lambda$ with $P \neq Q$, we let $\rho_{P,Q}(G)$ be the Λ-graph obtained from G by relabeling all vertices in P by Q, that is, $V(\rho_{P,Q}(G)) := V(G)$, $P(\rho_{P,Q}(G)) := \emptyset$, $Q(\rho_{P,Q}(G)) := Q(G) \cup P(G)$, and $R(\rho_{P,Q}(G)) := R(G)$ for all $R \in \Lambda \setminus \{P, Q\}$.

Deleting labels: For a Λ-graph G and a unary relation symbol $P \in \Lambda$, we let $\delta_P(G)$ be the restriction of G to $\Lambda \setminus \{P\}$, that is, the $(\Lambda \setminus \{P\})$-graph obtained from G by removing the relation $P(G)$.

We also introduce a modification of the disjoint-union operator, namely the *ordered-disjoint-union operator* $\uplus^<$, which is used in Section 6 to simplify notations.

Ordered disjoint union: To introduce this operator, we need two distinguished unary relation symbols $P_1^<$ and $P_2^<$. For disjoint Λ-graphs G_1, G_2, where we assume $P_1^<, P_2^< \notin \Lambda$, we let $G_1 \uplus^< G_2$ be the $(\Lambda \cup \{P_1^<, P_2^<\})$-expansion of the disjoint union $G_1 \uplus G_2$ with $P_1^<(G_1 \uplus^< G_2) := V(G_1)$ and $P_2^<(G_1 \uplus^< G_2) := V(G_2)$. By deleting the relations $P_1^<, P_2^<$ immediately after introducing them in an ordered disjoint union, we can simulate a standard disjoint-union by an ordered disjoint union, that is, $G_1 \uplus G_2 = \delta_{P_1^<}(\delta_{P_2^<}(G_1 \uplus^< G_2))$.

A Λ-*expression* is a term formed from base expressions β, whose label set is a subset of Λ, using unary operators $\eta_{P,Q}$, $\rho_{P,Q}$, δ_P for $P, Q \in \Lambda$ with $P \neq Q$, and the binary operator $\uplus$. We require Λ-expressions to be well-formed, that is, all base expressions represent base graphs with mutually distinct vertices, and the label sets fit the operators.

Every Λ-expression Ξ *describes* a Λ'-graph G_Ξ for some $\Lambda' \subseteq \Lambda$. Note that there is a one-to-one correspondence between the base expressions in Ξ and the vertices of G_Ξ. Actually, we may simply identify the vertices of G_Ξ with the base expressions in Ξ. We let $V_\Xi := V(G_\Xi)$ be the set of these base expressions. We may then view an expression Ξ as a tree where V_Ξ is the set of leaves of this tree. We let $|\Xi|$ be the number of nodes of the tree. We have $|G_\Xi| = |V_\Xi| \leq |\Xi|$. In general, we cannot bound $|\Xi|$ in terms of $|G_\Xi|$, but for every Λ-expression Ξ, we can find a Λ-expression Ξ' such that $G_{\Xi'} = G_\Xi$ and $|\Xi'| \in \mathcal{O}(|\Lambda^2| \cdot |G_\Xi|)$.

Each subexpression Ξ' of Ξ describes a labeled graph $G_{\Xi'}$ on a subset $V_{\Xi'} \subseteq V_\Xi$ consisting of all base expressions in Ξ'. Note that, in general, $G_{\Xi'}$ is not a subgraph of G_Ξ.

For $c \in \mathbb{N}$, a c-*expression* is a Λ-expression for a label set Λ of size $|\Lambda| = c$. It is easy to see that every labeled graph of order n is described by an n-expression. The *clique-width* $\mathsf{cw}(G)$ of a (labeled) graph G is the least c such that G is described by a c-expression.

We remark that our notion of clique-width differs slightly from the one given by Courcelle and Olariu [24], since we allow vertices to have multiple labels, and we also allow the deletion of labels. Thus, our definition is similar to the definition of *multi-clique-width* [27]. However, for our algorithmic results, the definitions are equivalent, since we have $\mathsf{cw}(G) \leq \mathsf{cw}'(G)$ and $\mathsf{cw}'(G) \in 2^{\mathcal{O}(\mathsf{cw}(G))}$ for every (labeled) graph G, where cw' is the notion of clique-width from [24].

▶ **Lemma 2.1** ([44]). *For a graph G with n vertices and clique-width $c' := \mathsf{cw}(G)$, there is an algorithm that outputs a c-expression for G where $c = 2^{3c'+2} - 1$. The algorithm has a running time of $\mathcal{O}(n^9 \log n)$.*

2.2 Monadic Second-Order Logic

We consider monadic second-order (MSO) logic, which is a fragment of second-order logic where we only quantify over unary relations (sets). In MSO, we consider two kinds of free variables, which we call set variables (uppercase X, Y, X_i) and individual variables (lowercase x, y, x_i). The *quantifier rank* $\mathrm{qr}(\varphi)$ of a formula φ is the nesting depth of its quantifiers.

Let τ be a relational vocabulary and $q \in \mathbb{N}$. By $\mathsf{MSO}(\tau, q)$, we denote the set of all MSO formulas of quantifier rank at most q using only relation symbols in τ, and we let $\mathsf{MSO}(\tau) := \bigcup_q \mathsf{MSO}(\tau, q)$. By $\mathsf{MSO}(\tau, q, k, s)$, we denote the set of all $\mathsf{MSO}(\tau, q)$ formulas with free individual variables in $\{x_1, \ldots, x_k\}$ and free set variables in $\{X_1, \ldots, X_s\}$. In particular, $\mathsf{MSO}(\tau, q, 0, 0)$ denotes the set of *sentences*. Moreover, it will be convenient to separate the free individual variables into *instance variables* $(x_1, x_2, \ldots)$ and *parameter variables* $(y_1, y_2, \ldots)$. For this, we let $\mathsf{MSO}(\tau, q, k, \ell, s)$ denote the set of all $\mathsf{MSO}(\tau, q)$ formulas with free instance variables in $\{x_1, \ldots, x_k\}$, free parameter variables in $\{y_1, \ldots, y_\ell\}$, and free set variables in $\{X_1, \ldots, X_s\}$. Furthermore, we write $\varphi(\bar{x}, \bar{y}, \bar{X})$ to denote that the formula φ has its free instance variables among the entries of $\bar{x}$, its free parameter variables among the entries $\bar{y}$, and its free set variables among the entries $\bar{X}$.

We normalize formulas such that the set of normalized formulas in $\mathsf{MSO}(\tau, q, k, \ell, s)$ is finite, and there is an algorithm that, given an arbitrary formula in $\mathsf{MSO}(\tau, q, k, \ell, s)$, decides if the formula is normalized, and if not, computes an equivalent normalized formula. In the following, we assume that all formulas are normalized.

In this paper, all structures we consider will be labeled graphs for some label set Λ. In notations such as $\mathsf{MSO}(\tau, \dots)$, it will be convenient to write $\mathsf{MSO}(\Lambda, \dots)$ if $\tau = \{E\} \cup \Lambda$. For a Λ-labeled graph G and a tuple $\bar{v} \in (V(G))^k$, the *q-type of $\bar{v}$ in G* is the set $\mathsf{tp}_q^G(\bar{v})$ of all formulas $\varphi(\bar{x}) \in \mathsf{MSO}(\Lambda, q, k, 0)$ such that $G \models \varphi(\bar{v})$.

2.3 VC Dimension

For $q, k, \ell \in \mathbb{N}$, a formula $\varphi(\bar{x}, \bar{y}) \in \mathsf{MSO}(\Lambda, q, k, \ell, 0)$, a Λ-labeled graph G, and a tuple $\bar{w} \in (V(G))^\ell$, we let

$$\varphi(G, \bar{w}) \coloneqq \left\{\bar{v} \in (V(G))^k \;\middle|\; G \models \varphi(\bar{v}, \bar{w})\right\}.$$

For a set $X \subseteq (V(G))^k$, we let

$$H_\varphi(G, X) \coloneqq \left\{X \cap \varphi(G, \bar{w}) \;\middle|\; \bar{w} \in (V(G))^\ell\right\}.$$

We say that X is *shattered* by φ if $H_\varphi(G, X) = 2^X$. The *VC dimension* $\mathrm{VC}(\varphi, G)$ of φ on G is the maximum $d \in \mathbb{N}$ such that there is a set $X \subseteq V(G)^k$ of cardinality $|X| = d$ that is shattered by φ. In this paper, we are only interested in finite graphs, but for infinite G, we let $\mathrm{VC}(\varphi, G) \coloneqq \infty$ if the maximum does not exist. For a class $\mathscr{C}$ of Λ-labeled graphs, the VC dimension of φ over $\mathscr{C}$, $\mathrm{VC}(\varphi, \mathscr{C})$, is the least d such that $\mathrm{VC}(\varphi, G) \leq d$ for all $G \in \mathscr{C}$ if such a d exists, and ∞ otherwise.

▶ **Lemma 2.2** ([33, Theorem 17]). *There is a function $g\colon \mathbb{N}^5 \to \mathbb{N}$ such that the following holds. Let Λ be a label set, let $\mathscr{C}$ be the class of all Λ-graphs of clique-width at most c, and let $q, k, \ell \in \mathbb{N}$. Then $\mathrm{VC}(\varphi, \mathscr{C}) \leq g(c, |\Lambda|, q, k, \ell)$ for all $\varphi \in \mathsf{MSO}(\Lambda, q, k, \ell, 0)$.*

2.4 Parameterized Complexity

A *parameterization* κ is a function mapping the input x of a problem to a natural number $\kappa(x) \in \mathbb{N}$. An algorithm $\mathbb{A}$ is an *fpt algorithm with respect to* κ if there is a computable function $f\colon \mathbb{N} \to \mathbb{N}$ and a polynomial p such that for every input x the running time of $\mathbb{A}$ is at most $f(\kappa(x)) \cdot p(|x|)$.

A *parameterized problem* is a tuple (Q, κ). We say $(Q, \kappa) \in \mathsf{FPT}$ or (Q, κ) is *fixed-parameter tractable* if there is an fpt algorithm with respect to κ for Q, and we say (Q, κ) is *fixed-parameter linear* if the polynomial in the running time of the fpt algorithm is linear. We say $(Q, \kappa) \in$ para-NP if there is a nondeterministic fpt algorithm with respect to κ for Q. If the parameterization is clear from the context, then we omit it.

For two parameterized problems $(Q, \kappa), (Q', \kappa')$, an *fpt Turing reduction* from (Q, κ) to (Q', κ') is an algorithm $\mathbb{A}$ with oracle access to Q' such that $\mathbb{A}$ decides Q, $\mathbb{A}$ is an fpt algorithm with respect to κ, and there is a computable function $g\colon \mathbb{N} \to \mathbb{N}$ such that on input x, $\kappa'(x') \leq g\big((\kappa(x)\big)$ for all oracle queries with oracle input x'.

For additional background on parameterized complexity, we refer to [26].

3 Tractability for One-Dimensional Training Data on Well-Behaved Classes

We start by formalizing the parameterized version of the problem MSO-Consistent-Learn described in the introduction. For a training sequence S, a graph G, and a hypothesis $h_{\varphi,\bar{w}}$, we say *$h_{\varphi,\bar{w}}$ is consistent with S on G* if for every positive example $(\bar{v}, +) \in S$, we have $G \models \varphi(\bar{v}, \bar{w})$, and for every negative example $(\bar{v}, -) \in S$, we have $G \not\models \varphi(\bar{v}, \bar{w})$.

MSO-Consistent-Learn

Instance: Λ-labeled graph G, $q, k, \ell \in \mathbb{N}$, training sequence $S \in (V(G)^k \times \{+,-\})^m$
Parameter: $\kappa := |\Lambda| + q + k + \ell$
Problem: Return a hypothesis $h_{\varphi,\bar{w}}$ consisting of

- a formula $\varphi \in \mathsf{MSO}(\Lambda, q, k, \ell, 0)$ and
- a parameter setting $\bar{w} \in V(G)^\ell$

such that $h_{\varphi,\bar{w}}$ is consistent with the training sequence S on G, if such a hypothesis exists. Reject if there is no consistent hypothesis.

The problem 1D-MSO-Consistent-Learn refers to the 1-dimensional version of the problem MSO-Consistent-Learn where the arity k of the training examples is 1. The tractability results for 1D-MSO-Consistent-Learn are significantly more straightforward than those for the higher-dimensional problem. This is due to the fact that the full training sequence can be encoded into the graph by only adding two new labels, and a parameter setting can be encoded with ℓ more new labels.

As discussed in Section 2.2, there is a function $f\colon \mathbb{N}^4 \to \mathbb{N}$ such that $|\mathsf{MSO}(\Lambda, q, k, \ell, 0)| \leq f(|\Lambda|, q, k, \ell)$. Therefore, to solve 1D-MSO-Consistent-Learn, we can iterate over all formulas $\varphi \in \mathsf{MSO}(\Lambda, q, k, \ell, 0)$ and focus on finding a parameter setting $\bar{w} \in V(G)^\ell$ such that $h_{\varphi,\bar{w}}$ is consistent with S on G. Moreover, if the model-checking problem on a graph class with additional labels is tractable, then finding a consistent parameter setting is tractable as well by performing model checking on the graph with the encoded training sequence.

▶ **Lemma 3.1.** *Let $\mathscr{C}$ be a class of labeled graphs, let $\mathscr{C}_i$ be the class of all extensions of graphs from $\mathscr{C}$ by i additional labels for all $i \in \mathbb{N}$, let $f\colon \mathbb{N} \to \mathbb{N}$ be a function, and let $c \in \mathbb{N}$ such that the* MSO*-model-checking problem on $\mathscr{C}_i$ can be solved in time $f(|\varphi|) \cdot |V(G)|^c$ for all $i \in \mathbb{N}$, where φ is the* MSO *sentence and $G \in \mathscr{C}_i$ is the labeled graph given as input. There is a function $g\colon \mathbb{N}^3 \to \mathbb{N}$ such that* 1D-MSO-Consistent-Learn *can be solved on $\mathscr{C}$ in time $g(|\Lambda|, q, \ell) \cdot |V(G)|^{c+1}$.*

The formal proof of this result can be found in the full version [10]. If the input graph is given in the form of a c-expression, then the MSO model-checking problem is fixed-parameter linear on classes of bounded clique-width [23]. Therefore, Lemma 3.1 implies that there is a function $g\colon \mathbb{N}^4 \to \mathbb{N}$ such that 1D-MSO-Consistent-Learn can be solved in time $g(\mathsf{cw}(G), |\Lambda|, q, \ell) \cdot |G|^2$.

Theorem 1.2 improves this bound for classes of graphs of bounded clique-width even further, showing that the problem 1D-MSO-Consistent-Learn can be solved in time linear in the size of the graph. This can be done by again encoding the training sequence into the graph, but then extracting a consistent parameter setting directly, following techniques similar to the ones used by Courcelle and Seese for the corresponding model-checking problem on graphs of bounded clique-width. The full proof of Theorem 1.2 can be found in [10].

Since graphs of tree-width c_t have a clique-width of at most $3 \cdot 2^{c_t - 1}$ [22], Theorem 1.2 implies that for classes of graphs of bounded tree-width, 1D-MSO-Consistent-Learn is fixed-parameter linear as well. Moreover, although all background structures we consider in this paper are labeled graphs, we remark that the result for classes of bounded tree-width also holds on arbitrary relational structures and a corresponding version of 1D-MSO-Consistent-Learn.

4 Hardness for One-Dimensional Training Data

Previously, we restricted the input graph of the MSO-learning problem to certain well-behaved classes. Now, we consider the problem MSO-Consistent-Learn without any restrictions. Van Bergerem, Grohe, and Ritzert showed in [9] that there is a close relation between first-order model checking (FO-Mc) and learning first-order formulas. The fpt-reduction in [9] from model checking to learning yields AW[$*$]-hardness for learning first-order formulas on classes of structures that are not nowhere dense. It is simple to show (and not surprising) that MSO-Consistent-Learn is at least as hard as FO-Mc. The more interesting question is whether MSO-Consistent-Learn is at least as hard as the model-checking problem for MSO sentences (MSO-Mc), which is defined as follows.

MSO-Mc

Instance: Λ-labeled graph G, MSO(Λ) sentence φ
Parameter: $|\varphi|$
Problem: Decide whether $G \models \varphi$ holds.

We give a positive answer, which even holds for the MSO-learning problem with only one-dimensional training data where we restrict the training sequence to contain at most two training examples.

▶ **Lemma 4.1.** *The model-checking problem* MSO-Mc *is fpt Turing reducible to* 1D-MSO-Consistent-Learn *where we restrict the training sequence S given as input to have length at most* 2.

MSO-Mc is para-NP-hard under fpt Turing reductions as even for some fixed sentence φ, the corresponding model-checking problem can be NP-hard (for example for a formula defining 3-Colorability, see [26] for details). Hence, Lemma 4.1 proves Theorem 1.5. We give a proof sketch for Lemma 4.1. The full proof can be found in [10].

Proof sketch of Lemma 4.1. We describe an fpt algorithm solving MSO-Mc using access to a 1D-MSO-Consistent-Learn oracle. Let G be a Λ-labeled graph, and let φ be an MSO(Λ) sentence. We decide whether $G \models \varphi$ holds recursively by decomposing the input formula. While handling negation and Boolean connectives is easy, the crucial part of the computation is handling quantification. Thus, we assume that $\varphi = \exists x \psi$ or $\varphi = \exists X \psi$ for some MSO formula ψ. For both types of quantifiers, we use the 1D-MSO-Consistent-Learn oracle to identify a small set of candidate vertices or sets such that ψ holds for any vertex or set if and only if it holds for any of the identified candidates. Then, since the number of candidates will only depend on $|\psi|$, we can check recursively whether ψ holds for any of them and thereby decide MSO-Mc with an fpt algorithm.

More specifically, using the 1D-MSO-Consistent-Learn oracle, we partition the vertices and sets based on their qr(ψ)-type, and we only check one candidate for each class of the partition. Intuitively, for every pair of vertices, we call the oracle with one vertex as a positive example and the other vertex as a negative example. The oracle returns a hypothesis if and only if the types of the two vertices differ. When partitioning the sets of vertices, we encode the two sets to check into the graph before calling the oracle. Moreover, instead of calling the oracle for every pair of sets (which would lead to a running time that is exponential in the size of the graph), we group the sets based on their size, start by finding a small family of candidate sets of size 1, and we use these candidates iteratively to find a small family of sets with one additional vertex. With this technique, the number of oracle calls is only quadratic in the size of the graph. ◀

5 PAC Learning in Higher Dimensions

So far, we considered the consistent-learning setting, where the goal is to return a hypothesis that is consistent with the given examples. In this section, we study the MSO-learning problem in the agnostic PAC-learning setting. There, for an instance space $\mathbb{X}$, we assume an (unknown) probability distribution $\mathscr{D}$ on $\mathbb{X} \times \{+,-\}$. The learner's goal is to find a hypothesis $h \colon \mathbb{X} \to \{+,-\}$, using an oracle to draw training examples randomly from $\mathscr{D}$, such that h (approximately) minimizes the generalization error

$$\mathrm{err}_{\mathscr{D}}(h) \coloneqq \Pr_{(x,\lambda)\sim\mathscr{D}}\big(h(x) \neq \lambda\big).$$

For every Λ-labeled graph G and $q,k,\ell \in \mathbb{N}$, let $\mathscr{H}_{q,k,\ell}(G)$ be the hypothesis class

$$\mathscr{H}_{q,k,\ell}(G) \coloneqq \big\{h_{\varphi,\bar{w}} \bigm| \varphi \in \mathsf{MSO}(\Lambda,q,k,\ell,0), \bar{w} \in (V(G))^{\ell}\big\}.$$

Formally, we define the MSO PAC-learning problem as follows.

MSO-PAC-Learn

Instance: Λ-labeled graph G, numbers $k,\ell,q \in \mathbb{N}$, $\delta,\varepsilon \in (0,1)$, oracle access to probability distribution $\mathscr{D}$ on $(V(G))^k \times \{+,-\}$
Parameter: $\kappa \coloneqq |\Lambda| + k + \ell + q + 1/\delta + 1/\varepsilon$
Problem: Return a hypothesis $h_{\varphi,\bar{w}} \in \mathscr{H}_{q,k,\ell}(G)$ such that, with probability of at least $1-\delta$ over the choice of examples drawn i.i.d. from $\mathscr{D}$, it holds that

$$\mathrm{err}_{\mathscr{D}}(h_{\varphi,\bar{w}}) \leq \min_{h\in\mathscr{H}_{q,k,\ell}(G)} \mathrm{err}_{\mathscr{D}}(h) + \varepsilon.$$

The remainder of this section is dedicated to the proof of Theorem 1.3, that is, we want to show that MSO-PAC-Learn is fixed-parameter linear on classes of bounded clique-width when the input graph is given as a c-expression. To solve the problem algorithmically, we can follow the *Empirical Risk Minimization (ERM)* rule [51, 46], that is, our algorithm should minimize the *training error* (or *empirical risk*)

$$\mathrm{err}_S(h) \coloneqq \frac{1}{|S|} \cdot |\{(\bar{v},\lambda) \in S \mid h(\bar{v}) \neq \lambda\}|$$

on the training sequence S of queried examples. Roughly speaking, an algorithm can solve MSO-PAC-Learn by querying a sufficient number of examples and then following the ERM rule. To bound the number of needed examples, we combine a fundamental result of statistical learning [12, 46], which bounds the number of needed examples in terms of the VC dimension of a hypothesis class, with Lemma 2.2, a result due to Grohe and Turán [33], which bounds the VC dimension of MSO-definable hypothesis classes on graphs of bounded clique-width. Together, they imply the following result. See the full version [10] for details.

▶ **Lemma 5.1.** *There is a computable function $m \colon \mathbb{N}^5 \times (0,1)^2 \to \mathbb{N}$ such that any algorithm that proceeds as follows solves the problem* MSO-PAC-Learn*. Given a Λ-labeled graph G of clique-width at most c, numbers $k,\ell,q \in \mathbb{N}$, $\delta,\varepsilon \in (0,1)$, and oracle access to a probability distribution $\mathscr{D}$ on $(V(G))^k \times \{+,-\}$, the algorithm queries at least $m(c,|\Lambda|,q,k,\ell,\delta,\varepsilon)$ many examples from $\mathscr{D}$ and then follows the ERM rule.*

Using this lemma, we can now give a proof sketch for Theorem 1.3, showing that MSO-PAC-Learn is fixed-parameter linear on classes of bounded clique-width if the input graph is given as a c-expression, even for dimensions $k > 1$. The full proof can be found in [10].

Proof sketch of Theorem 1.3. Let G be a Λ-labeled graph, let $k, \ell, q \in \mathbb{N}$, $\delta, \varepsilon \in (0,1)$, and assume we are given oracle access to a probability distribution $\mathcal{D}$ on $(V(G))^k \times \{+,-\}$. Moreover, let $c \in \mathbb{N}$ and let Ξ be a c-expression given as input that describes G.

Let $s := m(c, |\Lambda|, q, k, \ell, \delta, \varepsilon)$, where m is the function from Lemma 5.1. We sample s examples from $\mathcal{D}$ and call the resulting sequence of training examples S. Then, for every subsequence S' of S, we make use of the techniques in Section 3 (adapted to higher-dimensional training data) and compute a hypothesis $h_{S'} \in \mathcal{H}_{q,k,\ell}$ that is consistent with S' if such a hypothesis exists. This can be computed by an fpt-algorithm with parameters $c, |\Lambda|, k, \ell, q, \delta$, and ε. Finally, from all subsequences with a consistent hypothesis, we choose a subsequence S^* of maximum length and return h_{S^*}. Note that this procedure minimizes the training error on the training sequence S, i. e., $\mathrm{err}_S(h_{S^*}) = \min_{h \in \mathcal{H}_{q,k,\ell}} \mathrm{err}_S(h)$. Hence, the procedure follows the ERM rule, and, by Lemma 5.1, it solves MSO-PAC-Learn. Since the number of subsequences to check can be bounded by 2^s, all in all, the described procedure is an fpt-algorithm that solves MSO-PAC-Learn. ◀

6 Consistent Learning in Higher Dimensions

In this section, we consider the problem MSO-Consistent-Learn with dimension $k > 1$ on labeled graphs of bounded clique-width. A brute-force attempt yields a solution in time $g(c, |\Lambda|, q, k, \ell) \cdot (V(G))^{\ell+1} \cdot m$, where m is the length of the training sequence, for some $g\colon \mathbb{N}^5 \to \mathbb{N}$. This is achieved by iterating over all formulas, then iterating over all parameter assignments, and then performing model checking for each training example. We assume that the graph G is considerably larger in scale than the sequence of training examples S. Therefore, Theorem 1.4 significantly improves the running time to $\mathcal{O}\big((m+1)^{g(c,|\Lambda|,q,k,\ell)}|V(G)|^2\big)$. While Theorem 1.4 is not a fixed-parameter tractability result in the classical sense, we show that this is optimal in Section 7. The present section is dedicated to the proof of Theorem 1.4.

Until now, we have viewed the training sequence as a sequence of tuples $S \in ((V(G))^k \times \{+,-\})^m$. In the following, it is useful to split the training sequence into two parts, a sequence of vertex tuples $\mathscr{a} \in ((V(G))^k)^m$ and a function $\sigma\colon [m] \to \{+,-\}$ which assigns the corresponding label to each tuple. Let G be a Λ-labeled graph, $\mathscr{a} = (\bar{v}_1, \dots, \bar{v}_m) \in ((V(G))^k)^m$, $\sigma\colon [m] \to \{+,-\}$, and $\varphi(\bar{x}, \bar{y}) \in \mathsf{MSO}(\Lambda, q, k, \ell, 0)$. We call $(G, \mathscr{a}, \sigma)$ *φ-consistent* if there is a parameter setting $\bar{w} \in (V(G))^\ell$ such that for all $i \in [m]$, $G \models \varphi(\bar{v}_i, \bar{w}) \iff \sigma(\bar{v}_i) = +$. We say that $\bar{w}$ is a *φ-witness* for $(G, \mathscr{a}, \sigma)$. This notation allows us to state the main technical ingredient for the proof of Theorem 1.4 as follows.

▶ **Theorem 6.1.** *There is a computable function $g\colon \mathbb{N}^5 \to \mathbb{N}$ and an algorithm that, given a Λ-graph G of clique-width $\mathsf{cw}(G) \leq c$, a sequence $\mathscr{a} = (\bar{v}_1, \dots, \bar{v}_m) \in ((V(G))^k)^m$, a function $\sigma\colon [m] \to \{+,-\}$, and a formula $\varphi(\bar{x}, \bar{y}) \in \mathsf{MSO}(\Lambda, q, k, \ell, 0)$, decides if $(G, \mathscr{a}, \sigma)$ is φ-consistent in time $(m+1)^{g(c,|\Lambda|,q,k,\ell)}|G|$.*

Using Theorem 6.1, we can now prove Theorem 1.4.

Proof of Theorem 1.4. Given a Λ-labeled graph G and a training sequence $S = ((\bar{v}_1, \lambda_1), \dots, (\bar{v}_m, \lambda_m)) \in ((V(G))^k \times \{+,-\})^m$, we let $\mathscr{a} := (\bar{v}_1, \dots, \bar{v}_m) \in ((V(G))^k)^m$ and $\sigma\colon [m] \to \{+,-\}, i \mapsto \lambda_i$. We iterate over all formulas $\varphi \in \mathsf{MSO}(\Lambda, q, k, \ell, 0)$ and use Theorem 6.1 to check whether $(G, \mathscr{a}, \sigma)$ is φ-consistent. If there is no φ such that $(G, \mathscr{a}, \sigma)$ is φ-consistent, then we reject the input. Otherwise, let $\varphi \in \mathsf{MSO}(\Lambda, q, k, \ell, 0)$ be such that $(G, \mathscr{a}, \sigma)$ is φ-consistent. We compute a φ-witness following the same construction as in the proof of Lemma 3.1. That is, using a fresh label, we encode the parameter choice of a single variable into the graph, and then we check whether consistency still holds for the corresponding formula φ' that enforces this parameter choice. In total, we perform up to $\ell \cdot |V(G)|$ such consistency checks to compute a φ-witness $\bar{w}$. The consistent formula φ together with the φ-witness $\bar{w}$ can then be returned as a hypothesis $h_{\varphi,\bar{w}}$ that is consistent with S on G and therefore a solution to MSO-Consistent-Learn. ◀

The remainder of this section is dedicated to the proof of Theorem 6.1. We start by introducing the formal definitions for types and sets of types over sequences of elements.

6.1 Type Definitions

Let G be a Λ-labeled graph and $\bar{v} \in (V(G))^k$. Recall that the *q-type of $\bar{v}$ in G* is the set $\mathsf{tp}_q^G(\bar{v})$ of all formulas $\varphi(\bar{x}) \in \mathsf{MSO}(\Lambda, q, k, 0)$ such that $G \models \varphi(\bar{v})$. A *$(\Lambda, q, k)$-type* is a set $\theta \subseteq \mathsf{MSO}(\Lambda, q, k, 0)$ such that, for each $\varphi \in \mathsf{MSO}(\Lambda, q, k, 0)$, either $\varphi \in \theta$ or $\neg\varphi \in \theta$. We denote the set of all (Λ, q, k)-types by $\mathsf{Tp}(\Lambda, q, k)$. Note that $\mathsf{tp}_q^G(\bar{v}) \in \mathsf{Tp}(\Lambda, q, k)$. For a type $\theta \in \mathsf{Tp}(\Lambda, q, k)$, we write $G \models \theta(\bar{v})$ if $G \models \varphi(\bar{v})$ for all $\varphi(\bar{x}) \in \theta$. Observe that $G \models \theta(\bar{v}) \iff \mathsf{tp}_q^G(\bar{v}) = \theta$. We say that a type $\theta \in \mathsf{Tp}(\Lambda, q, k)$ is *realizable* if there is some Λ-labeled graph G and tuple $\bar{v} \in (V(G))^k$ such that $\theta = \mathsf{tp}_q^G(\bar{v})$. We are not particularly interested in types that are not realizable, but it is undecidable if a type θ is realizable, whereas the sets $\mathsf{Tp}(\Lambda, q, k)$ are decidable. (More precisely, there is an algorithm that, given Λ, q, k and a set θ of formulas, decides if $\theta \in \mathsf{Tp}(\Lambda, q, k)$.) For a (Λ, q, k)-type θ and a Λ-labeled graph G, we let

$$\theta(G) := \{\bar{v} \in (V(G))^k \mid G \models \theta(\bar{v})\}.$$

If $\theta(G) \neq \emptyset$, we say that θ is *realizable in G*.

As for formulas, we split the variables for types into two parts, so we consider (Λ, q, k, ℓ)-types $\theta \subseteq \mathsf{MSO}(\Lambda, q, k, \ell, 0)$, and we denote the set of all these types by $\mathsf{Tp}(\Lambda, q, k, \ell)$. For a Λ-labeled graph G and tuples $\bar{v} \in (V(G))^k, \bar{w} \in (V(G))^\ell$, we often think of $\mathsf{tp}_q^G(\bar{v}, \bar{w})$ as the *q-type of $\bar{w}$ over $\bar{v}$ in G*. Moreover, we let

$$\theta(\bar{v}, G) := \{\bar{w} \in (V(G))^\ell \mid G \models \theta(\bar{v}, \bar{w})\}.$$

If $\theta(\bar{v}, G) \neq \emptyset$, we say that θ is *realizable over $\bar{v}$ in G*.

For a vector $\bar{k} = (k_1, \ldots, k_m) \in \mathbb{N}^m$ and a set V, we let $V^{\bar{k}}$ be the set of all sequences $\mathscr{a} = (\bar{v}_1, \ldots, \bar{v}_m)$ of tuples $\bar{v}_i \in V^{k_i}$. Let G be a labeled graph, $\mathscr{a} = (\bar{v}_1, \ldots, \bar{v}_m) \in V^{\bar{k}}$ for some $\bar{k} \in \mathbb{N}^m$, and $\bar{w} \in (V(A))^\ell$. We define the *q-type of $\bar{w}$ over $\mathscr{a}$ in G* to be the tuple

$$\mathsf{tp}_q^G(\mathscr{a}, \bar{w}) := \big(\mathsf{tp}_q^G(\bar{v}_1, \bar{w}), \ldots, \mathsf{tp}_q^G(\bar{v}_m, \bar{w})\big).$$

Again, we need an "abstract" notion of type over a sequence. A *$(\Lambda, q, \bar{k}, \ell)$-type* for a tuple $\bar{k} = (k_1, \ldots, k_m) \in \mathbb{N}^m$ is an element of

$$\mathsf{Tp}(\Lambda, q, \bar{k}, \ell) := \prod_{i=1}^{m} \mathsf{Tp}(\Lambda, q, k_i, \ell).$$

Let $\bar{\theta} = (\theta_1, \ldots, \theta_m) \in \mathsf{Tp}(\Lambda, q, \bar{k}, \ell)$. For a labeled graph G, a sequence $\mathscr{a} = (\bar{v}_1, \ldots, \bar{v}_m) \in (V(G))^{\bar{k}}$, and a tuple $\bar{w} \in (V(G))^\ell$, we write $G \models \bar{\theta}(\mathscr{a}, \bar{w})$ if $G \models \theta_i(\bar{v}_i, \bar{w})$ for all $i \in [m]$. Note that $G \models \bar{\theta}(\mathscr{a}, \bar{w}) \iff \mathsf{tp}_q^G(\mathscr{a}, \bar{w}) = \bar{\theta}$. For a type $\bar{\theta} \in \mathsf{Tp}(\Lambda, q, \bar{k}, \ell)$, a Λ-labeled graph G, and a sequence $\mathscr{a} \in (V(G))^{\bar{k}}$, we let

$$\bar{\theta}(\mathscr{a}, G) := \{\bar{w} \in (V(G))^\ell \mid G \models \theta(\mathscr{a}, \bar{w})\}.$$

If $\bar{\theta}(\mathscr{a}, G) \neq \emptyset$, we say that $\bar{\theta}$ is *realizable over $\mathscr{a}$ in G*.

6.2 Computing the Realizable Types

For the proof of Theorem 6.1, we use the following result that allows us to compute the realizable types of an expression.

▶ **Lemma 6.2.** *There is a computable function $f\colon \mathbb{N}^4 \to \mathbb{N}$ and an algorithm that, given $c, q, k, \ell, m \in \mathbb{N}$, a vector $\bar{k} = (k_1, \dots, k_m) \in \mathbb{N}^m$ with $k_i \leq k$ for all $i \in [m]$, a Λ-expression Ξ with $|\Lambda| \leq c$, and a sequence $\mathcal{a} \in (V_\Xi)^{\bar{k}}$, computes the set of all $\bar{\theta} \in \mathsf{Tp}(\Lambda, q, \bar{k}, \ell)$ that are realizable over $\mathcal{a}$ in G_Ξ in time*

$$\mathcal{O}\Big((m+1)^{f(c,q,k,\ell)} \cdot |\Xi|\Big).$$

Before proving this result, we first show that it implies Theorem 6.1.

Proof of Theorem 6.1. We assume that the input graph G is given as a c-expression. To check whether $(G, \mathcal{a}, \sigma)$ is φ-consistent, we compute the set $\mathcal{R}$ of all $\bar{\theta} \in \mathsf{Tp}(\Lambda, q, \bar{k}, \ell)$ that are realizable over $\mathcal{a}$ in G, using Lemma 6.2. Then, for each $\bar{\theta} = (\theta_1, \dots, \theta_m) \in \mathcal{R}$ we check if $\varphi \in \theta_i \iff \sigma(i) = +$. If we find such a $\bar{\theta}$, then $(G, \mathcal{a}, \sigma)$ is φ-consistent; otherwise it is not. ◀

It remains to prove Lemma 6.2. In a bottom-up algorithm, starting at the leaves of Ξ, we compute the set of all tuples $\bar{\theta} = (\theta_1, \dots, \theta_m)$ of (Λ, q, k_i, ℓ)-types θ_i that are realizable over $\mathcal{a}$ in G. This is possible because the realizable tuples of types of an expression can be computed from the tuples of types of its subexpressions. We formally prove this for the operators $\eta_{P,Q}$, $\rho_{P,Q}$, δ_P, and $\uplus^<$ in the full version [10].

The difficulty with this approach is that we are talking about m-tuples of types. In general, the number of such tuples is exponential in m, and hence the size of the set we aim to compute could be exponentially large. Fortunately, this does not happen in graphs of bounded clique-width. By Lemma 2.2, we can bound the VC dimension of a first-order formula over classes of graphs of bounded clique-width. Further, we show in Lemma 6.3 that this suffices to give a polynomial bound for the number of realizable tuples.

▶ **Lemma 6.3.** *Let $d, q, k, \ell \in \mathbb{N}$, let $t := |\mathsf{Tp}(\Lambda, q, k, \ell)|$, and let G be a Λ-labeled graph such that $\mathrm{VC}(\varphi, G) \leq d$ for all $\varphi \in \mathsf{MSO}(\Lambda, q, k, \ell, 0)$. Let $\mathcal{a} \in (V(G))^{\bar{k}}$ for some $\bar{k} \in \{0, \dots, k\}^m$. Then at most $(k+1) \cdot g(d, m)^t$ types in $\mathsf{Tp}(\Lambda, q, \bar{k}, \ell)$ are realizable over $\mathcal{a}$ in G.*

The proof of Lemma 6.3 is based on the Sauer–Shelah Lemma [45, 47]. See [10] for proof details. Based on this, we can now prove Lemma 6.2.

Proof of Lemma 6.2. As argued in Section 2, we may assume that Ξ only contains ordered-disjoint-union operators and no plain disjoint-union operators.

For every subexpression Ξ', we let $\Lambda_{\Xi'}$ be the set of labels of Ξ', that is, the set of unary relation symbols such that $G_{\Xi'}$ is a $\Lambda_{\Xi'}$-graph. Moreover, let $\mathcal{a}_{\Xi'} := \mathcal{a} \cap V_{\Xi'}$, and let $\bar{k}_{\Xi'} \subseteq \mathbb{N}^m$ such that $\mathcal{a}_{\Xi'} \in (V_{\Xi'})^{\bar{k}_{\Xi'}}$.

We inductively construct, for every subexpression Ξ' of Ξ and $0 \leq \ell' \leq \ell$, the set $\mathcal{R}_{\ell'}(\Xi')$ of all types $\bar{\theta} \in \mathsf{Tp}(\Lambda_{\Xi'}, q, \bar{k}_{\Xi'}, \ell')$ that are realizable over $\mathcal{a}_{\Xi'}$ in $G_{\Xi'}$.

Case 1: Ξ' is a base expression. In this case, for each $\ell' \in [\ell]$, we can construct $\mathcal{R}_{\ell'}(\Xi')$ by brute force in time $f_1(c, q, k, \ell) \cdot m$ for a suitable (computable) function f_1. Let $(k'_1, \dots, k'_m) := \bar{k}_{\Xi'}$. We compute θ_i by iterating over all formulas φ with $k'_i + \ell'$ free variables and evaluating φ on the single vertex graph $G_{\Xi'}$.

Case 2: $\Xi' = \eta_{P,Q}(\Xi'')$. Let $0 \leq \ell' \leq \ell$, and let $\bar{k}' = (k'_1, \dots, k'_m) \coloneqq \bar{k}_{\Xi'} = \bar{k}_{\Xi''}$. As we show in the full version [10], there is a computable mapping $T_{\eta,P,Q} \colon \mathsf{Tp}(\Lambda_{\Xi'}) \to 2^{\mathsf{Tp}(\Lambda_{\Xi''})}$ such that $\mathscr{R}_{\ell'}(\Xi')$ is the set of all $\bar{\theta} = (\theta_1, \dots, \theta_m) \in \mathsf{Tp}(\Lambda_{\Xi'}, q, \bar{k}', \ell')$ such that there is a $\bar{\theta}' = (\theta'_1, \dots, \theta'_m) \in \mathscr{R}(\Xi'')$ with $\theta'_i \in T_{\eta,P,Q}(\theta_i)$ for all $i \in [m]$. Moreover, for every realizable $\theta' \in \mathsf{Tp}(\Lambda_{\Xi'})$, we guarantee that there is at most one type $\theta \in \mathsf{Tp}(\Lambda_{\Xi''})$ such that $\theta' \in T_{\eta,P,Q}(\theta)$. To compute the set $\mathscr{R}_{\ell'}(\Xi')$, we step through all $\bar{\theta}' \in \mathscr{R}(\Xi'')$. For each such $\bar{\theta}' = (\theta'_1, \dots, \theta'_m)$, for all $i \in [m]$, we compute the unique $\theta_i \in \mathsf{Tp}(\Lambda_{\Xi'}, q, k'_i, \ell')$ such that $\theta'_i \in T_{\eta,P,Q}(\theta_i)$. If for some $i \in [m]$, no such θ_i exists, we move on to the next $\bar{\theta}'$. Otherwise, we add $\bar{\theta} = (\theta_1, \dots, \theta_m)$ to $\mathscr{R}(\Xi')$.

Case 3: $\Xi' = \rho_{P,Q}(\Xi'')$. Analogous to Case 2, again based on results from the full version [10].

Case 4: $\Xi' = \delta_P(\Xi'')$. Analogous to Case 2, based on results from [10].

Case 5: $\Xi' = \Xi_1 \uplus^< \Xi_2$. Let $\Lambda' \coloneqq \Lambda_{\Xi'}$, $V' \coloneqq V_{\Xi'}$, $\bar{k}' = (k'_1, \dots, k'_m) \coloneqq \bar{k}_{\Xi'}$, and $\mathscr{a}' \coloneqq (\bar{v}'_1, \dots, \bar{v}'_m) \coloneqq \mathscr{a}_{\Xi'} = \mathscr{a} \cap V'$. For $j = 1, 2$, let $\Lambda_j \coloneqq \Lambda_{\Xi_j}$, $V_j \coloneqq V_{\Xi_j}$, $\bar{k}_j \coloneqq (k_{j1}, \dots, k_{jm}) \coloneqq \bar{k}_{\Xi_j}$, and for all $i \in [m]$, let $K_{ji} \subseteq [k_{ji}]$ such that $\bar{v}'_i \cap V_j = (\bar{v}_i)_{K_{ji}}$. Let $0 \leq \ell' \leq \ell$.

For all $L_1, L_2 \subseteq [\ell']$ such that $L_2 = [\ell'] \setminus L_1$, we let $\mathscr{R}_{L_1,L_2}$ be the set of all $\bar{\theta} = (\theta_1, \dots, \theta_m) \in \mathsf{Tp}(\Lambda', q, \bar{k}', \ell')$ such that for $j = 1, 2$, we have

$$\bar{\theta}_j \coloneqq \big(T_{\uplus,j,L_j,K_{j1}}(\theta_1), \dots, T_{\uplus,j,L_j,K_{jm}}(\theta_m)\big) \in \mathscr{R}_{|L_j|}(\Xi_j),$$

where $T_{\uplus,j,L_j,K_{ji}} \colon \mathsf{Tp}(\Lambda') \to \mathsf{Tp}(\Lambda_j)$ for $i \in [m]$ is a computable mapping that we give in the full version [10]. Then, as we also show in [10],

$$\mathscr{R}_{\ell'}(\Xi') = \bigcup_{\substack{L_1 \subseteq [\ell'] \\ L_2 = [\ell'] \setminus L_1}} \mathscr{R}_{L_1,L_2}.$$

To compute $\mathscr{R}_{L_1,L_2}$, we iterate over all $\bar{\theta}_1 = (\theta_{11}, \dots, \theta_{1m}) \in \mathscr{R}_{|L_1|}(\Xi_1)$. For all $i \in [m]$ we compute the unique $\theta_i \in \mathsf{Tp}(\Lambda', q, k'_i, \ell')$ such that $T_{\uplus,1,L_1,K_{1i}}(\theta_i) = \theta_{1i}$. If, for some $i \in [m]$, no such θ_i exists, then we move on to the next $\bar{\theta}_1$. Otherwise, we compute

$$\bar{\theta}_2 = \big(T_{\uplus,2,L_2,K_{21}}(\theta_1), \dots, T_{\uplus,2,L_2,K_{2m}}(\theta_m)\big)$$

and check if $\bar{\theta}_2 \in \mathscr{R}_{|L_2|}(\Xi_2)$. If it is, we add $\bar{\theta}$ to $\mathscr{R}_{L_1,L_2}$. Otherwise, we move on to the next $\bar{\theta}_1$.

This completes the description of our algorithm. To analyze the running time, let

$$r \coloneqq \max_{\Xi'} |\mathscr{R}_{\Xi'}|,$$

where Ξ' ranges over all subexpressions of Ξ. By Lemmas 2.2 and 6.3, there is a computable function $f_2 \colon \mathbb{N}^4 \to \mathbb{N}$ such that

$$r \leq (m+1)^{f_2(c,q,k,\ell)}.$$

The running time of each step of the constructions can be bounded by $f_3(c, q, k, \ell) \cdot r$ for a suitable computable function f_3. We need to make $|\Xi|$ steps. Thus, overall, we obtain the desired running time. ◀

7 Hardness of Checking Consistency in Higher Dimensions

The following result shows, under the assumption $\mathsf{FPT} \neq \mathsf{W}[1]$, that Theorem 6.1 can not be improved to an fpt-result.

▶ **Theorem 7.1.** *There is a $q \in \mathbb{N}$ such that the following parameterized problem is* $\mathsf{W}[1]$*-hard.*

Instance: graph G of clique-width at most 2, sequence $\mathcal{a} = (\bar{a}_1, \dots, \bar{a}_m) \in \big((V(G))^2\big)^m$, function $\sigma\colon [m] \to \{+1, -1\}$, formula $\varphi(\bar{x}, \bar{y}) \in \mathsf{MSO}(\Lambda, q, 2, \ell, 0)$
Parameter: ℓ
Problem: decide if $(G, \mathcal{a}, \sigma)$ is φ-consistent.

Proof sketch. We prove this by a reduction from the $\mathsf{W}[1]$-complete *weighted satisfiability problem* for Boolean formulas in 2-conjunctive normal form [26]. The *weight* of an assignment to a set of Boolean variables is the number of variables set to 1.

WSat(2-CNF)

Instance: Boolean formula Φ in 2-CNF
Parameter: ℓ
Problem: decide if Φ has a satisfying assignment of weight ℓ.

Given a 2-CNF formula $\Phi = \bigwedge_{i=1}^{m} (L_{i,1} \vee L_{i,2})$ in the variables $\{X_1, \dots, X_n\}$ and $\ell \in \mathbb{N}$, we construct an instance $(G, \mathcal{a}, \sigma, \varphi)$ of the consistency problem from Theorem 7.1 where the graph G is a forest that encodes Φ. Moreover, $\mathcal{a}$, φ, and σ are chosen to verify that a φ-witness $\bar{w} \in (V(G))^\ell$ for $(G, \mathcal{a}, \sigma)$ encodes exactly ℓ variables among $X_1, \dots, X_n$ that can be set to 1 to satisfy Φ. Hence, there is a φ-witness for $(G, \mathcal{a}, \sigma)$ if and only if Φ has a satisfying assignment of weight ℓ. Details on the construction can be found in the full version [10]. ◀

8 Conclusion

Just like model checking and the associated counting and enumeration problems, the learning problem we study here is a natural algorithmic problem for logics on finite structures. All these problems are related, but each has its own challenges requiring different techniques. Where model checking and enumeration are motivated by automated verification and database systems, we view our work as part of a descriptive complexity theory of machine learning [8].

The first problem we studied is 1D-MSO-Consistent-Learn, where the instances to classify consist of single vertices, and we extended the previous fixed-parameter tractability results for strings and trees [31, 30] to (labeled) graphs of bounded clique-width. Moreover, on general graphs, we showed that the problem is hard for the complexity class para-NP.

For MSO-learning problems in higher dimensions, we presented two different approaches that yield tractability results on graphs of bounded clique-width. For the agnostic PAC-learning problem MSO-PAC-Learn, we described a fixed-parameter tractable learning algorithm. Furthermore, in the consistent-learning setting for higher dimensions, we gave an algorithm that solves the learning problem and is fixed-parameter tractable in the size of the input graph. However, the algorithm is not fixed-parameter tractable in the size of the training sequence, and we showed that this is optimal.

In the learning problems considered so far, hypotheses are built using MSO formulas and tuples of vertices as parameters. We think that the algorithms presented in this paper for the 1-dimensional case could also be extended to hypothesis classes that allow tuples of sets as parameters. Finally, utilizing the full power of MSO, one could also consider a learning problem where, instead of classifying tuples of vertices, we are interested in classifying sets of vertices. That is, for a graph G, we are given labeled subsets of $V(G)$ and want to find a hypothesis $h\colon 2^{V(G)} \to \{+,-\}$ that is consistent with the given examples. It is easy to see that the techniques used in our hardness result also apply to this modified problem, proving that it is para-NP-hard. However, it remains open whether our tractability results could also be lifted to this version of the problem.

References

1 Azza Abouzied, Dana Angluin, Christos H. Papadimitriou, Joseph M. Hellerstein, and Avi Silberschatz. Learning and verifying quantified boolean queries by example. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA, June 22–27, 2013*, pages 49–60. ACM, 2013. doi:10.1145/2463664.2465220.

2 Howard Aizenstein, Tibor Hegedüs, Lisa Hellerstein, and Leonard Pitt. Complexity theoretic hardness results for query learning. *Comput. Complex.*, 7(1):19–53, 1998. doi:10.1007/PL00001593.

3 Bogdan Alexe, Balder ten Cate, Phokion G. Kolaitis, and Wang Chiew Tan. Characterizing schema mappings via data examples. *ACM Trans. Database Syst.*, 36(4):23:1–23:48, 2011. doi:10.1145/2043652.2043656.

4 Vikraman Arvind, Johannes Köbler, and Wolfgang Lindner. Parameterized learnability of k-juntas and related problems. In *Algorithmic Learning Theory, 18th International Conference, ALT 2007, Sendai, Japan, October 1–4, 2007*, volume 4754 of *Lecture Notes in Computer Science*, pages 120–134. Springer, 2007. doi:10.1007/978-3-540-75225-7_13.

5 Pablo Barceló, Alexander Baumgartner, Víctor Dalmau, and Benny Kimelfeld. Regularizing conjunctive features for classification. *J. Comput. Syst. Sci.*, 119:97–124, 2021. doi:10.1016/j.jcss.2021.01.003.

6 Pablo Barceló and Miguel Romero. The complexity of reverse engineering problems for conjunctive queries. In *20th International Conference on Database Theory, ICDT 2017, Venice, Italy, March 21–24, 2017*, volume 68 of *LIPIcs*, pages 7:1–7:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ICDT.2017.7.

7 Steffen van Bergerem. Learning concepts definable in first-order logic with counting. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24–27, 2019*, pages 1–13. IEEE, 2019. doi:10.1109/LICS.2019.8785811.

8 Steffen van Bergerem. *Descriptive Complexity of Learning.* PhD thesis, RWTH Aachen University, Germany, 2023. doi:10.18154/RWTH-2023-02554.

9 Steffen van Bergerem, Martin Grohe, and Martin Ritzert. On the parameterized complexity of learning first-order logic. In *PODS 2022: International Conference on Management of Data, Philadelphia, PA, USA, June 12–17, 2022*, pages 337–346. ACM, 2022. doi:10.1145/3517804.3524151.

10 Steffen van Bergerem, Martin Grohe, and Nina Runde. The parameterized complexity of learning monadic second-order logic, 2023. doi:10.48550/arXiv.2309.10489.

11 Steffen van Bergerem and Nicole Schweikardt. Learning concepts described by weight aggregation logic. In *29th EACSL Annual Conference on Computer Science Logic, CSL 2021, Ljubljana, Slovenia (Virtual Conference), January 25–28, 2021*, volume 183 of *LIPIcs*, pages 10:1–10:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.CSL.2021.10.

12 Anselm Blumer, A. Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *J. ACM*, 36(4):929–965, October 1989. doi:10.1145/76359.76371.

13 Angela Bonifati, Radu Ciucanu, and Aurélien Lemay. Learning path queries on graph databases. In *Proceedings of the 18th International Conference on Extending Database Technology, EDBT 2015, Brussels, Belgium, March 23–27, 2015*, pages 109–120. OpenProceedings.org, 2015. doi:10.5441/002/edbt.2015.11.

14 Angela Bonifati, Radu Ciucanu, and Slawek Staworko. Learning join queries from user examples. *ACM Trans. Database Syst.*, 40(4):24:1–24:38, 2016. doi:10.1145/2818637.

15 Angela Bonifati, Ugo Comignani, Emmanuel Coquery, and Romuald Thion. Interactive mapping specification with exemplar tuples. *ACM Trans. Database Syst.*, 44(3):10:1–10:44, 2019. doi:10.1145/3321485.

16 Cornelius Brand, Robert Ganian, and Kirill Simonov. A parameterized theory of PAC learning. In *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7–14, 2023*, pages 6834–6841. AAAI Press, 2023. doi:10.1609/aaai.v37i6.25837.

17 Balder ten Cate and Víctor Dalmau. Conjunctive queries: Unique characterizations and exact learnability. In *24th International Conference on Database Theory, ICDT 2021, Nicosia, Cyprus, March 23–26, 2021*, volume 186 of *LIPIcs*, pages 9:1–9:24. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICDT.2021.9.

18 Balder ten Cate, Víctor Dalmau, and Phokion G. Kolaitis. Learning schema mappings. *ACM Trans. Database Syst.*, 38(4):28:1–28:31, 2013. doi:10.1145/2539032.2539035.

19 Balder ten Cate, Phokion G. Kolaitis, Kun Qian, and Wang-Chiew Tan. Active learning of GAV schema mappings. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Houston, TX, USA, June 10–15, 2018*, pages 355–368. ACM, 2018. doi:10.1145/3196959.3196974.

20 Adrien Champion, Tomoya Chiba, Naoki Kobayashi, and Ryosuke Sato. ICE-based refinement type discovery for higher-order functional programs. *J. Autom. Reason.*, 64(7):1393–1418, 2020. doi:10.1007/s10817-020-09571-y.

21 William W. Cohen and C. David Page Jr. Polynomial learnability and inductive logic programming: Methods and results. *New Gener. Comput.*, 13(3&4):369–409, December 1995. doi:10.1007/BF03037231.

22 Derek G. Corneil and Udi Rotics. On the relationship between clique-width and treewidth. *SIAM J. Comput.*, 34(4):825–847, 2005. doi:10.1137/S0097539701385351.

23 Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory of Computing Systems*, 33(2):125–150, 2000. doi:10.1007/s002249910009.

24 Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discret. Appl. Math.*, 101(1-3):77–114, 2000. doi:10.1016/S0166-218X(99)00184-5.

25 P. Ezudheen, Daniel Neider, Deepak D'Souza, Pranav Garg, and P. Madhusudan. Horn-ICE learning for synthesizing invariants and contracts. *Proc. ACM Program. Lang.*, 2(OOPSLA):131:1–131:25, 2018. doi:10.1145/3276501.

26 Jörg Flum and Martin Grohe. *Parameterized complexity theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006. doi:10.1007/3-540-29953-X.

27 Martin Fürer. Multi-clique-width. In *8th Innovations in Theoretical Computer Science Conference, ITCS 2017, Berkeley, CA, USA, January 9–11, 2017*, volume 67 of *LIPIcs*, pages 14:1–14:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICS.ITCS.2017.14.

28 Pranav Garg, Christof Löding, P. Madhusudan, and Daniel Neider. ICE: A robust framework for learning invariants. In *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18–22, 2014*, volume 8559 of *Lecture Notes in Computer Science*, pages 69–87. Springer, 2014. doi:10.1007/978-3-319-08867-9_5.

29 Georg Gottlob and Pierre Senellart. Schema mapping discovery from data instances. *J. ACM*, 57(2):6:1–6:37, 2010. doi:10.1145/1667053.1667055.

30 Émilie Grienenberger and Martin Ritzert. Learning definable hypotheses on trees. In *22nd International Conference on Database Theory, ICDT 2019, Lisbon, Portugal, March 26–28, 2019*, volume 127 of *LIPIcs*, pages 24:1–24:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICDT.2019.24.

31 Martin Grohe, Christof Löding, and Martin Ritzert. Learning MSO-definable hypotheses on strings. In *International Conference on Algorithmic Learning Theory, ALT 2017, Kyoto University, Kyoto, Japan, October 15–17, 2017*, volume 76 of *Proceedings of Machine Learning Research*, pages 434–451. PMLR, October 2017. ISSN: 2640-3498. URL: https://proceedings.mlr.press/v76/grohe17a.html.

32 Martin Grohe and Martin Ritzert. Learning first-order definable concepts over structures of small degree. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavík, Iceland, June 20–23, 2017*, pages 1–12. IEEE, June 2017. doi:10.1109/LICS.2017.8005080.

33 Martin Grohe and György Turán. Learnability and definability in trees and similar structures. *Theory Comput. Syst.*, 37(1):193–220, January 2004. doi:10.1007/s00224-003-1112-8.

34 David Haussler. Learning conjunctive concepts in structural domains. *Mach. Learn.*, 4:7–40, 1989. doi:10.1007/BF00114802.

35 David Haussler. Decision theoretic generalizations of the PAC model for neural net and other learning applications. *Inf. Comput.*, 100(1):78–150, 1992. doi:10.1016/0890-5401(92)90010-D.

36 Kouichi Hirata. On the hardness of learning acyclic conjunctive queries. In *Algorithmic Learning Theory, 11th International Conference, ALT 2000, Sydney, Australia, December 11–13, 2000*, volume 1968 of *Lecture Notes in Computer Science*, pages 238–251. Springer, 2000. doi:10.1007/3-540-40992-0_18.

37 Michael J. Kearns and Umesh V. Vazirani. *An Introduction to Computational Learning Theory.* MIT Press, 1994. URL: https://mitpress.mit.edu/books/introduction-computational-learning-theory.

38 Benny Kimelfeld and Christopher Ré. A relational framework for classifier engineering. *ACM Trans. Database Syst.*, 43(3):11:1–11:36, 2018. doi:10.1145/3268931.

39 Yuanzhi Li and Yingyu Liang. Learning mixtures of linear regressions with nearly optimal complexity. In *Conference On Learning Theory, COLT 2018, Stockholm, Sweden, July 6–9, 2018*, volume 75 of *Proceedings of Machine Learning Research*, pages 1125–1144. PMLR, 2018. URL: http://proceedings.mlr.press/v75/li18b.html.

40 Christof Löding, P. Madhusudan, and Daniel Neider. Abstract learning frameworks for synthesis. In *Tools and Algorithms for the Construction and Analysis of Systems - 22nd International Conference, TACAS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2–8, 2016*, volume 9636 of *Lecture Notes in Computer Science*, pages 167–185. Springer, 2016. doi:10.1007/978-3-662-49674-9_10.

41 Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning.* Adaptive computation and machine learning. MIT Press, 2nd edition, 2018.

42 Stephen H. Muggleton. Inductive logic programming. *New Gener. Comput.*, 8(4):295–318, February 1991. doi:10.1007/BF03037089.

43 Stephen H. Muggleton and Luc De Raedt. Inductive logic programming: Theory and methods. *J. Log. Program.*, 19/20:629–679, 1994. doi:10.1016/0743-1066(94)90035-3.

44 Sang-il Oum and Paul D. Seymour. Approximating clique-width and branch-width. *J. Comb. Theory, Ser. B*, 96(4):514–528, 2006. doi:10.1016/j.jctb.2005.10.006.

45 Norbert Sauer. On the density of families of sets. *J. Comb. Theory, Ser. A*, 13(1):145–147, 1972. doi:10.1016/0097-3165(72)90019-2.

46 Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning - From Theory to Algorithms.* Cambridge University Press, Cambridge, 2014. doi:10.1017/CBO9781107298019.

47 Saharon Shelah. A combinatorial problem: stability and order for models and theories in infinitary languages. *Pacific Journal of Mathematics*, 41(1):247–261, 1972. doi:10.2140/pjm.1972.41.247.

48 Robert H. Sloan, Balázs Szörényi, and György Turán. Learning Boolean functions with queries. In *Boolean Models and Methods in Mathematics, Computer Science, and Engineering*, pages 221–256. Cambridge University Press, 2010. doi:10.1017/cbo9780511780448.010.

49 Slawek Staworko and Piotr Wieczorek. Learning twig and path queries. In *15th International Conference on Database Theory, ICDT 2012, Berlin, Germany, March 26–29, 2012*, pages 140–154. ACM, 2012. doi:10.1145/2274576.2274592.

50 Leslie G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, November 1984. doi:10.1145/1968.1972.

51 Vladimir Vapnik. Principles of risk minimization for learning theory. In *Advances in Neural Information Processing Systems 4, [NIPS Conference, Denver, Colorado, USA, December 2–5, 1991]*, pages 831–838, 1991. URL: https://proceedings.neurips.cc/paper_files/paper/1991/file/ff4d5fbbafdf976cfdc032e3bde78de5-Paper.pdf.

52 He Zhu, Stephen Magill, and Suresh Jagannathan. A data-driven CHC solver. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2018, Philadelphia, PA, USA, June 18–22, 2018*, pages 707–721. ACM, 2018. doi:10.1145/3192366.3192416.