


Consistency-Checking Problems: A Gateway to Parameterized Sample Complexity

Robert Ganian ✉ 🏠 

Technische Universität Wien, Vienna, Austria

Liana Khazaliya ✉ 🏠 

Technische Universität Wien, Vienna, Austria

Kirill Simonov ✉ 🏠 

Hasso Plattner Institute, Universität Potsdam, Germany

Abstract

Recently, Brand, Ganian and Simonov introduced a parameterized refinement of the classical PAC-learning sample complexity framework. A crucial outcome of their investigation is that for a very wide range of learning problems, there is a direct and provable correspondence between fixed-parameter PAC-learnability (in the sample complexity setting) and the fixed-parameter tractability of a corresponding “consistency checking” search problem (in the setting of computational complexity). The latter can be seen as generalizations of classical search problems where instead of receiving a single instance, one receives multiple yes- and no-examples and is tasked with finding a solution which is consistent with the provided examples.

Apart from a few initial results, consistency checking problems are almost entirely unexplored from a parameterized complexity perspective. In this article, we provide an overview of these problems and their connection to parameterized sample complexity, with the primary aim of facilitating further research in this direction. Afterwards, we establish the fixed-parameter (in)-tractability for some of the arguably most natural consistency checking problems on graphs, and show that their complexity-theoretic behavior is surprisingly very different from that of classical decision problems. Our new results cover consistency checking variants of problems as diverse as $(k-)$ PATH, MATCHING, 2-COLORING, INDEPENDENT SET and DOMINATING SET, among others.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases consistency checking, sample complexity, fixed-parameter tractability

Funding *Robert Ganian*: Austrian Science Fund (FWF) [Y1329].

Liana Khazaliya: Vienna Science and Technology Fund (WWTF) [10.47379/ICT22029]; Austrian Science Fund (FWF) [Y1329]; European Union’s Horizon 2020 COFUND programme [LogiCS@TUWien, grant agreement No. 101034440].

Kirill Simonov: DFG Research Group ADYN via grant DFG 411362735.

1 Introduction

While the notion of time complexity is universally applicable and well studied across the whole spectrum of theoretical computer science, on its own it cannot capture the performance of the kinds of algorithms typically studied in the context of machine learning: **learning algorithms**. That is the domain of sample complexity, and here we will focus on the notion of (efficient) *PAC learning* [22, 16]—arguably the most classical, fundamental and widely known sample complexity framework. An important trait of PAC learning is that while it is built on different principles than time complexity, the two frameworks are connected in a way which allows us to translate intractability and tractability results from one domain to another. It is precisely this connection that gave rise to famous lower bounds in the PAC learning setting, such as the inability to efficiently and properly learn 3-term DNF and

3-clause CNF formulas [20, 2] under the assumption that $P \neq NP$, and *consistency checking problems* form the pillar of this connection.

Given the success of parameterized complexity as a concept generalizing classical time complexity analysis, it would seem natural to ask whether its principles can also be used to obtain a deeper understanding of efficient PAC-learnability. Brand, Ganian and Simonov [6] very recently introduced the foundations for a parameterized theory of PAC learning, which crucially also includes a bridge to parameterized complexity theory in the usual time complexity setting. The primary goal of this article is to show how the parameterized complexity paradigm can be used to draw new boundaries of tractability in the PAC learning domain, and to provide the parameterized algorithms community with an understanding of the parameterized consistency checking problems which allow us to travel between the sample and time complexity settings in the parameterized regime. We showcase the tools that can be used to deal with parameterized consistency checking problems and the obstacles that await there in the domain of graph problems, where we obtain new algorithmic upper and lower bounds for consistency checking variants of multiple natural problems on graphs.

A Gentle Introduction to PAC Learning. It will be useful to set the stage with a high-level and informal example of the setting in which PAC learning operates¹. Let us imagine we would like to “learn” a way of labeling points in a plane as either “good” or “bad”, knowing that the good points are precisely those contained in some unknown axis-parallel rectangle R in the plane. A learning algorithm in the PAC regime would be allowed to ask for a set of correctly labeled sample points, each of which would be drawn from some unknown distribution D , and would attempt to use these to “learn” R (so that it can use it to label any point that it looks at, even those which were not given as samples). This mental experiment is useful since it immediately clarifies that

- there is some probability that a PAC learning algorithm completely fails, since the samples we receive could be non-representative (*for instance, there is a non-zero probability that even if D is uniform and R is small, the sample points could all be drawn from inside R*), and
- even if a PAC learning algorithm intuitively “works correctly”, it is essentially guaranteed that it will not classify some samples (i.e., the sample points) correctly (*for instance, there could be points that lie close to the exact boundary of R which are unlikely to be drawn as samples based on D , making it impossible to obtain the exact position of R*).

Given these natural limitations, we can informally explain what it means for a learning problem to be *efficiently PAC-learnable*: it admits an algorithm which

1. takes as input a sample size n , a confidence measure δ and an accuracy measure ε ,
2. runs in time $(n + \frac{1}{\delta} + \frac{1}{\varepsilon})^{\mathcal{O}(1)}$ and asks for $(n + \frac{1}{\delta} + \frac{1}{\varepsilon})^{\mathcal{O}(1)}$ samples, and then
3. outputs something which will, with probability at least $1 - \delta$, “work correctly” in almost all cases (measured by ε).

It needs to be clarified that beyond the study of efficient PAC learnability, a substantial amount of fundamental work in the PAC learning direction has also been carried out on whether a problem is PAC learnable at all [4, 14, 1], on the distinction between so-called proper and improper learning [16, 5], and on many other aspects and considerations that lie outside of the scope of this paper. Here, our focus lies on how the gap between efficient and “non-efficient” PAC-learnability of learning problems can be bridged by the parameterized

¹ Formal definitions are provided in Section 2.

PAC learning framework of Brand, Ganian and Simonov [6], and the associated study of consistency checking problems.

To illustrate how parameterized complexity can be used here, let us turn to a different example of a simple learning problem that is based on the idea of representing cyber-attacks as graphs proposed, e.g., by Sheyner and Wing [21, 24]. Assume we have a network consisting of n nodes which is protected by k hidden defense nodes. A cyberattack on this network can be represented as a set of edges over the n nodes, and is evaluated as successful if and only if an edge in that attack is not incident to any defense node (i.e., an attack fails if and only if the defense nodes form a vertex cover of the attack edges). Individual samples represent attacks made on the network, and the learning task is to identify all the defense nodes. This problem corresponds to VERTEX COVER LEARNING [9], which Brand, Ganian and Simonov showed to admit a PAC learning algorithm which requires polynomially many samples but time $2^k \cdot (n + \frac{1}{\delta} + \frac{1}{\epsilon})^{\mathcal{O}(1)}$ where k is the size of the sought-after vertex cover [6]. This is a prototypical representative of the class $\text{FPT-PAC}_{\text{time}}$. We remark that in the context of PAC learning, one explicitly distinguishes between the time required by the learning algorithm and the number of samples it uses, as the latter may in some contexts be much more difficult to obtain. A picture of the parameterized complexity landscape above efficient PAC learnability is provided later together with the formal definitions (see Figure 1).

Crucially, whenever we are dealing with a learning problem $\mathcal{P}_{\text{learn}}$ where the size of the hypothesis space (i.e., the number of “possible outputs”) is upper-bounded by a certain function (see Theorem 11), the parameterized sample complexity of $\mathcal{P}_{\text{learn}}$ can be directly and formally linked to the parameterized time complexity of the consistency checking variant $\mathcal{P}_{\text{cons}}$ of the same problem [6], where the task is to compute a “solution” (a hypothesis) which is consistent with a provided set of positive and negative examples. This motivates the systematic study of parameterized consistency checking problems, an area which has up to now remained almost entirely unexplored from the perspective of fixed-parameter (in-)tractability.

The Parameterized Complexity of Consistency Checking on Graphs. A few initial examples of parameterized consistency checking problems have been solved by the theory-building work of Brand, Ganian and Simonov [6]; in particular, they showed that consistency checking for vertex-deletion problems where the base class \mathcal{H} can be characterized by a finite set of forbidden induced subgraphs is fixed-parameter tractable (which implies the aforementioned fact that VERTEX COVER LEARNING is in $\text{FPT-PAC}_{\text{time}}$), but no analogous result can be obtained for all classes \mathcal{H} characterized by a finite set of forbidden minors unless $\text{FPT} \neq \text{W}[1]$.

In this article, we expand on these results by establishing the fixed-parameter (in-)tractability of consistency checking for several other classical graph problems whose decision versions are well-known to the parameterized complexity community. The aim here is to showcase how parameterized upper- and lower-bound techniques fare when dealing with these new kinds of problems.

It is important to note that the tractability of consistency checking requires the tractability of the corresponding decision/search problem (as the latter can be seen as a special case of consistency checking), but the former can be much more algorithmically challenging than the latter: many trivial decision problems become computationally intractable in the consistency checking regime. We begin by illustrating this behavior on the classical 2-COLORING problem, i.e., the task of partitioning the vertices of the graph into two independent sets. We show that while consistency checking for 2-COLORING is intractable (and hence a 2-coloring is not efficiently PAC-learnable), consistency checking for SPLIT GRAPH, i.e., the task of

partitioning the vertices into an independent set and a clique, is polynomial-time tractable.

Moving on to parameterized problems, we begin by considering three classical edge search problems, notably MATCHING, (k -)PATH and EDGE CLIQUE COVER. In the classical decision or search settings, the first problem is polynomial-time solvable while the latter two admit well-known fixed-parameter algorithms. Interestingly, we show that consistency checking for the former two problems is $W[2]$ -hard², but is fixed-parameter tractable for the third, i.e., EDGE CLIQUE COVER.

Next, we turn our attention to the behavior of two classical vertex search problems, specifically INDEPENDENT SET and DOMINATING SET. While both problems are fixed-parameter intractable already in the classical search regime, here we examine their behavior on bounded-degree graphs (where they are well-known to be fixed-parameter tractable). Again, the consistency checking variants of these problems on bounded-degree graphs exhibit a surprising complexity-theoretic behavior: DOMINATING SET is FPT, but INDEPENDENT SET is $W[2]$ -hard even on bounded-degree graphs.

As the final contribution of the paper, we show that most of the aforementioned consistency checking lower bounds can be overcome if one additionally parameterizes by the number of negative samples. In particular, we obtain fixed-parameter consistency checking algorithms for 2-COLORING, MATCHING and (k -)PATH when we additionally assume that the number of negative samples is upper-bounded by the parameter. On the other hand, INDEPENDENT SET remains fixed-parameter intractable (at least $W[1]$ -hard) even under this additional restriction. As our final result, we show that INDEPENDENT SET becomes fixed-parameter tractable if we instead consider the total number of samples (i.e., both positive and negative) as an additional parameter. The proofs of these results are more involved than those mentioned in the previous paragraphs and rely on auxiliary graph constructions in combination with color coding. We remark that the parameterization by the number of negative samples in the consistency checking regime could be translated into a corresponding parameterization of the distribution in the PAC learning framework. A summary of our individual results for consistency checking problems is provided in Table 1.

Related Work. The connection between parameterized learning problems and parameterized consistency checking was also hinted at in previous works that studied the (parameterized) sample complexity of learning juntas [3] or learning first-order logic [23]. Moreover, the problem of computing optimal decision trees, which has received a significant amount of recent attention [19, 11], can also be seen as a consistency checking problem where the sought-after solution is a decision tree.

2 Preliminaries

We assume familiarity with basic graph terminology [10] and parameterized complexity theory [7]. We use $[t]$ to denote the set $\{1, \dots, t\}$. For brevity, we will denote sets of tuples of the form $\{(\alpha_1, \beta_1), \dots, (\alpha_t, \beta_t)\}$ as $(\alpha_i, \beta_i)_{i \in [t]}$, and the set of two-element subsets of a set Z as $\binom{Z}{2}$. As basic notation and terminology, we set $\{0, 1\}^* = \bigcup_{m \in \mathbb{N}} \{0, 1\}^m$. A *distribution* on $\{0, 1\}^n$ is a mapping $\mathcal{D}_n : \{0, 1\}^n \rightarrow [0, 1]$ such that $\sum_{x \in \{0, 1\}^n} \mathcal{D}_n(x) = 1$, and the *support* of \mathcal{D}_n is the set $\text{supp } \mathcal{D}_n = \{x \mid \mathcal{D}_n(x) > 0\}$.

² More precisely, a fixed-parameter algorithm for either of these problems would imply $\text{FPT} = W[1]$ (see Section 4).

Problem	Decision/Search	Consistency Checking	Consistency Checking[samples]
2-COLORING	P	NP-hard (Thm. 12)	FPT (Thm. 16)
SPLIT GRAPH	P	P (Thm. 14)	—
MATCHING	P	W[2]-hard (Thm. 17)	FPT (Thm. 20)
(k)-PATH	FPT	W[2]-hard (Thm. 18)	FPT (Thm. 21)
EDGE CLIQUE COVER	FPT	FPT (Thm. 19)	—
INDEPENDENT SET[degree]	FPT	W[2]-hard (Thm. 22)	W[1]-hard* (Thm. 24, 25)
DOMINATING SET[degree]	FPT	FPT (Thm. 23)	—

■ **Table 1** An overview of the concrete results obtained for consistency checking problems in this article, where the columns provide a comparison between the complexity of the decision/search variant, the consistency checking variant, and the consistency checking variant where the number of negative samples is taken as an additional parameter. Problems marked with “[degree]” are considered over bounded-degree input graphs/samples, and the “*” marks that the problem becomes fixed-parameter tractable when additionally parameterized by the total number of samples. The lower bounds stated in the table are simplified; the precise formal statements are provided in the appropriate theorems.

2.1 Consistency Checking

While the original motivation for consistency checking problems originates from specific applications in PAC learning, one can define a consistency checking version of an arbitrary *search problem*.

In a search problem, we are given an instance $I \in \{0, 1\}^*$, and the task is to find a solution $S \in \{0, 1\}^*$, where the solution is verified by a predicate $\phi(\cdot, \cdot)$, so that $\phi(I, S)$ is true if and only if S is a solution to I . Since our focus here will lie on problems which are in NP, the predicate $\phi(\cdot, \cdot)$ will in all cases be polynomial-time computable. In the context of graph problems, I will typically be a graph (possibly with some auxiliary information such as edge weights or the bound on solution size), and S could be a set of vertices, a set of edges, a partitioning of the vertex set, etc. For example, in the search version of the VERTEX COVER problem the input is a graph G together with a bound k on the size of the target vertex cover, potential solutions are subsets of $V(G)$, and a subset S is a solution if and only if the size of S is k and S covers all edges of the graph G . One can then write the verifying predicate as

$$\phi((G, k), S) = (S \subset V(G)) \wedge (|S| = k) \wedge (\forall \{u, v\} \in E(G), \{u, v\} \cap S \neq \emptyset).$$

For a search problem \mathcal{P} , we define the corresponding consistency checking problem $\mathcal{P}_{\text{cons}}$ as follows. Instead of receiving a single instance $I \in \{0, 1\}^*$ as input, we receive a set of labeled samples $\mathcal{I} = \{(I_1, \lambda_1), (I_2, \lambda_2), \dots, (I_t, \lambda_t)\}$ where each $I_i, i \in [t]$, is an element of $\{0, 1\}^*$ and $\lambda_i \in \{0, 1\}$. The task is to compute a (*consistent*) *solution* $S \subset \{0, 1\}^*$ such that $\phi(I_i, S)$ holds if and only if $\lambda_i = 1$, for each $i \in [t]$, or to correctly determine that no such solution exists.

In the example of VERTEX COVER, for each $i \in [t]$, the instance is the pair (G_i, k_i) , so that the target solution has to be a vertex subset³ of $V(G_i)$, of size k_i , and it has to cover

³ The property of being a subset is given by the implicit encoding in $\{0, 1\}^*$, e.g., vertices in all $V(G_i)$ and S are indexed by integers, and is defined in the same way across all instances. We thus say that S could be a subset of all $V(G_i)$ even though, formally speaking, these are disjoint sets.

all edges of G_i , for each $i \in [t]$. Since vertices in all G_i 's and S are implicitly associated with their respective counterparts in the other graphs, we can instead treat the graphs G_i as defined over the same vertex set. Also, for instances $i \in [t]$ where $\lambda_i = 1$, if their values of k_i mismatch, then there is clearly no solution; and for those $i \in [t]$ with $\lambda_i = 0$, if the value k_i does not match the respective value of a positive sample, then the condition for λ_i is always satisfied. Therefore, we can equivalently reformulate the consistency checking version of VERTEX COVER as follows: Given the vertex set V , a number k , and a sequence of labeled edge sets $(E_1, \lambda_1), \dots, (E_t, \lambda_t)$, over V , is there a subset $S \subset V$ of size exactly k , so that S covers all edges of E_i if and only if $\lambda_i = 1$, for each $i \in [t]$?

One can immediately observe that the polynomial-time tractability of a search problem is a prerequisite for the polynomial-time tractability of the corresponding consistency checking problem. At the same time, the exact definition of the search problem (and in particular the solution S) can have a significant impact on the complexity of the consistency checking problem. We remark that there are two possible ways one can parameterize a consistency checking problem: one either uses the parameter to restrict the sought-after solution S , or the input \mathcal{I} . Each of these approaches can be tied to a parameterization of the corresponding PAC learning problem (see Subsection 2.3).

Formally, we say that $(\mathcal{P}_{\text{cons}}, \kappa, \lambda)$ is a parameterized consistency checking problem, where $\mathcal{P}_{\text{cons}}$ is a consistency checking problem, κ maps solutions $S \in \{0, 1\}^*$ to natural numbers, and λ maps lists of labeled instances $((I_1, \lambda_1), \dots, (I_t, \lambda_t))$, $I_i \in \{0, 1\}^*$, $\lambda_i \in \{0, 1\}$, to natural numbers. The input is then a list of labeled instances $\mathcal{L} = ((I_1, \lambda_1), \dots, (I_t, \lambda_t))$ together with parameters k, ℓ , such that $\ell = \lambda(\mathcal{L})$, and the task is to find a consistent solution S with $\kappa(S) = k$. For example, k could be a size bound on the targeted solution, and ℓ could be the maximum degree in any of the given graphs or the number of instances with $\lambda_i = 0$.

2.2 PAC-Learning

The remainder of this section is dedicated to a more formal introduction of the foundations of parameterized PAC learning theory and its connection to parameterized consistency checking problems. We note that while the content of the following subsections is important to establish the implications and corollaries of the results obtained in the article, readers who are interested solely in the obtained complexity-theoretic upper and lower bounds for consistency checking problems can safely skip them and proceed directly to Section 3.

To make the connection between consistency checking problems and parameterized sample complexity clear, we first recall the formalization of the classical theory of PAC learning [22, 17].

► **Definition 1.** A concept is an arbitrary Boolean function $c : \{0, 1\}^n \rightarrow \{0, 1\}$. An assignment $x \in \{0, 1\}^n$ is called a positive sample for c if $c(x) = 1$, and a negative sample otherwise. A concept class \mathcal{C} is a set of concepts. For every $m \in \mathbb{N}$, we write $\mathcal{C}_m = \mathcal{C} \cap \mathcal{B}_m$, where \mathcal{B}_m is the set of all m -ary Boolean functions.

► **Definition 2.** Let \mathcal{C} be a concept class. A surjective mapping $\rho : \{0, 1\}^* \rightarrow \mathcal{C}$ is called a representation scheme of \mathcal{C} .

We call each r with $\rho(r) = c$ a representation of concept c .

► **Definition 3.** A learning problem is a pair (\mathcal{C}, ρ) , where \mathcal{C} is a concept class and ρ is a representation scheme for \mathcal{C} .

► **Definition 4.** A learning algorithm for a learning problem (\mathcal{C}, ρ) is a randomized algorithm such that:

1. It obtains the values n, ε, δ as inputs, where n is an integer and $0 < \varepsilon, \delta \leq 1$ are rational numbers.
2. It has access to a hidden representation r^* of some concept $c^* = \rho(r^*)$ and a hidden distribution \mathcal{D}_n on $\{0, 1\}^n$ through an oracle that returns labeled samples $(x, c^*(x))$, where $x \in \{0, 1\}^n$ is drawn at random from \mathcal{D}_n .
3. The output of the algorithm is a representation of some concept, called its hypothesis.

When dealing with individual instances of a learning problem, we will use $s = |r^*|$ to denote the size of the hidden representation.

► **Definition 5.** Let \mathcal{A} be a learning algorithm. Fix a hidden hypothesis c^* and a distribution on $\{0, 1\}^n$. Let h be a hypothesis output by \mathcal{A} and $c = \rho(h)$ be the concept h represents. We define

$$\text{err}_h = \mathbb{P}_{x \sim \mathcal{D}_n}(c(x) \neq c^*(x))$$

as the probability of the hypothesis and the hidden concept disagreeing on a sample drawn from \mathcal{D}_n , the so-called generalization error of h under \mathcal{D}_n .

The algorithm \mathcal{A} is called probably approximately correct (PAC) if it outputs a hypothesis h such that $\text{err}_h \leq \varepsilon$ with probability at least $1 - \delta$.

Usually, learning problems in this framework are regarded as tractable if they are PAC-learnable within polynomial time bounds. More precisely, we say that a learning problem L is *efficiently* PAC-learnable if there is a PAC algorithm for L that runs in time polynomial in $n, s, 1/\varepsilon$ and $1/\delta$.

Consider now a classical search problem \mathcal{P} and its consistency checking version $\mathcal{P}_{\text{cons}}$. One can naturally define the corresponding learning problem $\mathcal{P}_{\text{learn}}$: For a solution $S \in \{0, 1\}^*$, let $\phi(\cdot, S)$ be a concept and S its representation; this describes the concept class and its representation scheme. Going back to the VERTEX COVER example, for each graph size N , the concepts are represented by subsets of $[N]$ (encoded in binary). For a subset $S \subset [N]$, the respective concept c_S is a binary function that, given the encoding of an instance E , returns 1 if and only if S is a vertex cover of $G = ([N], E)$ of size k , where $[N]$ is treated as the respective ‘‘ground’’ vertex set of size N . A PAC-learning algorithm for this problem is thus given a vertex set $V = [N]$, an integer k , and an oracle that will produce a sequence of samples $(E_1, \lambda_1), \dots, (E_t, \lambda_t)$, where the instances E_i are drawn from a hidden distribution \mathcal{D} . With probability at least $(1 - \delta)$, the algorithm has to return a subset $S \subset [N]$ that is consistent with an instance sampled from \mathcal{D} with probability at least $(1 - \varepsilon)$. In fact, for VERTEX COVER and many other problems, it is sufficient to return a hypothesis that is consistent only with the seen samples (E_i, λ_i) , $i \in [t]$; this is formalized in the next subsection.

Naturally, we do not expect the learning version of VERTEX COVER to be efficiently PAC-learnable, as even finding a vertex cover of a certain size in a single instance is NP-hard. This motivates the introduction of parameters into the framework, which is presented next. We also recall the complexity reductions between (parameterized) consistency checking problem and its respective (parameterized) learning problem, which in particular allows to formally transfer the hardness results such as NP-hardness above.

Remark. A more general definition of learning problems is sometimes considered in the literature, where the output of a learning algorithm need not necessarily be from the same concept class \mathcal{C} (e.g., it can be a sub- or a super-class of \mathcal{C}). This is usually called *improper learning*, as opposed to the classical setting of *proper learning* defined above and considered in this article.

2.3 Parameterized PAC-Learning

We now define parameterized learning problems and recall the connection to the consistency checking problems, as given by the framework of Brand, Ganian, and Simonov [6]. For brevity, we omit some of the less important technical details; interested readers can find the full technical exposition in the full description of the framework [6]

First we note that in parameterized PAC-learning, both the hidden concept and the hidden distribution can be parameterized, which is formally represented in the next definitions. We call a function κ from representations in $\{0, 1\}^*$ to natural numbers *parameterization of representations*, and a function λ assigning a natural number to every distribution on $\{0, 1\}^n$ for each n *parameterization of distributions*.

► **Definition 6** (Parameterized Learning Problems). *A parameterized learning problem is a learning problem (\mathcal{C}, ρ) together with a pair (κ, λ) , called its parameters, where κ is a parameterization of representations and λ is a parameterization of distributions.*

► **Definition 7** (Parameterized Learning Algorithm). *A parameterized learning algorithm for a parameterized learning problem $(\mathcal{C}, \rho, \kappa, \lambda)$ is a learning algorithm for (\mathcal{C}, ρ) in the sense of Definition 4. In addition to n, ε, δ , a parameterized learning algorithm obtains two inputs k and ℓ , which are promised to satisfy $k = \kappa(r^*)$ as well as $\ell = \lambda(\mathcal{D}_n)$, and the algorithm is required to always output a hypothesis h satisfying $\kappa(h) \leq k$.*

Let $\text{poly}(\cdot)$ denote the set of functions that can be bounded by non-decreasing polynomial functions in their arguments. Furthermore, $\text{fpt}(x_1, \dots, x_t; k_1, \dots, k_t)$ and $\text{xp}(x_1, \dots, x_t; k_1, \dots, k_t)$ denote those functions bounded by $f(k_1, \dots, k_t) \cdot p(x_1, \dots, x_t)$ and $p(x_1, \dots, x_t)^{f(k_1, \dots, k_t)}$, respectively, for any non-decreasing computable function f in k_1, \dots, k_t and $p \in \text{poly}(x_1, \dots, x_t)$.

► **Definition 8** ((T, S) -PAC Learnability). *Let $T(n, s, 1/\varepsilon, 1/\delta, k, \ell), S(n, s, 1/\varepsilon, 1/\delta, k, \ell)$ be any two functions taking on integer values, and non-decreasing in all of their arguments.*

A parameterized learning problem $\mathcal{L} = (\mathcal{C}, \rho, \{\mathcal{R}_k\}_{k \in \mathbb{N}}, \lambda)$ is (T, S) -PAC learnable if there is a PAC learning algorithm for \mathcal{L} that runs in time $\mathcal{O}(T(n, s, 1/\varepsilon, 1/\delta, k, \ell))$ and queries the oracle at most $\mathcal{O}(S(n, s, 1/\varepsilon, 1/\delta, k, \ell))$ times.

We denote the set of parameterized learning problems that are (T, S) -PAC learnable by $\text{PAC}[T, S]$. This is extended to sets of functions \mathbf{S}, \mathbf{T} through setting $\text{PAC}[T, S] = \bigcup_{S \in \mathbf{S}, T \in \mathbf{T}} \text{PAC}[T, S]$.

► **Definition 9.** *Define the complexity classes as follows:*

$$\text{FPT-PAC}_{\text{time}} = \text{PAC}[\text{fpt}, \text{poly}],$$

$$\text{FPT-PAC} = \text{PAC}[\text{fpt}, \text{fpt}],$$

$$\text{XP-PAC}_{\text{time}} = \text{PAC}[\text{xp}, \text{poly}],$$

$$\text{XP-PAC} = \text{PAC}[\text{xp}, \text{xp}],$$

where we fix

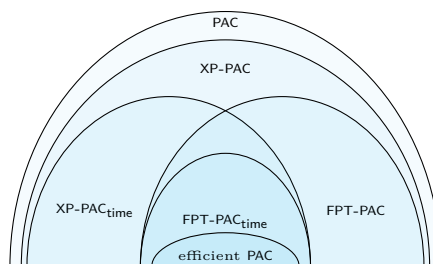
$$\text{poly} = \text{poly}(n, s, 1/\varepsilon, 1/\delta, k, \ell),$$

$$\text{fpt} = \text{fpt}(n, s, 1/\varepsilon, 1/\delta; k, \ell),$$

$$\text{xp} = \text{xp}(n, s, 1/\varepsilon, 1/\delta; k, \ell).$$

There are examples of natural problems falling into each of these classes [6]. In addition to the above, there is a fifth class that may be considered here: $\text{PAC}[\text{xp}, \text{fpt}]$. However, we are not aware of any natural problems residing there that are not given by the “lower” classes.

Figure 1 provides an overview of these complexity classes and their relationships.



■ **Figure 1** A schematic view of the parameterized learning classes defined in Definition 9.

2.4 Consistency Checking for PAC-Learning

We now recall the results tying the complexity of (parameterized) PAC-learning to (parameterized) consistency checking. We have already shown that a consistency checking problem can be transformed into a learning problem, by viewing the hidden solution as the representation of the hidden concept; the same operation can also be done the other way around. Moreover, this transformation can be performed while respecting the parameters. Let $\mathcal{P}_{\text{cons}}$ be a consistency checking problem, and let $\mathcal{P}_{\text{learn}}$ be the respective learning problem. Consider a parameterized version $(\mathcal{P}_{\text{cons}}, \kappa + \lambda)$ of $\mathcal{P}_{\text{cons}}$, where κ maps solutions $S \in \{0, 1\}^*$ to natural numbers, and λ maps lists of labeled instances $((I_1, \lambda_1), \dots, (I_t, \lambda_t))$, $I_i \in \{0, 1\}^*$, $\lambda_i \in \{0, 1\}$, to natural numbers. The parameterized learning problem is then $(\mathcal{P}_{\text{learn}}, \kappa, \lambda')$, where κ is given by the same function as the parameterization of representations, as representations of concepts are exactly the solutions in the original search problem, and $\lambda'(\mathcal{D})$ for a distribution \mathcal{D} is the maximum value of $\lambda(\mathcal{L})$, where \mathcal{L} is any set of labeled instances produced by sampling from \mathcal{D} .

It is well-known that, under the assumption that the hypothesis space is not too large, there is an equivalence between a learning problem being PAC-learnable and the corresponding consistency checking problem being solvable in randomized polynomial time [20]. Brand, Ganian and Simonov proved a generalization of this equivalence in the parameterized sense [6], which we recall next

► **Theorem 10** (Corollary of Theorem 3.17 [6]). *Let $\mathcal{P}_{\text{cons}}$ be a parameterized consistency checking problem, and $\mathcal{P}_{\text{learn}} = (\mathcal{C}, \rho, \kappa, \lambda)$ be its matching parameterized learning problem, where λ depends only on the support of the distribution.*

If $\mathcal{P}_{\text{learn}}$ is in FPT-PAC, then $\mathcal{P}_{\text{cons}}$ is in FPT.

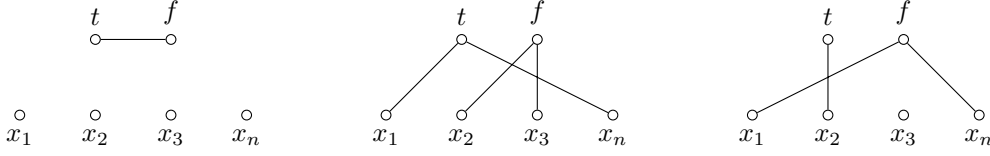
Similarly, if $\mathcal{P}_{\text{learn}}$ is in XP-PAC, then $\mathcal{P}_{\text{cons}}$ is in XP.

► **Theorem 11** (Corollary of Theorem 3.19 [6]). *Let $\mathcal{P}_{\text{cons}}$ be a parameterized consistency checking problem, and $\mathcal{P}_{\text{learn}} = (\mathcal{C}, \rho, \kappa, \lambda)$ be its matching parameterized learning problem. Denote the set of representations of concepts in $C \in \mathcal{C}$ of arity n with $\kappa(C) = k$ by $\mathcal{H}_{n,k}$.*

If $\mathcal{P}_{\text{cons}}$ is in FPT and $\log |\mathcal{H}_{n,k}| \in \text{fpt}(n; k)$, then \mathcal{L} is in FPT-PAC_{time}.

Similarly, if $\mathcal{P}_{\text{cons}}$ is in XP and $\log |\mathcal{H}_{n,k}| \in \text{xp}(n; k)$, then \mathcal{L} is in XP-PAC_{time}.

The theorems above allow us to automatically transfer parameterized algorithmic upper and lower bounds for consistency checking into upper and lower bounds for parameterized learning problems, respectively. If a parameterized consistency checking problem is efficiently solvable by a parameterized algorithm, by Theorem 11 we get that the parameterized learning problem is efficiently solvable. Note that in the problems considered in this paper the solution



■ **Figure 2** For the SAT instance $\varphi = (x_1 \vee \bar{x}_2 \vee \bar{x}_3 \vee x_n) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_n)$, $n = 4$ the correspondent CONS CHECK: 2-COLORING instance $\mathcal{I} = \{V, \{(E_+, 1), (E_{C_1}, 0), (E_{C_2}, 0)\}\}$, $V = \{t, f, x_1, x_2, x_3, x_n\}$.

is always a set of vertices/edges, or a partition into such sets, thus $\log |\mathcal{H}_{n,k}|$ is always polynomial.

On the other hand, Theorem 11 tells us that an efficient algorithm for a parameterized learning problem implies an efficient algorithm for the corresponding parameterized consistency checking problem. Turning this around, we see that lower bounds on consistency checking imply lower bounds for learning. That is, if $\mathcal{P}_{\text{cons}}$ is W[1]-hard, then $\mathcal{P}_{\text{learn}}$ is not in FPT-PAC_{time} unless FPT = W[1].

3 Partitioning Problems: 2-Coloring and Split Graphs

We begin our investigation by using two basic vertex bipartition problems on graphs to showcase some of the unexpected complexity-theoretic behavior of consistency checking problems. Let us first consider 2-COLORING, i.e., the problem of partitioning the vertex set into two independent sets. There are two natural ways one can formalize 2-COLORING as a search problem: either one asks for a vertex set X such that both X and the set of vertices outside of X are independent (i.e., they form a proper 2-coloring), or one asks for two independent sets X, Y which form a bipartition of the vertex set. Here, we consider the former variant since it has a slightly smaller hypothesis space⁴.

CONS CHECK: 2-COLORING

Input: $\mathcal{I} = \{V, (E_i, \lambda_i)_{i \in [t]}\}$ where for each $i \in [t]$, $G_i = (V, E_i)$ is a graph and $\lambda_i \in \{0, 1\}$.

Output: A set $X \subseteq V$ such that for each $i \in [t]$, $(X, V \setminus X)$ forms a proper 2-coloring of G_i if and only if $\lambda_i = 1$.

As our first result, we show that CONS CHECK: 2-COLORING is NP-hard.

► **Theorem 12.** *There is no polynomial-time algorithm that solves CONS CHECK: 2-COLORING unless P = NP.*

Proof. We present a reduction that takes an n -variable instance φ of the SATISFIABILITY problem (SAT) and constructs an instance \mathcal{I} of CONS CHECK: 2-COLORING which admits a solution if and only if φ is satisfiable. Let \mathcal{C} denote the set of clauses of φ .

Construction. First, we set the vertex set V in \mathcal{I} to be $\{f, t, x_1, x_2, \dots, x_n\}$. For each clause $C \in \mathcal{C}$, we construct an edge set E_C as follows. For each $i \in [n]$, if a true (false) assignment of x_i satisfies C , then we add the edge tx_i (fx_i) to E_C . For each such edge set E_C , we set $\lambda_C = 0$. Finally, we add to \mathcal{I} a positive sample $(E_+, 1)$ such that $E_+ = \{tf\}$. An illustration is provided in Figure 2.

⁴ In general, the precise definition of the sought-after object can be of great importance in the context of consistency checking; this is related to the well-known fact that the selection of a hypothesis space can have a fundamental impact on PAC learnability. However, in our case the proofs provided in this section can also be used to obtain the same results for the latter variant.

Correctness. Suppose, given an instance φ of SAT, that the reduction described above returns $\mathcal{I} = \{V, (E_i, \lambda_i)\}_{i \in \mathcal{C} \cup \{+\}}$ as an instance of CONSCHECK: 2-COLORING.

Assume that φ admits a satisfying assignment $\mathcal{A}: \{x_i\}_{i \in [n]} \rightarrow \{\text{True}, \text{False}\}$. Consider the coloring $\chi: V \rightarrow \{\text{blue}, \text{red}\}$ such that $\chi(t) = \text{red}$, $\chi(f) = \text{blue}$, and for each $i \in [n]$, $\chi(x_i) = \text{red}$ if and only if $\mathcal{A}(x_i) = \text{True}$.

First, the sample $(E_+, 1)$ of \mathcal{I} is consistent with the coloring χ , since its only edge ft was colored properly. Then, for each $C \in \mathcal{C}$, the sample $(E_C, 0)$ must be consistent with χ , i.e., there exists at least one edge in E_C with same colored endpoints. Indeed, there must exist a variable x_i is such that $\mathcal{A}(x_i)$ satisfies φ .

Then, by the construction of \mathcal{I} instance, if $x_i = \text{True}$ ($x_i = \text{False}$) satisfies C then $x_i \in E_C$ ($fx_i \in E_C$) and hence both x_i and t are **red** (both x_i and f are **blue**) under the constructed coloring χ .

For the other direction, suppose that there is a coloring $\chi: V \rightarrow \{\text{blue}, \text{red}\}$ that is consistent with the instance \mathcal{I} of CONSCHECK: 2-COLORING. Then $\chi(t) \neq \chi(f)$ due to the construction of $(E_+, 1) \in \mathcal{I}$; without loss of generality, let $\chi(t) = \text{red}$, $\chi(f) = \text{blue}$. We retrieve a variable assignment \mathcal{A} for φ in the following way. Recall that for each $C \in \mathcal{C}$, the coloring χ is consistent with the sample $(E_C, 0)$. Since the edge ft has a proper coloring, at least one vertex x_i has an edge to either t or f such that both its endpoints are colored the same way. If this edge is $x_i f$ ($x_i t$), then let $\mathcal{A}(x_i) = \text{False}$ ($\mathcal{A}(x_i) = \text{True}$). If this only results in a partial assignment, we extend this to a complete assignment of all variables in φ by assigning the remaining variables arbitrarily.

We conclude by arguing that the resulting assignment \mathcal{A} has to satisfy φ . Let us consider an arbitrary clause $C \in \mathcal{C}$ and an edge in the corresponding edge set E_C with same colored endpoints, w.l.o.g. $x_i f$. Then, by the way we defined the assignment, $\mathcal{A}(x_i) = \text{False}$. But by our construction, the edge $x_i f \in E_C$ only if $x_i = \text{False}$ satisfies the clause C . Thus, the clause C is satisfied by the assignment \mathcal{A} . Following the same argument, each clause $C \in \mathcal{C}$, and accordingly the instance φ , is satisfied. ◀

It is worth noting that the graphs constructed by the reduction underlying Theorem 12 are very simple—in fact, even the graph induced by the union of all edges occurring in the instances of CONSCHECK: 2-COLORING produced by the reduction has a vertex cover number of 2. This essentially rules out tractability via most standard structural graph parameters. A similar observation can also be made for most other consistency checking lower bounds obtained within this article.

As an immediate corollary of Theorem 12, we obtain that the corresponding learning problem is not efficiently PAC-learnable [2]. To provide a concrete example of the formal transition from consistency checking to the corresponding learning problem described in Section 2.2, we state the problem: In 2-COLORING LEARNING, we are given (1) a set V of vertices, a confidence measure δ and an accuracy measure ε , (2) have access to an oracle that can be queried to return labeled samples of the form (E, λ) where E is an edge set over V and $\lambda \in \{0, 1\}$ according to some hidden distribution, and (3) are asked to return a vertex subset $X \subseteq V$, whereas a sample E is evaluated as positive for X if and only if $(X, V \setminus X)$ forms a 2-coloring on (V, E) .

► **Corollary 13.** 2-COLORING LEARNING is not efficiently PAC-learnable unless $\text{P} = \text{NP}$.

While the intractability of consistency checking for CONSCHECK: 2-COLORING might already be viewed as surprising, let us now consider the related problem of partitioning the vertex set into one independent set and one clique—i.e., the SPLIT GRAPH problem.

As a graph search problem, SPLIT GRAPH is well-known to be polynomially tractable [13]. Following the same line of reasoning as for 2-COLORING, we formalize the corresponding search problem below. Let a pair of vertex subsets $(X \subseteq V, Y \subseteq V)$ be a *split* in a graph $G = (V, E)$ if (X, Y) is a bipartition of V such that X is a clique and Y is an independent set.

CONSCHECK: SPLIT GRAPH

Input: $\mathcal{I} = \{V, (E_i, \lambda_i)_{i \in [t]}\}$ where for each $i \in [t]$, $G_i = (V, E_i)$ is a graph and $\lambda_i \in \{0, 1\}$.

Output: A set $X \subseteq V$ such that for each $i \in [t]$, $(X, V \setminus X)$ is a split in G_i if and only if $\lambda_i = 1$.

Unlike CONSHECK: 2-COLORING, CONSHECK: SPLIT GRAPH turns out to be tractable.

► **Theorem 14.** CONSHECK: SPLIT GRAPH *can be solved in time* $\mathcal{O}(|\mathcal{I}|^3)$.

Proof. Let us consider an input instance $\mathcal{I} = \{V, (E_i, \lambda_i)_{i \in [t]}\}$ of CONSHECK: SPLIT GRAPH. The algorithm first checks whether \mathcal{I} contains at least one positive sample or consists of negative samples only; each of these two cases will be handled by a separate procedure and arguments.

If \mathcal{I} contains at least one positive sample, say w.l.o.g. (E_1, λ_1) , then the algorithm enumerates all possible splits of $G_1 = (V, E_1)$. Indeed, for each pair $(X_1, Y_1), (X_2, Y_2)$ of splits it holds that (X_2, Y_2) can be obtained from (X_1, Y_1) by moving at most one vertex from the clique part to the independent part, and at most one vertex from the independent part to the clique part. Hence after computing an arbitrary split in linear time (e.g., from its degree sequence) [13], the algorithm can enumerate the set of all splits of G_1 in at most quadratic time. The algorithm then checks, for each split of G_1 , whether it is a solution for \mathcal{I} —in particular, whether it is a split for each positive sample and a non-split for each negative sample. This check can be done in linear time. For correctness of this case, it suffices to observe that every solution for \mathcal{I} must necessarily be a split of G_1 .

For the case where each sample in \mathcal{I} is negative, we exploit the fact that the set of all splits in a graph can be enumerated in polynomial time in a different manner. In particular, for each sample $(E_j, 0)$ in \mathcal{I} , we construct the set Q_j of all splits of $G_j = (V, E_j)$ in at most quadratic time; in particular, Q_j will be empty if G_j is not a split graph, and otherwise will contain at most a quadratic number of splits. Let $Q = \bigcup_{j \in [t]} Q_j$, and observe that $|Q| \leq |V|^2 \cdot t$. The algorithm then proceeds by enumerating, in brute force and in arbitrary order, all possible vertex 2-partitions of V , and for each such 2-partition $(X, V \setminus X)$ it checks whether $(X, V \setminus X) \in Q$ or not. If $(X, V \setminus X) \in Q$, then we proceed to the next 2-partition, while otherwise we output $(X, V \setminus X)$ as the solution; if every 2-partition turns out to be in Q then the algorithm outputs that no solution exists. Clearly, this procedure terminates after at most $|Q| + 1$ steps. For correctness, it suffices to observe that every 2-partition is a solution for \mathcal{I} if and only if it does not lie in Q —indeed, every 2-partition in Q is a split for at least one negative sample (and hence cannot be a solution), and every 2-partition that is not in Q is not a split for any negative sample and hence is a solution. ◀

Naturally, one can formalize the learning problem for CONSHECK: SPLIT GRAPH in an analogous way as was done for 2-COLORING LEARNING. Since the hypothesis bound of Theorem 11 holds here as well, Theorem 14 implies:

► **Corollary 15.** SPLIT GRAPH LEARNING *is efficiently PAC-learnable.*

Let us now conclude the section by revisiting the polynomial-time intractability of CONSHECK: 2-COLORING through the lens of parameterized complexity theory. Naturally,

there are many parameterizations one may consider in the setting—as an exercise that follows the same exhaustive-branching ideas as those used for VERTEX COVER [6, Lemma 6.1], one could for instance attempt to parameterize by the size of the smaller color class in the sought-after coloring, whereas a fixed-parameter algorithm in this setting (based on exhaustive branching) would yield a FPT-PAC_{time} algorithm for 2-COLORING LEARNING in the corresponding parameterization of the concept. In this article, we instead showcase a less straightforward fixed-parameter algorithm for the problem when parameterized by the number of negative samples on the input (which in turn corresponds to a parameterization of the distribution in the learning setting [6]). It will later turn out that the same parameterization can be used to achieve fixed-parameter tractability for several other consistency checking problems as well, albeit the individual techniques used vary from problem to problem.

Let $t^- = |\{(E_i, \lambda_i)_{i \in [t]} \mid \lambda_i = 0\}|$ be the number of negative samples in an input instance \mathcal{I} .

► **Theorem 16.** CONS_{CHECK}: 2-COLORING is fixed-parameter tractable when parameterized by the number t^- of negative samples.

Proof. Let $G^+ = (V, E^+)$ be the graph obtained from an input instance $\mathcal{I} = \{V, (E_i, \lambda_i)_{i \in [t]}\}$ of CONS_{CHECK}: 2-COLORING by setting $E^+ := \bigcup_{i \mid \lambda_i = 1} E_i$, i.e., G^+ is the (non-disjoint) union of all yes samples in \mathcal{I} . It will be useful to observe that the solution for \mathcal{I} must also be a 2-coloring for G^+ . Let $\{C_1, \dots, C_\ell\}$ be the set of connected components of G^+ ; for each connected component C_j , $j \in \ell$, we check whether C_j is bipartite. If any such component is not bipartite, we can correctly output that \mathcal{I} does not admit a solution; otherwise we assume that each C_j is bipartitioned into independent sets (A_j, B_j) and proceed with the algorithm.

Let us now consider a negative sample $(E_i, 0)$. We define the *signature* S_i of $(E_i, 0)$ as the set $\{\{X, Y\} \mid \exists xy \in E_i : (x \in X) \wedge (y \in Y) \wedge (X, Y \in \{A_1, \dots, A_\ell\} \cup \{B_1, \dots, B_\ell\})\}$; in other words, the signature is obtained by abstracting away the individual identities of the endpoints of E_i and replacing this by information about which part of which component the endpoints belong to. We remark that the signature is treated as a set of two-element multisets to accommodate for the (admittedly trivial) case where $X = Y$.

We now distinguish two cases based on the size of the signature S_i of each negative sample $(E_i, 0)$. If $|S_i| > 16(t^-)^2$ then we mark S_i as *large*, and otherwise we mark it as *small*. Observe that each large signature must contain pairs from more than $2t^-$ components of G^+ ; indeed, the size of any signature that only contains pairs from $2t^-$ components of G^+ is upper-bounded by $(2t^-)^2 \cdot 4$. We then perform exhaustive branching to identify a single pair $(X, Y) \in S_i$ in each small signature. In every branch, we proceed as follows:

1. For each pair (X, Y) identified in a small signature, we add a new degree-2 vertex to G^+ and make it adjacent to an arbitrary vertex in X and an arbitrary vertex in Y . Observe that this ensures that a proper 2-coloring of G^+ must use the same color for X and Y . We mark all connected components which have been attached to a new degree-2 vertex in this way as *used*.
2. We check if the subgraph of G^+ induced on the used connected components admits a proper 2-coloring ζ . If that is not the case, we proceed to the next branch. Otherwise, we apply the pigeon-hole principle to identify, for each large signature S_j , a pair $\{X_j, Y_j\} \in S_j$ such that either X_j or Y_j are from a component C_j^* which is not marked as used and mark this component C_j^* as used as well; this can be done via linear-time enumeration of all connected components since each large signature must contain pairs from more than $2t^-$ components of G^+ .
3. We now expand the proper 2-coloring ζ to a proper 2-coloring ν of all of G^+ as follows. We drop the coloring of the auxiliary degree-2 vertices constructed in the first step (as

these are not part of G^+). Each connected component of G^+ will be properly 2-colored, but for each pair $\{X_j, Y_j\}$ in a large signature S_j selected in the previous step we ensure that v uses the same color for X_j and Y_j . this fully determines the two color classes for all connected components of G^+ . We check that v is a solution for \mathcal{I} and output it if that is the case.

This concludes the description of the algorithm. The running time is upper-bounded by $\mathcal{O}(2^{16(t^-)^2} \cdot |\mathcal{I}|)$, since the exhaustive branching preceding the three steps listed above loops over $2^{16(t^-)^2}$ cases and the three steps can be carried out in linear time. For correctness, observe that every solution provided by the algorithm is a solution for \mathcal{I} . On the other hand, assume that \mathcal{I} admits a solution. For each negative sample $(E_i, 0)$ with a small signature, there must be at least one edge $x_i y_i \in E_i$ such that both x_i and y_i belong to the same part in the solution, and let $\{X_i, Y_i\}$ be the corresponding tuple in S_i . Consider the branch of the algorithm which selects $\{X_i, Y_i\} \in S_i$. The supergraph of G^+ constructed in this branch now admits a 2-coloring ζ . At that point the algorithm is guaranteed to succeed in finding a solution for \mathcal{I} , as the constructed 2-coloring v will be proper for all positive samples (as it is proper for G^+), will not be proper for negative samples with a small signature (as ensured already by ζ), and will not be proper for negative samples with a large signature either (as guaranteed in the final step when constructing v). ◀

4 Consistency Checking for Selected Edge Search Problems

In this section, we perform a parameterized analysis of consistency checking for three natural and extensively studied edge search problems on graphs: MATCHING, $(k-)$ PATH and EDGE CLIQUE COVER. We formalize the parameterized consistency checking formulations of these three problems below; recall that a set $\mathcal{F} = \{F_1, \dots, F_\ell\}$ is an *edge clique cover* if each F_i , $i \in [\ell]$ is the edge set of a clique in the graph and each edge in the graph is contained in at least one F_i , $i \in [\ell]$ [7, Subsection 2.2.3].

CONSCHECK: MATCHING

Input: $\mathcal{I} = \{V, (E_i, \lambda_i)_{i \in [t]}\}$ where for each $i \in [t]$, $G_i = (V, E_i)$ is a graph and $\lambda_i \in \{0, 1\}$, and an integer k .

Parameter: k .

Output: A set $F \subseteq \binom{V}{2}$ of size k such that for each $i \in [t]$, F forms a matching in G_i if and only if $\lambda_i = 1$.

CONSCHECK: $(k-)$ PATH

Input: $\mathcal{I} = \{V, (E_i, \lambda_i)_{i \in [t]}\}$ where for each $i \in [t]$, $G_i = (V, E_i)$ is a graph and $\lambda_i \in \{0, 1\}$, and an integer k .

Parameter: k .

Output: A set $F \subseteq \binom{V}{2}$ of size k such that for each $i \in [t]$, F forms a path in G_i if and only if $\lambda_i = 1$.

CONSCHECK: EDGE CLIQUE COVER

Input: $\mathcal{I} = \{V, (E_i, \lambda_i)_{i \in [t]}\}$ where for each $i \in [t]$, $G_i = (V, E_i)$ is a graph and $\lambda_i \in \{0, 1\}$, and an integer k .

Parameter: k .

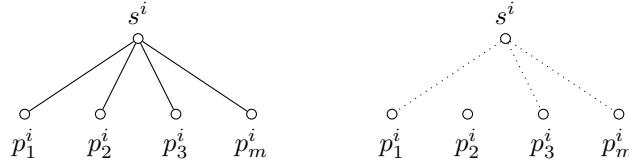
Output: A set $\mathcal{F} \subseteq 2^{\binom{V}{2}}$ of size k such that for each $i \in [t]$, \mathcal{F} forms an edge clique cover in G_i if and only if $\lambda_i = 1$.

An observant reader may notice that in the first of the three problems above, we consider solution size as a parameter even though the corresponding search problem of finding a maximum matching in a graph is polynomial-time tractable. This is due to the fact that, as it turns out, MATCHING in the consistency checking regime is not polynomial-time tractable unless $P = NP$. In fact, we show an even stronger (and more surprising) result:

► **Theorem 17.** CONSCHECK: MATCHING *does not admit a fixed-parameter algorithm unless* $FPT = W[2]$.

Proof. We present a reduction that given an instance $(\mathcal{U}, \mathcal{F}, k')$ of the classical SET COVER problem [7], constructs an instance \mathcal{I} of CONSCHECK: MATCHING which admits a solution if and only if $(\mathcal{U}, \mathcal{F}, k')$ is a yes-instance. An instance $(\mathcal{U}, \mathcal{F}, k')$ of SET COVER is a family $\mathcal{F} = \{F_1, \dots, F_m\}$ of m subsets over the n -element universe $\mathcal{U} = \{u_1, \dots, u_n\}$, and we are asked whether there exists a k' -element subset of \mathcal{F} whose union contains all of \mathcal{U} .

Construction. We construct the instance $\mathcal{I} = \{V, (E_i, \lambda_i)_{i \in [t]}\}$ of CONSCHECK: MATCHING as follows, with the parameter set to $k = k'$. Let the unique positive sample in \mathcal{I} be the edge set E_1 such that the graph (V, E_1) is a set of k disjoint stars, whereas for each $i \in [k]$ the graph (V, E_1) contains a center s^i adjacent to pendants p_1^i, \dots, p_m^i . Next, for each element $u_j \in \mathcal{U}$, $j \in [n]$, we add a negative sample (V, E_{j+1}) into \mathcal{I} which only contains non-edges between the centers of stars and the leaves (of the same star) corresponding to the sets containing that element; formally, $E_{j+1} = \binom{V}{2} \setminus \{s^i p_\ell^i \mid i \in [k], u_j \in F_\ell\}$. This completes the construction of \mathcal{I} (see also Figure 3).



■ **Figure 3** Reducing from SET COVER, the CONSCHECK: MATCHING instance has a positive sample with k ($i \in [k]$) stars as shown on the left; for each $u_j \in \mathcal{U}$, the correspondent NO-instance is a complete graph but excluding $\{s^i p_\ell^i \mid i \in [k], u_j \in F_\ell\}$. So, as an example, if $m = 4$ and $u_j \in F_\ell$ for any $\ell \in \{1, 3, m\}$, then for all $i \in [k]$, the dotted edges are out of the construction.

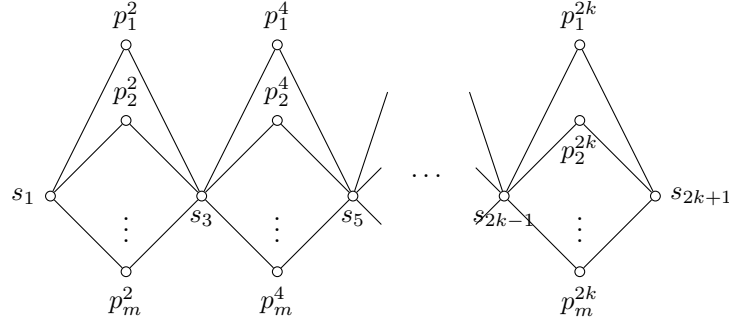
Correctness. If \mathcal{I} admits a solution Q , then Q must be a matching in (V, E_1) of size k and hence can only contain a single edge from each of the k stars. Hence, $Q = \{s^1 p_{\alpha(1)}, s^2 p_{\alpha(2)}, \dots, s^k p_{\alpha(k)}\}$ for some mapping α . Moreover, since Q is not a matching in (V, E_{j+1}) for any $j \in [n]$, the set $\{F_{\alpha(1)}, \dots, F_{\alpha(k)}\}$ is a set cover for $(\mathcal{U}, \mathcal{F}, k)$. At the same time, given a set cover $\{F_{\beta(1)}, \dots, F_{\beta(k)}\}$ (for some mapping β), we can construct a solution for \mathcal{I} by taking $\{s^1 p_{\beta(1)}, \dots, s^k p_{\beta(k)}\}$. This yields a reduction from SET COVER to the problem of deciding the existence of a solution for CONSCHECK: MATCHING; in particular, this means that a fixed-parameter algorithm for CONSCHECK: MATCHING would imply $FPT = W[2]$. ◀

A similar reduction also allows us to establish the intractability of consistency checking for PATH.

► **Theorem 18.** CONSCHECK: (k -)PATH *does not admit a fixed-parameter algorithm unless* $FPT = W[2]$.

Proof. We present a reduction that given an instance $(\mathcal{U}, \mathcal{F}, k')$ of SET COVER, constructs an instance \mathcal{I} of CONSCHECK: (k) -PATH which admits a solution if and only if $(\mathcal{U}, \mathcal{F}, k)$ is a yes-instance. Recall that an instance $(\mathcal{U}, \mathcal{F}, k)$ of SET COVER is a family $\mathcal{F} = \{F_1, \dots, F_m\}$ of m subsets over the n -element universe $\mathcal{U} = \{u_1, \dots, u_n\}$, and we are asked whether there exists a k -element subset of \mathcal{F} whose union contains all of \mathcal{U} .

Construction. We construct the instance $\mathcal{I} = \{V, (E_i, \lambda_i)_{i \in [t]}\}$ of CONSCHECK: (k) -PATH as follows, with the parameter k set to $2k'$. Let the unique positive sample in \mathcal{I} be the edge set E_1 such that the graph (V, E_1) is a $(2k+1)$ -partite graph consisting of independent sets S_1, \dots, S_{2k+1} . For each odd $j \in [2k+1]$, the set S_j contains a single vertex s_j . For each even $j \in [2k+1]$, the set S_j contains one vertex for each set in \mathcal{F} , and in particular $S_j := \{p_1^j, \dots, p_m^j\}$. E_1 is then the set of all edges connecting consecutive sets in this partition of V , i.e., $E_1 = \{s_{2j-1}p_\ell^{2j} \mid \ell \in [m], j \in [k]\} \cup \{p_\ell^{2j}s_{2j+1} \mid \ell \in [m], j \in [k]\}$ (see in Figure 4). Next, for each element $u_j \in \mathcal{U}$, $j \in [n]$, we add a negative sample (V, E_{j+1}) into \mathcal{I} which only contains non-edges incident to the sets containing that element; formally, $E_{i+1} = \binom{V}{2} \setminus \{vp_\ell^{2j} \mid j \in [k], u_i \in F_\ell\}$. This completes the construction of \mathcal{I} .



■ **Figure 4** A unique positive sample for an instance of CONSCHECK: (k) -PATH.

Correctness. If \mathcal{I} admits a solution Q , then Q must be a path in (V, E_1) of size $2k$ and hence must be a path which consecutively visits vertices $(s_1, p_{\alpha(1)}^2, s_3, p_{\alpha(2)}^4, \dots, p_{\alpha(k)}^{2k}, s_{2k+1})$. Moreover, since Q is not a path in (V, E_{j+1}) for any $j \in [n]$, the set $\{F_{\alpha(1)}, \dots, F_{\alpha(k)}\}$ is a set cover for $(\mathcal{U}, \mathcal{F}, k)$. At the same time, given a set cover $\{F_{\beta(1)}, \dots, F_{\beta(k)}\}$ (for some mapping β), we can construct a solution for \mathcal{I} by taking $(s_1, p_{\beta(1)}^2, s_3, p_{\beta(2)}^4, \dots, p_{\beta(k)}^{2k}, s_{2k+1})$. This yields a reduction from SET COVER to the problem of deciding the existence of a solution for CONSCHECK: (k) -PATH; in particular, this means that a fixed-parameter algorithm for CONSCHECK: (k) -PATH would imply FPT=W[2]. ◀

However, we show that the third problem under consideration—EDGE CLIQUE COVER—does not become more difficult in the consistency checking regime.

► **Theorem 19.** CONSCHECK: EDGE CLIQUE COVER admits a fixed-parameter algorithm which runs in time $\mathcal{O}(2^{2^k} \cdot |\mathcal{I}|)$.

Proof. We begin by observing that an edge clique cover $\mathcal{F} \subseteq 2^{\binom{V}{2}}$ for a graph (V, E_1) cannot be an edge clique cover for any graph (V, E_2) such that $E_2 \neq E_1$. Indeed, for each edge $e \in E_2 \setminus E_1$ it holds that e cannot be covered by \mathcal{F} , while every edge $e \in E_1 \setminus E_2$ must be covered by some clique $F \in \mathcal{F}$ in (V, E_1) and hence F would no longer be a clique in (V, E_2) . Hence if an instance \mathcal{I} of CONSCHECK: EDGE CLIQUE COVER contains two distinct positive samples, the algorithm can correctly output that \mathcal{I} has no solution.

So, let us consider the case where \mathcal{I} contains precisely one positive sample, say $(E_1, 1)$. There is a well-known set of simple rules that can reduce the number of vertices of (V, E_1) to 2^k [12]; in fact, under the Exponential Time Hypothesis [15] this reduction combined with a brute-force exhaustive search on the reduced instance produces an essentially optimal algorithm for EDGE CLIQUE COVER [8]. We apply this procedure to either identify, in $2^{2^k} \cdot |V|$ time, an edge clique cover \mathcal{F} of cardinality k for $(E_1, 1)$, or correctly determine that no such \mathcal{F} exists (in which case the algorithm can, as before, correctly output that \mathcal{I} has no solution). The algorithm then simply checks to ensure \mathcal{F} is not an edge clique cover for any of the negative samples, or equivalently, checks that there is no negative sample of the form $(E_1, 0)$. If this check succeeds, the algorithm outputs the solution \mathcal{F} .

Finally, for the case where \mathcal{I} contains only negative samples, let us consider $\mathcal{G} = (V, \bigcap_{i \in [t]} E_i)$. If $|V| \leq k$, we can apply exhaustive branching to check each potential choice of \mathcal{F} , resulting in an algorithm with running time 2^{2^k} . Otherwise, choose an arbitrary set of distinct vertices $v_1, \dots, v_k \in V$. Set $\mathcal{F} = \{F_1, \dots, F_k\}$ where $F_1 = \{v_1 v_2, v_1 v_3\}$, and for each F_i , $2 \leq i \leq k$ we use $F_i = \{v_i v_{i+1}\}$. \blacktriangleleft

Given the fixed-parameter intractability of CONSCHECK: MATCHING and CONSCHECK: $(k-)$ PATH w.r.t. the solution size alone, it is natural to ask whether one could solve these problems at least when the number of negative samples is small, similarly as was done in Theorem 16 for 2-COLORING. We conclude this section by answering this question positively, albeit the algorithmic techniques used here are different from Theorem 16. In fact, it turns out that an adaptation of the classical color-coding technique suffices in this case [7, Subsections 5.2 and 5.6]. For both problems, the task essentially boils down to intersecting all positive samples into one, and then looking for a solution where the set of k edges is not contained in any negative sample. After assuming that all vertices of the solution receive distinct colors, we can perform dynamic programming to find a colorful solution, and while doing so we also store information about which negative samples are already “dealt with”, i.e., which negative samples do not contain the edges in the partial solution. We provide the formal proofs of both results below.

► Theorem 20. CONSCHECK: MATCHING admits an algorithm which runs in time $2^{\mathcal{O}(k+t^-)}$. $|\mathcal{I}|^{\mathcal{O}(1)}$; in particular, the problem is fixed-parameter tractable when parameterized by $k + t^-$.

Proof. Let $(V, k, (E_1, \lambda_1), \dots, (E_t, \lambda_t))$, be the input of CONSCHECK: MATCHING; w.l.o.g. assume that $\lambda_1 = \dots = \lambda_{t^-} = 0$, and $\lambda_{t^-+1} = \dots = \lambda_t = 1$. First, consider the special case where $t^- = t$. If $|V| \geq 3$, return a set F of two vertex pairs that share a vertex, which ensures that F is not a matching in any sample. If $|V| \leq 2$, the problem is trivial.

We now have that $t^- < t$. Clearly, a set $F \subseteq \binom{V}{2}$ of size k is a solution if and only if (1) the pairs in F are vertex-disjoint, (2) for every $i \in [t^- + 1, t]$, $F \subseteq E_i$, and (3) for every $i \in [t^-]$, $F \not\subseteq E_i$. The instance is thus equivalent to $(V, k, (E_1, 0), \dots, (E_{t^-}, 0), (E, 1))$, where $E = E_{t^-+1} \cap \dots \cap E_t$, i.e., an instance obtained from the original one by intersecting all positive samples. A solution is now a matching F of size k in (V, E) such that for every $i \in [t^-]$, $F \not\subseteq E_i$.

Assume now that the vertex set V is colored in $2k$ colors with a mapping $c : V \rightarrow [2k]$, and we are looking for a colorful solution, i.e., a matching M such that in the set of vertices of M each color appears exactly once, in addition to the properties of the solution above. We show how to find such a colorful solution with the help of dynamic programming.

For $C \subseteq [2k]$, $I \subseteq [t^-]$ let $\alpha(C, I)$ be 1 if there exists a matching M in (V, E) such that its vertex set has exactly the colors in C , and such that $i \in I$ if and only if $M \not\subseteq E_i$; $\alpha(C, I) = 0$ if there is no such matching. We compute the values $\alpha(C, I)$ in the order of increasing $|C|$.

To initialize, we set $\alpha(\emptyset, \emptyset) = 1$ and $\alpha(\emptyset, I) = 0$ for all $I \subseteq [t^-]$, $I \neq \emptyset$. This is clearly correct, as the only matching whose vertex set contains no colors is the empty matching, and the empty matching is a subset of every set of edges E_i , $i \in [t^-]$.

Now consider $C \subseteq [2k]$ with $|C| > 0$, $I \subseteq [t^-]$, and assume $\alpha(C', I')$ is correctly computed for all $|C'| < |C|$ and all $I' \subseteq [t^-]$. For an edge $e \in E$, denote by $\beta(e) \subseteq [t^-]$ the set of negative samples that are “dealt with” by e , i.e., $\beta(e) = \{i \in [t^-] : e \notin E_i\}$. We set

$$\alpha(C, I) = \max_{\substack{uv \in E: c(u), c(v) \in C \\ I' \subseteq I: I' \cup \beta(uv) = I}} \alpha(C \setminus \{c(u), c(v)\}, I'). \quad (1)$$

We now argue the correctness of the computation above. First, assume $\alpha(C, I) = 1$, and consider a corresponding solution M . Let $uv \in M$ be an arbitrary edge of the solution, and let I' be the set of negative samples “dealt with” by the remaining edges of M , i.e., $I' = \bigcup_{e \in M \setminus \{uv\}} \beta(e)$. Since M is a colorful matching, the endpoints of edges in $M \setminus \{uv\}$ have colors in $C \setminus \{c(u), c(v)\}$. Therefore, $\alpha(C \setminus \{c(u), c(v)\}, I') = 1$ as $M \setminus \{uv\}$ is a suitable solution. Then also $\alpha(C, I) = 1$ by (1), since $uv \in E$, $c(u), c(v) \in C$ and $I' \cup \beta(uv) = I$, which implies that $\alpha(C \setminus \{c(u), c(v)\}, I')$ appears on the right-hand side of (1).

In the other direction, let $\alpha(C, I) = 0$, and assume for the sake of contradiction that (1) assigns 1 to it, i.e., there exists $uv \in E$ and $I' \subseteq I$ such that $c(u), c(v) \in C$, $I' \cup \beta(uv) = I$, and $\alpha(C \setminus \{c(u), c(v)\}, I') = 1$. Consider then the matching M' certifying $\alpha(C \setminus \{c(u), c(v)\}, I') = 1$, and let M be $M' \cup \{uv\}$. Clearly, M is colorful and is a matching, as the edges in M' have their endpoints' colors in $C \setminus \{c(u), c(v)\}$. Moreover, the set of colors covered by the edges of M is then exactly C , and the set of “dealt with” negative samples is exactly I , as $I = I' \cup \beta(uv)$, where the former are satisfied by M' and the latter by the edge uv . Therefore, we reach a contradiction that $\alpha(C, I)$ is 0, which finishes the proof of correctness.

Finally, a standard color-coding argument shows that solving the colorful version of the problem is sufficient, and this step could also be done in deterministic fashion with the claimed running time bound [7, Subsections 5.2 and 5.6]. Observe that the dynamic programming above contains at most $2^k \cdot 2^{t^-}$ states $\alpha(C, I)$, and each is computed in time at most $2^{t^-} \cdot n^{O(1)}$ by (1). ◀

The proof of the next theorem builds on the color-coding algorithm for k -PATH, but otherwise the arguments are fairly similar to those used in the previous theorem.

► **Theorem 21.** *CONSCHECK: (k) -PATH admits an algorithm which runs in time $2^{O(k+t^-)} \cdot |\mathcal{I}|^{O(1)}$; in particular, the problem is fixed-parameter tractable when parameterized by $k + t^-$.*

Proof. Let $(V, k, (E_1, \lambda_1), \dots, (E_t, \lambda_t))$, be the input of **CONSCHECK: (k) -PATH**; w.l.o.g. assume that $\lambda_1 = \dots = \lambda_{t^-} = 0$, and $\lambda_{t^-+1} = \dots = \lambda_t = 1$. First, assume $t^- = t$. If $|V| \geq 4$, return a set F of two vertex pairs that do not share a vertex; clearly such F is not a path in any instance and hence is a solution. On the other hand, the case where $|V| \leq 3$ is trivial.

We now have that $t^- < t$. By an analogous arguments as above, we can intersect all positive samples and arrive at the following equivalent formulation: A solution is a path P of length k in (V, E) such that for every $i \in [t^-]$, $P \not\subseteq E_i$, i.e., there is an edge in P that is not present in E_i . Here, $E = E_{t^-+1} \cap \dots \cap E_t$.

We again look for a colorful solution: assume that the vertex set V is colored in $k + 1$ colors with a mapping $c : V \rightarrow [k + 1]$, and the task is to find a path P such that in the set of vertices of P each color appears exactly once, in addition to the properties of the solution above. Next we present a dynamic programming algorithm that finds such a colorful solution.

For a vertex $v \in V$, $C \subseteq [k+1]$, $I \subseteq [t^-]$ let $\alpha(v, C, I)$ be 1 if there exists a path P in (V, E) such that its endpoint is v , its vertex set has exactly the colors in C , and such that $i \in I$ if and only if $P \not\subseteq E_i$; $\alpha(v, C, I) = 0$ if there is no such path. We compute the values $\alpha(v, C, I)$ in the order of increasing $|C|$. To initialize, for every vertex $v \in V$ we set $\alpha(v, \{c(v)\}, \emptyset) = 1$ and $\alpha(v, \{c(v)\}, I) = 0$ for all $I \subseteq [t^-]$, $I \neq \emptyset$. This is correct since every path with a single vertex is characterized by this vertex; each such path has an empty set of edges, and it is a subset of every set of edges E_i , $i \in [t^-]$.

Now consider $v \in V$, $C \subseteq [k+1]$ with $|C| > 0$, $I \subseteq [t^-]$, and assume $\alpha(v', C', I')$ is correctly computed for all $|C'| < |C|$ and all $v' \in V$, $I' \subseteq [t^-]$. For an edge $e \in E$, denote by $\beta(e)$ the set of negative samples that are “dealt with” by e , i.e., $\beta(e) = \{i \in [t^-] : e \notin E_i\}$. We set

$$\alpha(v, C, I) = \max_{\substack{uv \in E: c(u) \in C \\ I' \subseteq I: I' \cup \beta(uv) = I}} \alpha(u, C \setminus \{c(v)\}, I'), \quad (2)$$

if $c(v) \in C$; otherwise $\alpha(v, C, I)$ is clearly 0.

Next we show the correctness of the computation above. First, assume $\alpha(v, C, I) = 1$, and consider a corresponding solution P . Let $uv \in P$ be the final edge of the path, and let I' be the set of negative samples “dealt with” by the remaining edges of P , i.e., $I' = \bigcup_{e \in P \setminus \{uv\}} \beta(e)$. Since P is a colorful path, the endpoints of edges in $P \setminus \{uv\}$ have colors in $C \setminus \{c(v)\}$. Therefore, $\alpha(u, C \setminus \{c(v)\}, I') = 1$ as $P \setminus \{uv\}$ is a suitable solution. Then also $\alpha(v, C, I) = 1$ by (2), since $uv \in E$, $c(u) \in C$ and $I' \cup \beta(uv) = I$, which implies that $\alpha(u, C \setminus \{c(v)\}, I')$ appears on the right-hand side of (2).

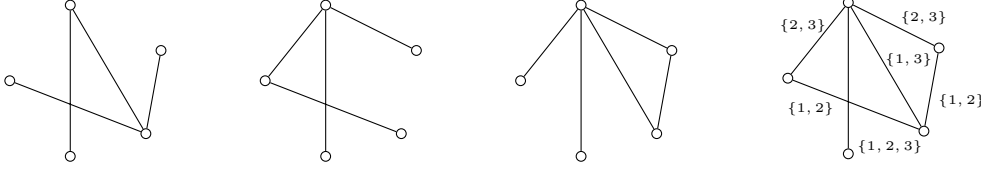
In the other direction, let $\alpha(v, C, I) = 0$, and assume for the sake of contradiction that (2) assigns 1 to it, i.e., there exists $uv \in E$ and $I' \subseteq I$ such that $c(u) \in C$, $I' \cup \beta(uv) = I$, and $\alpha(u, C \setminus \{c(v)\}, I') = 1$. Consider then the path P' certifying $\alpha(u, C \setminus \{c(v)\}, I') = 1$, and let P be $P' \cup \{uv\}$. Clearly, P is colorful and is a path, as the edges in P' have their endpoints' colors in $C \setminus \{c(v)\}$. Moreover, the set of colors covered by the edges of P is exactly C , and the set of satisfied negative samples is exactly I , as $I = I' \cup \beta(uv)$. Therefore, we reach a contradiction that $\alpha(v, C, I)$ is 0, which finishes the proof of correctness.

A standard color-coding argument also shows that solving the colorful version of the problem is sufficient, and this step could also be done in deterministic fashion in the claimed running time bound [7, Subsections 5.2 and 5.6]. \blacktriangleleft

5 Consistency Checking for Selected Vertex Search Problems

In the final technical section of this article, we focus our attention on consistency checking for two prominent vertex search problems in parameterized algorithmics: INDEPENDENT SET and DOMINATING SET. As mentioned in the introduction, both problems are believed to be fixed-parameter intractable (the former is $W[1]$ -hard while the latter is $W[2]$ -hard), and so for the purposes of this article we restrict our attention to bounded-degree input graphs—or, more precisely, we consider the maximum degree as an additional parameter⁵. We formalize the consistency checking problems below.

⁵ We remark that all of the obtained results and proofs carry over also to the case where the maximum degree is considered to be an arbitrary fixed constant



■ **Figure 5** CONSCHECK: INDEPENDENT SET[degree] instance with G_1, G_2 and G_3 ; and \mathcal{G} for the reformulation of CONSCHECK: INDEPENDENT SET[degree].

CONSCHECK: INDEPENDENT SET[degree]

Input: Integers k, d , and $\mathcal{I} = \{V, (E_i, \lambda_i)_{i \in [t]}\}$ where for each $i \in [t]$, $G_i = (V, E_i)$ is a graph of degree at most d and $\lambda_i \in \{0, 1\}$.

Parameter: $k + d$.

Output: A set $X \subseteq V$ of size k such that for each $i \in [t]$, X forms an independent set in G_i if and only if $\lambda_i = 1$.

CONSCHECK: DOMINATING SET[degree]

Input: Integers k, d , and $\mathcal{I} = \{V, (E_i, \lambda_i)_{i \in [t]}\}$ where for each $i \in [t]$, $G_i = (V, E_i)$ is a graph of degree at most d and $\lambda_i \in \{0, 1\}$.

Parameter: $k + d$.

Output: A set $X \subseteq V$ of size k such that for each $i \in [t]$, X forms a dominating set in G_i if and only if $\lambda_i = 1$.

Once again, the complexity-theoretic properties of these problems turn out to be very different from those of their simpler graph search analogues. In particular, consistency checking for INDEPENDENT SET is fundamentally harder than for the other two problems.

► **Theorem 22.** *There is no fixed-parameter algorithm for CONSCHECK: INDEPENDENT SET[degree] unless $\text{FPT} = \text{W}[2]$.*

Proof. We present a reduction that given an instance $(\mathcal{U}, \mathcal{F}, k)$ of the classical SET COVER problem [7], constructs an equivalent instance \mathcal{I} of CONSCHECK: INDEPENDENT SET[degree]. Recall that an instance $(\mathcal{U}, \mathcal{F}, k)$ of SET COVER consists of a family \mathcal{F} of m subsets over the n -element universe $\mathcal{U} = [n]$, where we are interested in at most k sets from \mathcal{F} whose union covers all of \mathcal{U} .

Before proceeding with the reduction, we present a reformulation of CONSCHECK: INDEPENDENT SET[degree] for the case where the instance \mathcal{I} only consists of negative samples.

Reformulation. Consider an instance \mathcal{I} of CONSCHECK: INDEPENDENT SET[degree] that contains negative samples only, i.e. $\mathcal{I} = \{V, (E_i, 0)_{i \in [t]}\}$ for some t ; to avoid overloading k , let us use k' to denote the first component of the parameter for \mathcal{I} .

Let us consider a graph \mathcal{G} over the vertex set V and with the edge set $E(\mathcal{G}) = \bigcup_{i=1}^t E(G_i)$. Now, for each edge $e \in E(\mathcal{G})$, we assign a label set $\mathcal{L}_e = \{i \mid e \in E(G_i)\}$. In other words, for each edge $e \in E(\mathcal{G})$ we remember the samples e originates from.

Now, our aim is to select a set $H \subseteq V(\mathcal{G})$ of k' vertices such that the union of all label-sets of edges between the vertices of H is exactly $[t]$; formally, we seek a set H of k' vertices such that $\bigcup_{v, u \in H} \mathcal{L}_{uv} = [t]$. Note that such a set $H \subseteq V(\mathcal{G})$ guarantees that, for each $i \in [t]$, there exist $u, v \in H$ such that $i \in \mathcal{L}_{uv}$; the latter, by the definition of the label-set, means that $uv \in G_i$. Thus, the set H is not an independent set for any of the graphs $G_i = (V, E_i)$, $i \in [t]$. Conversely, if for each $i \in [t]$ there are $u, v \in H$ such that $vu \in E_i$, then $uv \in \mathcal{L}_{vu}$, and a union of the label-sets for each pair $u, v \in H$ then gives $[t]$.

Construction. Now, let us consider an instance $(\mathcal{U}, \mathcal{F}, k)$ of SET COVER. We construct an instance (G, k') of the reformulated CONSCHECK: INDEPENDENT SET[degree] problem described above, with $t = n$ being the size of the universe. The graph \mathcal{G} is constructed as follows. For each $F_i \in \mathcal{F}$, we introduce two vertices f_i^1, f_i^2 , and add an edge f_i with the set F_i as its label-set \mathcal{L}_{f_i} ; $k' = 2k$.

Correctness. Suppose, given an instance $(\mathcal{U}, \mathcal{F}, k)$ of SET COVER, that the reduction returns (G, k') as an instance of the reformulated CONSCHECK: INDEPENDENT SET[degree].

Assume that $(\mathcal{U}, \mathcal{F}, k)$ admits $S \subseteq \mathcal{F}$ such that both $\bigcup_{F \in S} F = \mathcal{U}$ and $|S| \leq k$ hold. Then, let us construct a solution $H \subseteq V(G)$ for (G, k') . For each $F \in S$, let us include both f_i^1 and f_i^2 to H . Then, obviously, the condition on the set's cardinality will hold, i.e., $|H| \leq k' = 2k$. Also, $\bigcup_{v, u \in H} \mathcal{L}_{uv} = [n]$ since the label-sets are exactly the sets used in the solution S for $(\mathcal{U}, \mathcal{F}, k)$ and $[n]$ is the universe.

For the other direction, suppose that we have $H \subseteq V(G)$ such that $\bigcup_{v, u \in H} \mathcal{L}_{uv} = [n]$. Observe that $\mathcal{L}_{uv} \neq \emptyset$ only for pairs of the form f_i^1, f_i^2 for some $i \in [m]$. So, let $S \subseteq \mathcal{F}$ be defined as follows: $S = \{F_i \mid \{f_i^1, f_i^2\} \subseteq H\}$. Then, by the above construction, the sets in S are identical to the label-sets of edges whose both endpoints are in H . And, since the union of the label-sets of edges whose both endpoints are in H are $[n]$, the selected S is a solution for the considered instance of SET COVER, i.e., $\bigcup_{F \in S} F = \mathcal{U} = [n]$. ◀

► **Theorem 23.** CONSCHECK: DOMINATING SET[degree] can be solved by a fixed-parameter algorithm running in time $\mathcal{O}(2^{kd}) \cdot |\mathcal{I}|$.

Proof. We show that CONSCHECK: DOMINATING SET[degree] admits a kernel of size $k(d+1)$.

Indeed, each vertex in the solution dominates at most d vertices outside of the solution. Thus, if $|V| > k \cdot (d+1)$ and CONSCHECK: DOMINATING SET[degree] contains a positive sample, we can immediately output that there is no solution, while if $|V| > k \cdot (d+1)$ and CONSCHECK: DOMINATING SET[degree] contains negative samples only, we can correctly output an arbitrary set of k vertices as a solution.

At this point, it remains to deal with instances of CONSCHECK: DOMINATING SET[degree] such that $|V| \leq k \cdot (d+1)$. To deal with these, it suffices iterate over all subsets of V and check for each such subset whether it is a solution for CONSCHECK: DOMINATING SET[degree].

Since each such check can be carried out in linear time, in $\mathcal{O}(2^{kd}) \cdot |\mathcal{I}|$ we either find a solution for CONSCHECK: DOMINATING SET[degree] or correctly identify that no such solution exists. ◀

Similarly to Theorems 16, 20 and 21, we turn our attention to whether the lower bound for CONSCHECK: INDEPENDENT SET[degree] can be overcome if the number of negative samples is bounded by the parameter. While the $W[2]$ -hardness reduction of Theorem 22 does not hold if we are given a bound on the number of samples, it turns out that—unlike for 2-COLORING, MATCHING and (k) -PATH—consistency checking for INDEPENDENT SET remains fixed-parameter intractable even under this additional restriction.

► **Theorem 24.** There is no fixed-parameter algorithm for CONSCHECK: INDEPENDENT SET[degree] even when the number t^- of negative samples is assumed to be an additional parameter, unless $\text{FPT} = W[1]$.

Proof. We present a simple reduction that given an instance (G, k') of the classical INDEPENDENT SET decision problems on graphs, constructs an equivalent instance \mathcal{I} of CONSCHECK:

INDEPENDENT SET[degree] where each graph (V, E_i) has maximum degree 1, $t^- = 0$, and $k = k'$.

Construction. We construct the instance $\mathcal{I} = \{(V, (E_i, \lambda_i)_{i \in [t]})\}$ where V is the set of vertices in the instance (G, k') of INDEPENDENT SET, and for each edge e_j in G the set (E_i, λ_i) contains a tuple $(\{e_j\}, 1)$. Notice that t is then the number of edges in G .

Correctness. Suppose, given an instance (G, k') of INDEPENDENT SET, that the reduction described above returns $\mathcal{I} = \{(V, (E_i, \lambda_i)_{i \in [t]})\}$ as an instance of CONSCHECK: INDEPENDENT SET[degree]. Then every independent set in G is also an independent set for each of the positive samples in \mathcal{I} , and at the same time an independent set on V that is independent for each of the positive samples in \mathcal{I} is also an independent set in G . Hence, the existence of a fixed-parameter algorithm that solves every instance \mathcal{I} obtained in this way parameterized by k plus the number of negative samples plus the maximum degree of a sample would imply a fixed-parameter algorithm for INDEPENDENT SET. ◀

While restricting the number of negative samples alone is insufficient to achieve tractability, we conclude by showing that restricting the total number of samples allows for a fixed-parameter algorithm that solves the problem via a combination of multi-step exhaustive branching and color coding.

► **Theorem 25.** CONSCHECK: INDEPENDENT SET[degree] admits an algorithm which runs in time $(kdt)^{\mathcal{O}(k^2)} \cdot n^{\mathcal{O}(1)}$; in particular, it is fixed-parameter tractable when parameterized by $k + d + t$.

Proof. Consider an input instance $\mathcal{I} = \{(V, (E_i, \lambda_i)_{i \in [t]})\}$, k, d of CONSCHECK: INDEPENDENT SET[degree], and recall that the solution is a vertex set of size k . Let us denote the negative samples in \mathcal{I} as $(E_1^-, 0), \dots, (E_{t^-}^-, 0)$, where $t^- \leq t$ is the number of negative samples in the instance. The algorithm begins by exhaustively branching over all k -vertex graphs and all possible labelings of the edges of these graphs by subsets of $[t^-]$. We call each such graph considered in a separate branch in the algorithm a *template*, and we note that the number of templates is upper-bounded by 2^{k^2} . We immediately discard templates such that there exists a label $z \in [t^-]$ which does not occur on any of its edges.

Intuitively, a template captures the behavior of a hypothetical solution with respect to the negative samples. Indeed, observe that for every hypothetical k -vertex solution S of \mathcal{I} , we can construct a template T_S as follows:

- the vertices of T_S are mapped to S by an arbitrary bijection;
- whenever two vertices $s, t \in S$ are not adjacent to each other in any negative sample, we keep their counterparts non-adjacent in T_S ; and
- whenever two vertices $s, t \in S$ are adjacent to each other in negative samples $\{E_z \mid z \in Z\}$ for some $Z \subseteq [t^-]$, we place an edge between their counterparts in T_S and label that edge by Z .

Second, in each branch where we have a fixed template T , we apply the color-coding technique with derandomization [7, Subsections 5.2 and 5.6] to construct a family \mathcal{B} of k -colorings of V such that if there exists a solution $S = \{s_1, \dots, s_k\}$, then there will exist a coloring $B \in \mathcal{B}$ such that each $s_i \in S$ will receive the color i . We then branch over all colorings in \mathcal{B} , and the running time required for this branching step is upper-bounded by $(2e)^k \cdot k^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ [7, Subsection 5.6].

In the third step, the algorithm exploits the degree bound d to enumerate all appropriately colored isomorphic copies of each of the connected components in T which form independent

sets in the positive instances. More precisely, for each connected component $C \in T$ consisting of vertices $s_{\alpha(1)}, \dots, s_{\alpha(\ell)}$ for some $\ell \leq k$, the algorithm branches over all of the at most $|V|$ many choices of $s'_{\alpha(1)} \in V$ which received color $\alpha(1)$. It then chooses a new vertex in C which is adjacent to $s_{\alpha(1)}$, say $s_{\alpha(i)}$, via an edge labeled by some edge-label set Z . To identify $s'_{\alpha(i)} \in V$, it chooses an arbitrary label $z \in Z$ and then branches over the at most d many neighbors of $s'_{\alpha(1)}$ in E_z^- ; for each such neighbor v , the algorithm tests whether v is independent from $s'_{\alpha(1)}$ in all positive instances and whether v is adjacent to $s'_{\alpha(1)}$ precisely in those negative instances whose indices are in Z . This branching procedure requires time at most $n \cdot d^{\ell-1} \leq n \cdot d^k$ and allows us to construct the set L_C of all ℓ -vertex subsets which are (1) independent in the positive samples, (2) colored in the same way as C , and (3) correspond to C in the way described in the second paragraph of the proof.

In the final fourth step of the algorithm, assume we have constructed the sets L_{C_1}, \dots, L_{C_p} for the connected components C_1, \dots, C_j of T . For each *small* set L_{C_j} , $j \in [p]$, such that $|L_{C_j}| \leq \sum_{i \in [k]} \binom{kdt \cdot (dt)^k}{i}$, we perform exhaustive branching to choose a subset $S_j \in L_{C_j}$ and add it to a solution set S' . On the other hand, *large* sets L_{C_j} such that $|L_{C_j}| > \sum_{i \in [k]} \binom{kdt \cdot (dt)^k}{i}$ are ignored; the justification for this will become clear in the correctness argument, whereas the intuition is that in this case we are guaranteed to find a vertex subset in L_{C_j} which will be independent from whichever other vertices are chosen to be part of the solution. Finally, we check whether the set S' constructed in this branch is consistent with all positive samples; in particular, it is necessary to check that vertices originating from different components of T are independent in all positive samples. If this test fails, the algorithm proceeds to the next branch. If S' succeeds with this final test, the algorithm searches each large set L_{C_j} until it finds an arbitrary set $S_j \in L_{C_j}$ which is independent from S' in all positive samples, and adds S_j to S' . It then outputs the constructed set S' .

The running time of the algorithm described above is upper-bounded by $n^{\mathcal{O}(1)} \cdot (kd^k t^k)^{\mathcal{O}(k)}$. For correctness, let us assume the existence of a hypothetical solution S and let T_S be a template which corresponds to S as described in the second paragraph of the proof. Consider the branch in which the algorithm considers the template T_S , and then a branch in which it selects a color family $B \in \mathcal{B}$ such that each vertex in S receives a unique color. In the third step, the algorithm will construct the sets L_{C_1}, \dots, L_{C_p} for each of the p connected components of T_S ; in particular, there exists some $S_1 \in L_{C_1}, \dots, S_p \in L_{C_p}$ such that $S = \bigcup_{\ell \in [p]} S_\ell$. For the final branching step, let us consider the branch in which the algorithm correctly identifies those subsets S_j , $j \in [p]$, such that $S_j \subseteq S$ for each small L_{C_j} .

To complete the proof, it remains to argue that the algorithm will extend the set $S_0 \subseteq S$ constructed so far into some k -vertex solution for \mathcal{I} . To this end, notice that since $|S_0| \leq k$, there are at most kdt many vertices that are adjacent to at least one vertex in S_0 in at least one **positive** sample; let us denote this vertex set M . Moreover, there are at most $kdt \cdot (dt)^k$ vertices in the distance- k neighborhood of M in the graph $(V, \bigcup_{u \in [t^-]} E_u^-)$ of all **negative** samples; let us denote the vertices in this distance- k neighborhood by M^+ . The total number of vertex subsets of size at most k in M^+ is upper-bounded by $\sum_{i \in [k]} \binom{kdt \cdot (dt)^k}{i}$. This means that each large L_{C_j} must contain at least one set, say S_j^* , which is not fully contained in M^+ and in particular contains at least one vertex outside of M^+ . And since S_j^* is connected and contains at most k vertices, it must be completely disjoint from M . Hence S_j^* must be non-adjacent to S_0 . This guarantees that the algorithm will discover at least one set in the first large L_{C_j} which can be added to its constructed set S_0 . Crucially, since the only property of S_0 that was used in this argument was that $|S_0| \leq k$, it can be repeated in verbatim for every other large L_{C_j} . In summary the algorithm is guaranteed to output a set $S' \supseteq S_0$ which is a k -vertex solution for \mathcal{I} , as desired.



6 Concluding Remarks

This article can be seen as a “brief expedition into the forgotten island of consistency checking”—a place where SPLIT GRAPH and EDGE CLIQUE COVER are tractable but 2-COLORING and MATCHING are not, and where on bounded-degree graphs INDEPENDENT SET is $W[2]$ -hard while DOMINATING SET admits a fixed-parameter algorithm.

To conclude on a more serious note, we remark that our understanding of parameterized consistency checking—and, more broadly, of sample complexity—is still in its infancy. Even in the setting of PAC learning considered here, we so far know very little about which learning problems belong to the classes FPT-PAC and XP-PAC. Still, we hope that the results and techniques presented in this article can contribute to bridging the gap between the parameterized (time) complexity and the sample complexity research fields. A natural target for future work in this direction would be to further deepen our understanding of problems such as learning CNF and DNF formulas [20, 2, 6] or juntas [18].

References

- 1 Sushant Agarwal, Nivasini Ananthkrishnan, Shai Ben-David, Tosca Lechner, and Ruth Uerner. Open problem: Are all VC-classes CPAC learnable? In Mikhail Belkin and Samory Kpotufe, editors, *Conference on Learning Theory, COLT 2021, 15-19 August 2021, Boulder, Colorado, USA*, volume 134 of *Proceedings of Machine Learning Research*, pages 4636–4641. PMLR, 2021. URL: <http://proceedings.mlr.press/v134/open-problem-agarwal21b.html>.
- 2 Michael Alekhnovich, Mark Braverman, Vitaly Feldman, Adam R. Klivans, and Toniann Pitassi. The complexity of properly learning simple concept classes. *J. Comput. Syst. Sci.*, 74(1):16–34, 2008. doi:10.1016/j.jcss.2007.04.011.
- 3 Vikraman Arvind, Johannes Köbler, and Wolfgang Lindner. Parameterized learnability of juntas. *Theor. Comput. Sci.*, 410(47-49):4928–4936, 2009. doi:10.1016/j.tcs.2009.07.003.
- 4 Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *J. ACM*, 36(4):929–965, 1989. doi:10.1145/76359.76371.
- 5 Olivier Bousquet, Steve Hanneke, Shay Moran, and Nikita Zhivotovskiy. Proper learning, helly number, and an optimal SVM bound. In Jacob D. Abernethy and Shivani Agarwal, editors, *Conference on Learning Theory, COLT 2020, 9-12 July 2020, Virtual Event [Graz, Austria]*, volume 125 of *Proceedings of Machine Learning Research*, pages 582–609. PMLR, 2020. URL: <http://proceedings.mlr.press/v125/bousquet20a.html>.
- 6 Cornelius Brand, Robert Ganian, and Kirill Simonov. A parameterized theory of PAC learning. In *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023*. AAAI Press, 2023. to appear. URL: <https://arxiv.org/abs/2304.14058>.
- 7 M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshantov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015.
- 8 Marek Cygan, Marcin Pilipczuk, and Michał Pilipczuk. Known algorithms for edge clique cover are probably optimal. *SIAM Journal on Computing*, 45(1):67–83, 2016. doi:10.1137/130947076.
- 9 Peter Damaschke and Azam Sheikh Muhammad. Competitive group testing and learning hidden vertex covers with minimum adaptivity. *Discret. Math. Algorithms Appl.*, 2(3):291–312, 2010. doi:10.1142/S1793833091000067X.
- 10 Reinhard Diestel. *Graph Theory*. Graduate Texts in Mathematics. Springer, Berlin, Heidelberg, 5th edition, 2017. doi:10.1007/978-3-662-53622-3.

- 11 Eduard Eiben, Sebastian Ordyniak, Giacomo Paesani, and Stefan Szeider. Learning small decision trees with large domain. In *The 32nd International Joint Conference on Artificial Intelligence (IJCAI-23), August 19–25, 2023, Macao, S.A.R.* International Joint Conferences on Artificial Intelligence Organization, 2023. to appear.
- 12 Jens Gramm, Jiong Guo, Falk Hüffner, and Rolf Niedermeier. Data reduction and exact algorithms for clique cover. *ACM Journal of Experimental Algorithmics*, 13, 09 2008. doi: 10.1145/1412228.1412236.
- 13 Peter L. Hammer and Bruno Simeone. The splittance of a graph. *Comb.*, 1(3):275–284, 1981. doi:10.1007/BF02579333.
- 14 Steve Hanneke. The optimal sample complexity of PAC learning. *J. Mach. Learn. Res.*, 17:38:1–38:15, 2016. URL: <http://jmlr.org/papers/v17/15-389.html>.
- 15 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.
- 16 M. Kearns and U. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, 1994.
- 17 Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. Adaptive computation and machine learning. MIT Press, 2012. URL: <http://mitpress.mit.edu/books/foundations-machine-learning-0>.
- 18 Elchanan Mossel, Ryan O’Donnell, and Rocco A. Servedio. Learning juntas. In Lawrence L. Larmore and Michel X. Goemans, editors, *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9-11, 2003, San Diego, CA, USA*, pages 206–212. ACM, 2003. doi:10.1145/780542.780574.
- 19 Sebastian Ordyniak and Stefan Szeider. Parameterized complexity of small decision tree learning. In *Proceeding of AAAI-21, the Thirty-Fifth AAAI Conference on Artificial Intelligence*, pages 6454–6462. AAAI Press, 2021. URL: <https://ojs.aaai.org/index.php/AAAI/article/view/16800>.
- 20 Leonard Pitt and Leslie G. Valiant. Computational limitations on learning from examples. *J. ACM*, 35(4):965–984, 1988. doi:10.1145/48014.63140.
- 21 Oleg Sheyner and Jeannette M. Wing. Tools for generating and analyzing attack graphs. In *FMCO*, volume 3188 of *Lecture Notes in Computer Science*, pages 344–372. Springer, 2003. URL: <https://www.cs.cmu.edu/~scenariograph/sheynerwing04.pdf>.
- 22 L. G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, 1984. doi: 10.1145/1968.1972.
- 23 Steffen van Bergerem, Martin Grohe, and Martin Ritzert. On the parameterized complexity of learning first-order logic. In *Proceedings of the 41st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS ’22*, page 337–346, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3517804.3524151.
- 24 Jeannette M. Wing. *Attack graph generation and analysis*. New York, NY, USA, 2006. Association for Computing Machinery. doi:10.1145/1128817.1128822.