Complexity of Independency and Cliquy Trees

Katrin Casel[a], Jan Dreier[c], Henning Fernau[a], Moritz Gobbert[a], Philipp Kuinke[c], Fernando Sánchez Villaamil[c], Markus L. Schmid[a], Erik Jan van Leeuwen[b]

[a] *Fachbereich 4 – Abteilung Informatikwissenschaften, CIRT, Universität Trier, 54286 Trier, Germany,* {casel,fernau,gobbert,mschmid}@uni-trier.de; Katrin.Casel@outlook.de
[b] *Departement of Information and Computing Sciences, Utrecht University, PO Box 80.089, 3508 TB Utrecht, The Netherlands,* e.j.vanleeuwen@uu.nl
[c] *Lehr- und Forschungsgebiet Theoretische Informatik, RWTH Aachen, 52074 Aachen, Germany,* {dreier,kuinke,fernando.sanchez}@cs.rwth-aachen.de

Abstract

An independency (cliquy) tree of an n-vertex graph G is a spanning tree of G in which the set of leaves induces an independent set (clique). We study the problems of minimizing or maximizing the number of leaves of such trees, and fully characterize their parameterized complexity. We show that all four variants of deciding if an independency/cliquy tree with at least/most ℓ leaves exists parameterized by ℓ are either Para-NP- or W[1]-hard. We prove that minimizing the number of leaves of a cliquy tree parameterized by the number of internal vertices is Para-NP-hard too. However, we show that minimizing the number of leaves of an independency tree parameterized by the number k of internal vertices has an $O^*(4^k)$-time algorithm and a $2k$ vertex kernel. Moreover, we prove that maximizing the number of leaves of an independency/cliquy tree parameterized by the number k of internal vertices both have an $O^*(18^k)$-time algorithm and an $O(k\,2^k)$ vertex kernel, but no polynomial kernel unless the polynomial hierarchy collapses to the third level. Finally, we present an $O(3^n \cdot f(n))$-time algorithm to find a spanning tree where the leaf set has a property that can be decided in $f(n)$ time and has minimum or maximum size.

1. Introduction

The well-known notion of a spanning tree has inspired the study of many variants [3, 14], such as Steiner trees and connected vertex covers. Fitting in this broad line of research, this paper considers two variants of a spanning tree obtained by restricting the structure of the leaves of the tree. The first considered variant is an *independency tree*[1]: a spanning tree in a graph G where the set L of its leaves is an independent set in G, i.e., the subgraph of G induced by L is edgeless. Independency trees were introduced by Böhme et al. [2] for two different graph-theoretic reasons. First, these trees block a certain local-search heuristic from finding Hamiltonian cycles in graphs. Second, they

[1] This notion of independency trees should not be confused with the one introduced by Gutin [16].

allow a(nother) proof of Brooks' theorem on the chromatic number of bounded-degree graphs. Böhme et al. [2] characterized the n-vertex graphs that admit an independency tree as those not isomorphic to the complete graph K_n, the cycle C_n, or (if n is even) the complete bipartite graph $K_{\frac{n}{2},\frac{n}{2}}$. Eppstein and Le [18] characterized the graphs in which every spanning tree is an independency tree as those in which the set of any two adjacent vertices of degree more than one is a separator.

The second considered variant –a natural antonym to independency trees– is a *cliquy tree*: a spanning tree in a graph G where the set L of its leaves is a clique in G, i.e., the subgraph of G induced by L is complete. It seems that the origin of this variant are discussions on the well-known scientific Internet forums *stackexchange* [20] and *mathoverflow* [19], where cliquy trees were introduced under the name *completeness trees* by Le [20]. Moreover, Le also characterized the n-vertex graphs in which every spanning tree is a cliquy tree as those isomorphic to the complete graph K_n or the cycle C_n [19]. Eppstein observed that in a triangle-free graph, a graph has a cliquy tree if and only if it admits a Hamiltonian cycle [20].

Only basic facts are known about the complexity of natural computational problems surrounding independency and cliquy trees. The characterization by Böhme et al. [2] implies a polynomial-time algorithm to decide whether a graph admits an independency tree, and if so, find one through a depth-first search. In contrast, the problem to decide whether a graph admits a cliquy tree is NP-hard, as shown by Eppstein [20]: combine the aforementioned observation that in a triangle-free graph, a graph admits a cliquy tree if and only if it admits a Hamiltonian cycle, with the fact that Hamiltonian Cycle is NP-hard on triangle-free graphs [22, Theorem 2.1]. The Min Leaf IT problem, aiming to minimize the number of leaves of an independency tree, is clearly NP-hard by reduction from Hamiltonian Path [2]. Prior to this paper, it was unknown whether the maximization variant Max Leaf IT was NP-hard. The corresponding minimization and maximization problems for cliquy trees, Min Leaf CT and Max Leaf CT, are NP-hard by the aforementioned result by Eppstein [20].

We are not aware of any further studies investigating the computational complexity of these problems, neither from the point of view of parameterized nor of approximation algorithms.

Results. In this paper, we initiate the study of Min Leaf IT, Max Leaf IT, Min Leaf CT, and Max Leaf CT from a parameterized point of view. We consider two natural parameters: the number ℓ of leaves of the independency or cliquy tree, and the number k of internal vertices. We completely settle the parameterized complexity each of the four problems under each of the two parameterizations.

The main results of the paper are summarized in Table 1. Here, the problem Min Leaf IT with parameter ℓ, written as Min Leaf IT (leaf), should be interpreted as the problem to decide whether a graph G has an independency tree with at most ℓ leaves. The problem Min Leaf IT with parameter k, written as Min Leaf IT (internal), should be interpreted as the problem to minimize the number of leaves of an independency tree with at least k internal vertices. Similar definitions apply with respect to the other three problems.

	$\ell = \#$leaves	$k = \#$internal vertices		exact
		algorithm	kernel, $\#$vertices	algorithm
Min Leaf IT	Para-NP-hard	$O^*(4^k)$	$2k$ kernel	$O^*(3^n)$
Max Leaf IT	W[1]-hard	$O^*(18^k)$	$O(k\,2^k)$, no poly kernel	$O^*(3^n)$
Min Leaf CT	Para-NP-hard	Para-NP-hard		$O^*(3^n)$
Max Leaf CT	Para-NP-hard	$O^*(18^k)$	$O(k\,2^k)$, no poly kernel	$O^*(3^n)$

Table 1: Summary of the results in this paper. The problem definitions are explained in the text. The results can be found in Sections 3, 4, and 5.

We also give an $O^*(3^n)$ time exact algorithm for a generalization of all considered problems, which implies the results in the final column of Table 1.

Theorem 1. *Let G be a graph of order n. There exists an algorithm running in time $O(3^n \cdot f(n))$ and space $O(3^n)$ that constructs a spanning tree of G with a maximal/minimal number of leaves, where the leaves satisfy a property P that can be checked in time $f(n)$.*

We also argue that no subexponential-time parameterized or exact algorithms exist for any of the studied problems, unless the Exponential Time Hypothesis fails.

Comparison to Other Spanning Tree Variants and Further Results. It seems natural to compare the problems studied in this paper to other spanning tree problems, such as Max-Leaf Spanning Tree and Max-Internal Spanning Tree. The Max-Internal Spanning Tree problem asks to decide if a graph has a spanning tree with at least k internal vertices [1, 23, 24, 28, 29, 30]. Note that Min Leaf IT (internal) and Min Leaf CT (internal) can be seen as the variant of this problem where the leaves of the spanning tree should induce an independent set respectively a clique. Max-Leaf Spanning Tree asks to decide if a graph has a spanning tree with at least k leaves [1, 4, 7]. Similarly, Max Leaf IT (leaf) and Max Leaf CT (leaf) can be seen as variants of this problem. Both Max-Internal Spanning Tree and Max-Leaf Spanning Tree are fixed-parameter tractable with respect to their standard parameter; see [1, 7].

Another natural notion to compare independency trees to is connected vertex cover. A *connected vertex cover* of a graph is a connected subgraph whose complement is an independent set. The internal vertices of an independency tree form a connected vertex cover of the graph. Every connected graph admits a connected vertex cover, but there exist connected graphs without an independency tree [2]. Even in connected graphs that do admit an independency tree, not every connected vertex cover induces an independency tree: add a pendant vertex to an $n-1$ vertex cycle to obtain a graph that has two independency trees and $O(n)$ connected vertex covers. Hence, these notions seem incomparable and their algorithms not interchangeable.

Salamon [30], however, conjectured the following connection, which we disprove in Section 2.

Conjecture 1 (Salamon [30])**.** *For any graph G with an independency tree and with a minimum connected vertex cover of size c, there is an independency tree with at most c internal vertices.*

Theorem 2. *Conjecture 1 is false.*

Preliminaries. In this work we will use common graph theory notation [9]. We do not introduce new notation and readers of previous work should feel familiar with the way the results are presented. We only consider undirected, finite, simple graphs. For a graph G let $V(G)$ be the vertex set and $E(G)$ the edge set. For $v \in V(G)$ we denote the open and closed neighborhoods of v by $N_G(v)$ and $N_G[v]$, respectively (or simply by $N(v)$ and $N[v]$, if G is clear from the context). For $X \subseteq V$ we write $N(X)$ for $\bigcup_{v \in X} N(v)$. The degree of a vertex v in graph G is denoted by $\deg(v)$. We write $G' \subseteq G$ if G' is a subgraph of G. For $X \subseteq V(G)$ we denote by $G[X]$ the subgraph of G induced by X. The graph $G[V(G) - X]$ is denoted by $G - X$. A vertex of degree one is called a *leaf*. The *size of a tree* is its number of vertices. If v is a non-leaf vertex in a tree we call it an *internal vertex* of the tree. A *complete matching* from A to B is a matching that matches every vertex in A with a vertex in B. We denote a complete graph on n vertices by K_n and a complete bipartite graph with independent sets of size n and m by $K_{n,m}$. Moreover, C_n denotes a cycle on n vertices.

We use the concepts of classical and parameterized complexity in the common way and any reader with the respective background should be able to follow our technical statements (for further information, we refer to the textbooks [6, 12, 17]). In order to prove Para-NP-hardness, we use the fact that a parameterized problem is Para-NP-hard if and only if the union of finitely many of its slices[2] is NP-hard [17, Theorem 2.14].

2. Relation to Connected Vertex Cover

Connected Vertex Cover (Dominating Set) asks for a given graph $G = (V, E)$ and an integer k if there exists a subset C of V of cardinality at most k such that C is a vertex cover (dominating set) in G and such that $G[C]$ is connected. The internal vertices of a spanning tree are a connected dominating set. It is known that if d is the size of the smallest connected dominating set and ℓ the leaf number of the max-leaf spanning tree, we have $n = d + \ell$, therefore Connected Dominating Set and Max-Leaf Spanning Tree are equivalent [11]. If one forces the leaf set to be independent, we enforce that the internal vertices are not only a dominating set but also a vertex cover. It has been conjectured that the analogous version of the above relationship still holds; that is, if c is the size of the smallest connected vertex cover and ℓ the leaf number of the max-leaf independency tree it holds that $n = c+\ell$ [30, Conjecture 4.10, Corollary 4.13]; see Conjecture 1.

We show that this conjecture is not true by constructing a graph Γ_d, that has an even stronger property in that the independency tree with the maximum number of leaves does not even have a *minimal* connected vertex cover

[2]For a constant c, the c^{th} slice of a parameterized problem is the classical decision problem containing all positive instances, where the parameter equals c.

as its internal vertices and thus also rules out potential weaker formulations of Conjecture 1. Theorem 2 follows as a corollary from the following theorem:

Theorem 3. *For any integer $d > 0$ there is a graph Γ_d with $n = 43d+2$ vertices that has an independency tree and every independency tree with a maximum number of leaves has $d = O(n)$ internal vertices which have to be removed from the set of internal vertices to form a minimal connected vertex cover.*

Construction of Γ_d First, for each integer $i, j \geq 1$, let $H_{i,j}$ be the graph with fourteen vertices depicted in Figure 1.

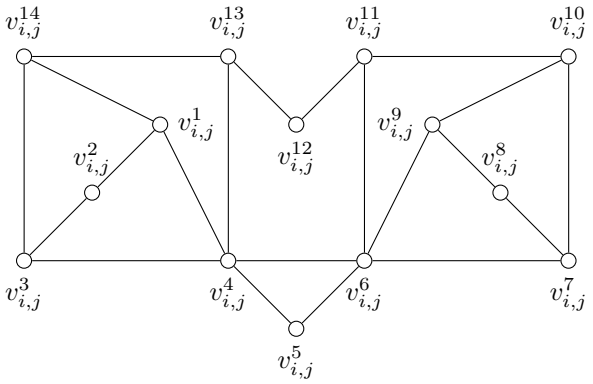

Figure 1: The graph $H_{i,j}$.

Next, for each $i \geq 1$, let X_i (see Figure 2) be the 43-vertex graph obtained from $H_{i,1}, H_{i,2}$ and $H_{i,3}$ by adding a new vertex v_i and the following 9 new edges:

- $v_i v^{11}_{i,j}$, $v_i v^{13}_{i,j}$ for $j = 1, 2, 3$, and
- $v^6_{i,1} v^4_{i,2}$, $v^6_{i,2} v^4_{i,3}$, $v^6_{i,3} v^4_{i,1}$.

Now, Γ_d (see Figure 3) is obtained from $X_1, \ldots, X_d$ by adding two new vertices v and w and the following $2d + 2$ new edges:

- vw;
- vv_i, $1 \leq i \leq d$;
- $vv^4_{1,1}$, $vv^6_{d,3}$;
- $v^6_{i,3} v^4_{i+1,1}$, $1 \leq i < d$.

Note that Γ_d has $43d + 2$ vertices (and it can be seen that Γ_d is planar).

Fact 1. *Γ_d has an independency tree. An independency tree with $18d+1$ leaves $v^2_{i,j}$, $v^5_{i,j}$, $v^8_{i,j}$, $v^{10}_{i,j}$, $v^{12}_{i,j}$, $v^{14}_{i,j}$, $1 \leq i \leq d, 1 \leq j \leq 3$, and w, consists of the following $43d + 1$ edges:*

- *vw, vv_i, $1 \leq i \leq d$; $v_i v^{11}_{i,j}$, $1 \leq i \leq d$, $1 \leq j \leq 3$;*

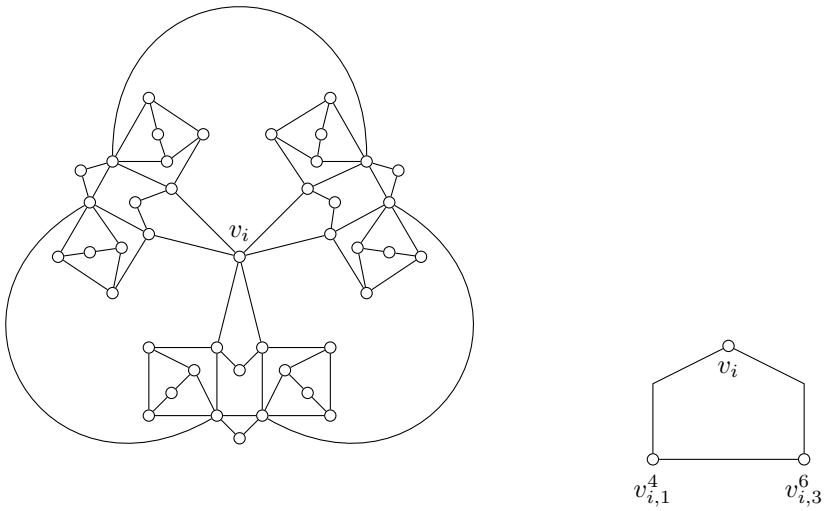

Figure 2: The graph X_i (left) and its symbolic drawing (right).

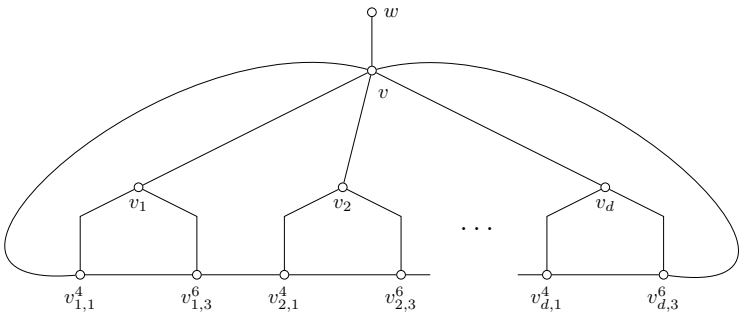

Figure 3: The graph Γ_d.

- $v^1_{i,j}v^2_{i,j}$, $v^1_{i,j}v^4_{i,j}$, $v^3_{i,j}v^4_{i,j}$, $v^3_{i,j}v^{14}_{i,j}$, $v^4_{i,j}v^5_{i,j}$, $v^4_{i,j}v^6_{i,j}$, $v^4_{i,j}v^{13}_{i,j}$, $v^6_{i,j}v^7_{i,j}$, $v^6_{i,j}v^9_{i,j}$, $v^6_{i,j}v^{11}_{i,j}$, $v^7_{i,j}v^{10}_{i,j}$, $v^8_{i,j}v^9_i$, $v^{12}_{i,j}v^{13}_{i,j}$, $1 \le i \le d$, $1 \le j \le 3$.

By the structure of $H_{i,j}, X_i$ and Γ_d we have the following facts for any independency tree T in Γ_d.

Fact 2. *v is an internal vertex of any independency tree T, and for each $1 \le i \le d$, $1 \le j \le 3$, we have the following situation.*

(i) *$v^4_{i,j}$ and $v^6_{i,j}$ are internal vertices of T;*

(ii) *T has at most two leaves in $\{v^1_{i,j}, v^2_{i,j}, v^3_{i,j}, v^4_{i,j}, v^{13}_{i,j}, v^{14}_{i,j}\}$ and at most two leaves in $\{v^6_{i,j}, v^7_{i,j}, v^8_{i,j}, v^9_{i,j}, v^{10}_{i,j}, v^{11}_{i,j}\}$. Hence, with $v^5_{i,j}$ and $v^{12}_{i,j}$, T has at most six leaves in $H_{i,j}$.*

(iii) *If $v^{11}_{i,j}$ or $v^{13}_{i,j}$ is a leaf of T, then T has at most five leaves in $H_{i,j}$.*

(iv) *If $v^{10}_{i,j}v^{11}_{i,j}$ or $v^{13}_{i,j}v^{14}_{i,j}$ is an edge of T, then T has at most five leaves in $H_{i,j}$.*

Proof. Because of w, v must be an internal vertex of T.

(i) Assume that for some i and j, $v^4_{i,j}$ is a leaf of T. Then $v^5_{i,j}v^6_{i,j}$ must be an edge of T (as $v^5_{i,j}$ has degree 2 in Γ_d). As T is an independency tree, $v^1_{i,j}$, $v^3_{i,j}$, $v^5_{i,j}$, $v^6_{i,j}$ and $v^{13}_{i,j}$ are not leaves. Hence, T contains the edge $v^5_{i,j}v^4_{i,j}$. Now, since $v^1_{i,j}$ and $v^3_{i,j}$ are internal vertices of T and have degree 3 in Γ_d, the four edges $v^1_{i,j}v^2_{i,j}$, $v^1_{i,j}v^{14}_{i,j}$, $v^3_{i,j}v^2_{i,j}$, $v^3_{i,j}v^{14}_{i,j}$ are edges in T. But they form a cycle in T, a contradiction. Thus, for any $1 \leq i \leq d$ and any $1 \leq j \leq 3$, $v^4_{i,j}$, and, by symmetry, $v^6_{i,j}$ are internal vertices of T.

(ii) This can be easily seen by inspection.

(iii) If $v^{11}_{i,j}$ or $v^{13}_{i,j}$ is a leaf of T, then $v^{12}_{i,j}$ is an internal vertex, hence, by (ii), T has at most five leaves in $H_{i,j}$.

(iv) By symmetry we consider only the case when $v^{10}_{i,j}v^{11}_{i,j}$ is an edge of T. Then $v^{10}_{i,j}$ must be an internal vertex of T (otherwise, as $v^7_{i,j}$ and $v^9_{i,j}$ have degree 3 in Γ_d, $v^7_{i,j}$, $v^8_{i,j}$, $v^9_{i,j}$, $v^6_{i,j}$ would induce a cycle in T, or two leaves among these vertices must be neighbors). Since at most one of $v^7_{i,j}$, $v^8_{i,j}$, $v^9_{i,j}$ is a leaf of T (as $v^8_{i,j}$ has degree 2 in Γ_d), it follows with (i) and (ii) that T has at most one leaf in $\{v^6_{i,j}, v^7_{i,j}, v^8_{i,j}, v^9_{i,j}, v^{10}_{i,j}, v^{11}_{i,j}\}$. Hence T has at most five leaves in $H_{i,j}$. □

Fact 3. *For any $1 \leq i \leq d$, T has at most 18 leaves in X_i. If v_i is a leaf, then T has at most 17 leaves in X_i.*

Proof. First we claim that if, for some $1 \leq i \leq d$ and some $1 \leq j \leq 3$, $v_iv^{11}_{i,j}$ and $v_iv^{13}_{i,j}$ are not edges of T, then T has at most five leaves in $H_{i,j}$. By Fact 2 (ii), this is clear if $v^{12}_{i,j}$ is not a leaf of T. Thus, let us assume that $v^{12}_{i,j}$ is a leaf of T. By symmetry, let $v^{11}_{i,j}v^{12}_{i,j}$ be the $v^{12}_{i,j}$-edge in T. Since the edge $v_iv^{13}_{i,j}$ is not in T, we have that $v^{13}_{i,j}v^{14}_{i,j}$ must be an edge in T. Therefore, by Fact 2 (iv), T has at most five leaves in $H_{i,j}$.

Now, if v_i is not a leaf of T, then T has at most 18 leaves in X_i by Fact 2 (ii). Assume that v_i is a leaf of T. Then, by the claim above and by Fact 2 (ii) again, T has at most $1 + 6 + 5 + 5 = 17$ leaves in X_i. □

Fact 4. *T has at most $18d + 1$ leaves. If, for some $1 \leq i \leq d$, v_i is a leaf of T, then T has at most $18d$ leaves.*

Proof. First, with w and Fact 3, T has at most $1 + 18d$ leaves.

Next, let $I = \{v_i \mid 1 \leq i \leq d, v_i \text{ is a leaf of } T\}$. By Fact 3 again, T has at most $1 + 17|I| + 18(d - |I|) \leq 18d$ many leaves, provided $I \neq \emptyset$. □

Proof of Theorem 3. Consider an arbitrary independency tree T of Γ_d with a maximal number of $18d+1$ leaves (by Facts 1 and 4). By Fact 4 and Fact 2 (iii), $D = \{v_1, \ldots, v_d\}$ is an independent set of internal vertices and all neighbors of D are internal vertices in T. Thus,

$$\{u \mid u \notin D \text{ is internal vertex of } T\}$$

is a vertex cover of Γ_d. By Fact 2 (i), this is a connected vertex cover of Γ_d. □

3. Parameterizing by the Number of Leaves

In this section we consider the natural parameterization by the number of leaves. Recall that a *Hamiltonian cycle* in a graph G is a subgraph of G that forms a cycle and contains all vertices of G. Clearly, removing any edge from a Hamiltonian cycle of G yields a spanning tree for G with exactly two leaves which are adjacent. The corresponding decision problem Hamiltonian Cycle asks for the existence of a Hamiltonian cycle in an input graph. Asking for a cliquy tree with exactly or at most two leaves is hence equivalent to Hamiltonian Cycle. Now, consider any graph $G = (V, E)$ with at least three vertices and pick any vertex $x \in V$. Construct the supergraph $G' = (V', E')$ of G by $V' = V \cup \{x', \ell, \ell'\}$, $E' = E \cup \{x'y \mid y \in N_G(x)\} \cup \{x\ell, x'\ell'\}$. As $|V| \geq 3$, G has a Hamiltonian cycle if and only if G' has a spanning tree with exactly two leaves, namely, ℓ and ℓ', which are not adjacent. Notice that G is triangle-free if and only if G' is triangle-free. Finally, observe that a triangle-free graph admits any cliquy tree if and only if it admits a cliquy tree with at most two leaves. As Hamiltonian Cycle is already NP-hard when restricted to triangle-free graphs [22], our observations (inspired by the considerations of Eppstein mentioned in the introduction) yield the following result.

Proposition 1. Min Leaf CT (leaf), Max Leaf CT (leaf) *and* Min Leaf IT (leaf) *are* Para-NP*-hard, even on the class of triangle-free graphs.*

For Max Leaf IT (leaf), we find that the requirement on the leaf set to be an independent set implicitly transfers the difficulty of the problem Independent Set. Simply by joining a large clique to a given, w.l.o.g., not complete, input graph G gives a graph G' for which it becomes possible and also trivial to build a spanning tree with any given subset of original vertices as leaves (assuming that this set has cardinality at least two). This way, finding an independency tree for G' is equivalent to finding an independent set in G which in particular shows that Max Leaf IT is NP-hard and also yields the following result.

Proposition 2. Max Leaf IT (leaf) *is* W[1]*-hard.*

Proof. We reduce from Independent Set. Given a graph G of order n and parameter $k \geq 2$, add a clique C of size $n-k+1$ and connect these new vertices with edges to all other vertices of the graph, yielding a new graph G'. If G' has an independency tree with leaf set I of size at least k, then, since $I \cap C = \emptyset$ (recall that every vertex of C is connected with every vertex in G'), I is an independent set for G.

If I is an independent set of size k in G, then we construct an independency tree for G' with leaf set I as follows. We first note that $G' - I$ contains $K_{n-k+1,n-k}$ as a subgraph (with C and $G - I$ being the partite sets of size $n - k + 1$ and $n - k$, respectively); thus $G' - I$ contains a Hamiltonian path P from $x \in C$ to $y \in C$. This path P with the vertices of I arbitrarily connected to the vertices x and y (in such a way that at least one vertex of I is connected to x and at least one vertex of I is connected to y) is an independency tree for G' with a leaf set of size k. □

4. Parameterizing by the Number of Internal Vertices

As we have seen in Section 3, parameterization by the number of leaves is hard. In this section we will therefore consider the dual parameter: the number

of internal vertices. The problem Max Leaf IT (internal), for example, is about deciding whether a graph admits an independency tree with at most k internal vertices, where k is the parameter.

This kind of dual parameterization is a common and in most cases successful strategy for graph-problems which are unlikely to be fixed parameter tractable by standard parameterization. For example, the fixed-parameter tractable problem Nonblocker is the dual to the W[2]-complete Dominating Set. We will see that this dual parameterization yields fixed-parameter tractability for most of the problems discussed in this paper, with the exception of Min Leaf CT (internal).

Theorem 4. Min Leaf CT (internal) *is* Para-NP*-hard, even on the class of triangle-free graphs.*

Proof. Recall that a graph of order $n \geq 3$ has a cliquy tree if and only if it has some cliquy tree with at least 1 internal vertex (thus, at most $n-1$ leaves). On triangle-free graphs, this is equivalent to the existence of a Hamiltonian cycle. Consequently, Hamiltonian Cycle for triangle-free graphs reduces to Min Leaf CT (internal), where the parameter number of internal vertices is fixed to 1. □

In the following, we will show that parameterizing by the number of internal vertices is more successful for the remaining problems.

4.1. Kernelizations for the Max-Leaf-Variants

In this section, we will show that Max Leaf IT (internal) and Max Leaf CT (internal) are in FPT by first proving the existence of single-exponential kernels and then designing single-exponential algorithms. Further, we will see that Min Leaf IT (internal) is in a sense equivalent to the fixed parameter tractable Max-Internal Spanning Tree.

Similar to *Buss'* rule for Vertex Cover, one can observe that for any input graph, a vertex of degree more than k cannot be a leaf of an independency tree with at most k internal vertices. The number of vertices of low degree not exclusively adjacent to high degree vertices is then bounded by $k(k+1)$. When constructing a kernel for Max Leaf IT (internal), however, it is not possible to delete vertices of high degree. It is only possible to remove some of the vertices exclusively adjacent to them, keeping only $k+1$ for each twin equivalence class (there are at most k vertices of degree larger than k, so there are at most 2^k equivalence classes). This kind of reduction gives the following result.

Theorem 5. *There is a kernel for* Max Leaf IT (internal) *with at most* $2^k(k+1)+k^2+2k$ *vertices.*

Proof. Let $G=(V,E)$ be a graph of order n and $k \in \mathbb{N}$. A vertex $v \in V$ with degree larger than k cannot be in an independent set of cardinality at least $n-k$, so if there exists an independency tree for G with at most k internal vertices, v cannot be a leaf of this tree. Let $V' := \{v \in V : deg(v) > k\}$. If $|V'| > k$, then there exists no independency tree with at most k internal vertices for G. Further, consider the set of vertices incident to edges which are not covered by V', given by $R := \{v \in V \setminus V' : N(v) \not\subseteq V'\}$. Covering the edges in $G[R]$ requires a subset of R of cardinality at least $|R|/(k+1)$, since all vertices in R

have degree at most k. Since the internal vertices of any independency tree are a vertex cover for G, we know that $|R| \leq k(k+1)$ or (G,k) is a no-instance for Max Leaf IT (internal).

The vertices $L := V \setminus (V' \cup R)$ only have neighbors in V' and especially form an independent set in G by the definition of R. Consider the set $V_S := \{v \in L\colon N(v) = S\}$ for some $S \subseteq V'$. If $|V_S| > k+1$, then deleting $|V_S| - k - 1$ vertices from V_S is a valid reduction rule. To see this, consider $G[V \setminus C]$ for a $C \subset V_S$ with $|C| = |V_S| - k - 1$. In any independency tree for $G[V \setminus C]$ with at most k internal vertices, the vertices in V' have to be internal, since their degree in $G[V \setminus C]$ is still larger than k. Consequently, if an independency tree with at most k internal vertices for $G[V \setminus C]$ exists, then we can build an independency tree for G by arbitrarily attaching the vertices in C to some vertex in V'; this does not violate the independence of the leaves, since $N(w) \subseteq V'$ for all $w \in C \subseteq L$, and does not increase the number of internal vertices. Let, on the other hand, T be an independency tree for G and let I be its set of internal vertices with $|I| \leq k$. We know that in T, a vertex $v \in V_S$ is either in I or a leaf attached to a vertex in V'. This means that $|V_S \setminus I|$ vertices from V_S are attached as leaves to $|V'| \leq |I \setminus V_S|$ internal vertices. By the pigeonhole principle, it is possible to delete at least $|V_S \setminus I| - |V'| \geq |V_S| - k$ of these without decreasing the number of internal vertices of T. Since vertices in V_S have the same properties with respect to their role in an independency tree, deleting $|V_S| - k - 1$ vertices from $V_S \setminus I$ in this way yields an independency tree for $G[V \setminus C]$.

After deleting $|V_S| - k - 1$ vertices from any V_S with $|V_S| > k+1$, we can conclude that $|L| \leq |\{S\colon S \subseteq V'\}| \cdot (k+1)$, which overall yields a reduced graph with at most $2^k(k+1) + k^2 + 2k$ vertices, as $|V'| \leq k$. □

Conversely, a vertex of degree less than $n-k$ cannot be a leaf of a cliquy tree with at most k internal vertices. As the set of internal vertices of a cliquy tree is a vertex cover of the complement graph, a similar reduction as for Theorem 5 yields:

Theorem 6. *There is a kernel for* Max Leaf CT (internal) *with at most* $2^k(k+1) + k^2 + 2k$ *vertices.*

Proof. Let $G = (V, E)$ be a graph of order n and $k \in \mathbb{N}$. Complementary to Max Leaf IT (internal), a vertex $v \in V$ with degree smaller than $n-k$ cannot be in a clique of cardinality at least $n-k$, so if there exists a cliquy tree for G with at most k internal vertices, then v cannot be a leaf of this tree. Let $V' := \{v \in V\colon deg(v) < n-k\}$. If $|V'| > k$, then there exists no cliquy tree with at most k internal vertices for G. Looking at the complement graph $\bar{G}$, it follows that the set of internal vertices of a cliquy tree for G is especially a vertex cover for $\bar{G}$. Vertices in $V \setminus V'$ have degree at most k in $\bar{G}$, so we can define the set R with similar properties as in Theorem 5 with respect to $\bar{G}$.

So, consider the set of vertices adjacent to edges in $\bar{G}$ which are not covered by V', given by $R := \{v \in V \setminus V'\colon V \setminus N(v) \not\subseteq V'\}$. Covering the edges in $\bar{G}[R]$ requires a subset of R of cardinality at least $|R|/(k+1)$, since all vertices in R have degree at most k in $\bar{G}$. Since the internal vertices of any cliquy tree for G are a vertex cover for $\bar{G}$, we know that $|R| \leq k(k+1)$ or (G,k) is a no-instance for Max Leaf CT (internal).

The vertices $L := V \setminus (V' \cup R)$ form an independent set in $\bar{G}$ and are also by definition adjacent to every vertex in R. Now, we can argue just like in Theorem 5 with the sets $V_S := \{v \in L \colon N(v) \cap V' = S\}$ for $S \subseteq V'$ that if $|V_S| > k+1$, then deleting $|V_S| - k - 1$ vertices from V_S is a valid reduction rule. Again, consider $G[V \setminus C]$ for a $C \subset V_S$ with $|C| = |V_S| - k - 1$. In any cliquy tree for $G[V \setminus C]$ with at most k internal vertices, the vertices in V' have to be internal: if $v \in S$ then $C \subset N[v]$ and the degree of v in $G[V \setminus C]$ is less than $n - |C| - k$; if $v \notin S$, then $V_S \subseteq V \setminus N[v]$, so the degree of v in $G[V \setminus C]$ is at most $n - |C| - |V_S \setminus C| = n - |C| - k - 1$. Consequently, if a cliquy tree with at most k internal vertices for $G[V \setminus C]$ exists, then we can build a cliquy tree for G by arbitrarily attaching the vertices in C to some vertex in V'; this does not violate the property of the leaves being a clique, since $R \cup L \subseteq N(v)$ for all $v \in C$ and does not increase the number of internal vertices. On the other hand, let T be a cliquy tree for G and let I be its set of internal vertices with $|I| \leq k$. We know that in T, a vertex $v \in V_S$ is either in I or a leaf. This means that $|V_S \setminus I|$ vertices from V_S are attached as leaves. By the pigeonhole principle, it is possible to delete at least $|V_S \setminus I| - |I \setminus V_S| \geq |V_S| - k$ of these from T without creating a leaf that is not adjacent to all other leaves (the only candidates for such a leaf are in $I \setminus V_S$). Since vertices in V_S have the same properties with respect to their role in a cliquy tree, deleting $|V_S| - k - 1$ vertices from $V_S \setminus I$ in this way yields a cliquy tree for $G[V \setminus C]$.

After deleting $|V_S| - k - 1$ vertices from any V_S with $|V_S| > k+1$, we can conclude that $|L| \leq |\{S \colon S \subseteq V'\}| \cdot (k+1)$, which overall yields a reduced graph with at most $2^k(k+1) + k^2 + 2k$ vertices. □

These exponential kernels prove membership in FPT but also raise the question if we can do better. Connected Vertex Cover considered as parameterized problem with the size of the cover as parameter is in FPT [5]. The close relation with Max Leaf IT only seems to transfer negative results in this perspective. It is known that, unless the polynomial time hierarchy collapses to the third level, Connected Vertex Cover with standard parameterization does not admit a polynomial kernel [10]. We will use a similar construction to show that same negative result holds for Max Leaf IT (internal).

We reduce from Red-Blue Dominating Set (RBDS), which is the following problem: Given a bipartite graph $G = (R \cup B, E)$ and an integer k, does there exist a set $X \subseteq R$, $|X| \leq k$, such that $N(X) = B$? As a parameter, one can (even) choose $k + |B|$. This problem admits no polynomial-size kernel unless the polynomial-time hierarchy collapses to the third level [10, Theorem 4.3].

Theorem 7. *Unless the polynomial-time hierarchy collapses to the third level, there is no polynomial-size kernel for* Max Leaf IT (internal).

Proof. Given a bipartite graph $G = (R \cup B, E)$ as input for RBDS, we construct a bipartite graph $G' = (V, E')$ with $V = R \cup B \cup \bar{B} \cup \{v\}$ where $\bar{B} := \{\bar{w} \colon w \in B\}$ is a copy of B and v is a new vertex, and $E' = E \cup \{(v, w) \colon w \in R\} \cup \{(w, \bar{w}) \colon w \in B\}$; observe that $B \cup \{v\}$ and $R \cup \bar{B}$ is a bipartition of G'. For this construction, there exists a red-blue dominating set of size at most k for G if and only if there exists an independency tree for G' with at most $|B| + k + 1$ internal vertices. Obviously, every vertex from B is internal (as only possible parent to its copy in $\bar{B}$) in an independency tree for G'. The only path connecting two vertices in B which do not have a common neighbor in R contains

v, so, unless there is a dominating set of cardinality one, v is also an internal vertex. Reaching all vertices in B from v with a minimum number of vertices from R is hence equivalent to finding a minimum red-blue dominating set for G and equivalently gives an independency tree for G' with a maximum number of leaves. □

Theorem 8. *Unless the polynomial-time hierarchy collapses to the third level, there is no polynomial-size kernel for the problem* Max Leaf CT (internal).

Proof. Given a bipartite graph $G = (R \cup B, E)$ and integer k as input for RBDS, we construct a split graph $G' = (V', E')$ with $V' = R \cup B \cup \bar{B} \cup C$ where $\bar{B} := \{\bar{w} \colon w \in B\}$ is a copy of B and C contains $k+2$ new vertices. Furthermore, E' contains all edges from E, all edges from $\{(w, \bar{w}) \colon w \in B\}$ and additional edges that turn $R \cup \bar{B} \cup C$ into a clique. We shall prove that G' has a cliquy tree with at least $|R| + |B| + 2$ leaves if and only if there exists a red-blue dominating set for G of cardinality at most k.

We start with the *only if* direction and assume that G' has a cliquy tree with a leaf set L with a cardinality of at least $|R| + |B| + 2$ and a set I of internal vertices with a cardinality of at most $|V'| - (|R| + |B| + 2) = k + |B|$. Since every leaf of the cliquy tree must have a degree of at least $|R| + |B| + 1$ (w. r. t. G') and every $b \in B$ has a degree of at most $|R| + 1$, we conclude that $B \subseteq I$. Let $D = I \setminus B$ with $|D| \leq k$ the remaining internal vertices. For every $b \in B$ not dominated by D, every neighbor of b in R (w. r. t. the cliquy tree) must be a leaf, which implies $\bar{b} \in I$. Consequently, $R \cap I$ with $|R \cap I| = k - k'$ for some k', $0 \leq k' \leq k$, dominate $|B| - k'$ vertices of B, which implies that there is a red-blue dominating set for G of cardinality at most k.

In order to prove the *if* direction, we assume that $D \subseteq R$ is a red-blue dominating set for G of cardinality k (any red-blue dominating set with cardinality less than k can be extended to one with cardinality exactly k). A cliquy tree for G' with at least $|R| + |B| + 2$ leaves can be constructed as follows. We connect all vertices of D to a path in arbitrary order and connect all vertices from B to this path by a single edge each (this is possible since D is a red-blue dominating set for G). The remaining $|V'| - (|B| + k) = |R| + |B| + 2$ vertices, which are all members of the clique $R \cup \bar{B} \cup C$, are now connected to some vertex of $D \cup B$ by a single edge each; thus, they are the leaves of the cliquy tree. □

We note that these reductions especially imply that Max Leaf IT restricted to bipartite graphs, and Max Leaf CT restricted to split graphs remains NP-hard.

4.2. Fpt-Algorithms for Max-Leaf-Variants

Given the kernelizations from the previous section, a brute-force algorithm on the kernel yields an fpt-algorithm for Max Leaf IT (internal) and Max Leaf CT (internal), however with a rather unpleasant running time. For Connected Vertex Cover with standard parameterization there exists an $O^*(2^k)$ algorithm [5] and it is tempting to try and use this approach to design a similar algorithm for Max Leaf IT (internal). This algorithm, however, relies on the fact that Connected Vertex Cover is closed under edge-contraction, a property that does not hold for independency trees; observe that edge contraction can even turn a graph with independency tree into a graph that has no

independency tree at all (e. g., K_4 minus one edge becomes K_3 after an edge contraction).

In this section we will use a different approach to design a single-exponential fpt-algorithm for Max Leaf IT (internal) and Max Leaf CT (internal). After providing some useful definitions, we proceed by proving in Lemma 1 and 2 a connection between independency trees with minimal number of internal vertices and a variant of minimal Steiner trees. We use this result in Theorem 10 to solve Max Leaf IT (internal) by enumerating minimal vertex covers and universal sets.

Definition 1 (Match-cover property)**.** *Let G be a connected graph with vertex set V and let C, L, N be subsets of V. We say that (G, C, L, N) satisfies the* match-cover property *if C is a vertex cover of G and $L \subseteq C$, such that there exists a complete matching from L to N. For (G, C, L, N) satisfying the match-cover property, we define $\mathrm{IT}_G(C, L, N)$ to be the set of all independency trees T on G such that*

- *all vertices in C are internal vertices of T,*
- *every vertex that has only leaves as children in T is contained in L,*
- *no internal vertex of T is contained in N.*

Furthermore, we define $\mathrm{ST}_G(C, L, N)$ to be the set of all trees T on G such that

- *every vertex from C is contained in T,*
- *the leaf set of T is a subset of L,*
- *no vertex of T is contained in N.*

Lemma 1. *Let (G, C, L, N) be a tuple which satisfies the match-cover property. The set $\mathrm{ST}_G(C, L, N)$ contains a tree with at most k vertices if and only if $\mathrm{IT}_G(C, L, N)$ contains an independency tree with at most k internal vertices.*

Proof. For the first direction let $T \in \mathrm{ST}_G(C, L, N)$ with $|T| = k$. We construct an independency tree T' on G such that the internal vertices of T' are exactly the vertices of T. Every leaf of T is contained in L. In order to become internal vertices, these vertices need at least one child. There exists a perfect matching from L to N and no vertex of N is contained in T. Thus, we can extend T by appending a private child from N to every leaf of T. Furthermore, G is connected and T contains the vertex cover C, so every vertex not in C is adjacent to at least one vertex in T. We attach all remaining vertices to arbitrary vertices in T. The result is an independency tree $T' \in \mathrm{IT}_G(C, L, N)$ with k internal vertices.

For the inverse direction, let $T \in \mathrm{IT}_G(C, L, N)$ be an independency tree with k internal vertices. We remove all leaves from T. The result is a tree $T' \in \mathrm{ST}_G(C, L, N)$ of size k. □

We will later decide whether a graph admits an independency tree with at most k internal vertices by searching for such a tree in various sets $\mathrm{IT}_G(C, L, N)$. Lemma 1 states that we might as well check if a set $\mathrm{ST}_G(C, L, N)$ contains a tree of size at most k. This subproblem is very similar to the well known problem Steiner Tree which asks for a given graph $G = (V, E)$ and integer k and a set

of terminals $C \subseteq V$ whether there exists a tree T of size at most k in G which contains C. However, a tree $T \in \mathrm{ST}_G(C, L, N)$ needs to satisfy the additional requirement that the leaf set is a subset of L.

We will solve this problem by modifying the classical Dreyfus-Wagner algorithm [13], which runs in $O^*(3^{|C|})$ time, in such a way, that we also check if the leaf set is a subset of L. For input (G, C, L, N) our algorithm computes for every subset X of C with $X \cap L \neq \emptyset$ and each $v \in X \setminus L$, if possible, the smallest among all spanning trees for X such that the set of leaves is a subset of $L \cup \{v\}$. The important idea here is that we can build up a tree with leaf set L bottom-up from subtrees for which at most one leaf is not in L, so that we have to remember at most one vertex $v \in X \setminus L$ for which a further edge is needed. Otherwise, just like in the dynamic program for the classical Steiner tree problem, we consider for X not just all subsets of C but all subsets $S \subset V$ with $|S \setminus C| \leq 1$, hence remembering one Steiner vertex in order to possibly append more subtrees to it. Here we make explicit use of the fact that C is a vertex cover, and thus $G \setminus C$ contains no edges (and particularly no edges of the Steiner tree). This yields the following result.

Algorithm 1: GenSteinerTree

Input : $G = (V, E)$, $C \subseteq V$, $L \subseteq C, k \in \mathbb{N}$
Output: Size k Steiner tree for G with terminal set C and leaves from L

for *every* $S \subseteq V$ *with* $|S \setminus C| \leq 1$ *and* $v \in V$ **do**
  $s[S] \leftarrow \infty$; $s_v[S] \leftarrow \infty$; $edges[S] \leftarrow \emptyset$; $edges_v[S] \leftarrow \emptyset$;
for *every* $u \in L$ **do**
  $s[\{u\}] \leftarrow 0$; $s_u[\{u\}] \leftarrow 0$;
for *every* i, $2 \leq i \leq |C|$ **do**
  for *every* $X \subseteq V$ *with* $|X \setminus C| \leq 1$, $X \cap L \neq \emptyset$ *and* $|X| = i$ **do**
    for *every* $X' \subset X$ *with* $\emptyset \neq X'$ **do**
      for *every* $(u, w) \in E \cap (X' \times (X \setminus X'))$ **do**
        if $s_u[X'] + s_w[X \setminus X'] < \min\{s[X], k + 1 - |C|\}$ **then**
          $s[X] \leftarrow s_u[X'] + s_w[X \setminus X']$;
          $edges[X] \leftarrow edges_u[X'] \cup edges_w[X \setminus X'] \cup \{(u, w)\}$;
    for *every* $v \in X \setminus L$ **do**
      for *every* $u \in N(v) \cap X$ **do**
        if $s_u[X \setminus \{v\}] + |\{v\} \setminus C| < \min\{s_v[X], k + 1 - |C|\}$ **then**
          $s_v[X] \leftarrow s_u[X \setminus \{v\}] + |\{v\} \setminus C|$;
          $edges_v[X] \leftarrow edges_u[X \setminus \{v\}] \cup \{(u, v)\}$;
      if $s[X] < s_v[X]$ **then**
        $s_v[X] \leftarrow s[X]$; $edges_v[X] \leftarrow edges[X]$;
return $edges[C]$;

Lemma 2. *Let (G, C, L, N) be a tuple which satisfies the match-cover property and $k \geq |C|$ be the parameter. It is possible to decide in time $O^*(3^{|C|})$ and space $O^*(2^{|C|})$ whether $\mathrm{ST}_G(C, L, N)$ contains a tree of size at most k.*

Proof. For simplicity, we first assume that $N = \emptyset$. Assume there exists a tree $T \in \mathrm{ST}_G(C, L, N)$ of size at most k. We say that a tree T' has property $\mathcal{P}$ iff at least one vertex is from L and at most one leaf of T' is not in L. Let T' be a

subtree of T with property $\mathcal{P}$ with at least two vertices. It is always possible to either delete an edge from T' to split it into two connected components which both satisfy property $\mathcal{P}$ or to delete the only vertex from $V \setminus L$ of degree one in T' and its adjacent edge to create a tree with property $\mathcal{P}$. Indeed, if a tree T' with property $\mathcal{P}$ has no degree-one vertex which is not in L, deleting any arbitrary edge yields two components of property $\mathcal{P}$. For a tree T' which has exactly one degree-one vertex v which is not in L, removing v and the edge that connects v to T' yields another tree with property $\mathcal{P}$. Obviously, T itself has property $\mathcal{P}$, so bottom-up, starting with $\{v\}$ for all $v \in L$ as exactly the smallest subtrees of T with property $\mathcal{P}$, we can hence inductively build up T by connecting two trees of property $\mathcal{P}$ by an edge or by appending one vertex from $V \setminus L$ to a tree of property $\mathcal{P}$.

We claim that Algorithm 1 computes a tree in $\mathrm{ST}_G(C, L, N)$ of size at most k or it returns the empty set in case no such tree exists. The algorithm computes, for every subset X of C with $X \cap L \neq \emptyset$ and each $v \in X \setminus L$, if possible, the smallest among all spanning trees for X such that the set of leaves is a subset of $L \cup \{v\}$. It stores the number of Steiner-nodes used in the table $s[X]$ if the tree has only leaves in L and in the table $s_v[X]$ if $v \in X$ is the only leaf which is not in L and stores the corresponding edge-set in table $\mathit{edges}[X]$ and $\mathit{edges}_v[X]$, respectively. Hence, finally $\mathit{edges}[C]$ stores the requested solution, a description of a Steiner tree of size k for G with terminal set C and leaves from L.

Just like in the dynamic program for the classical Steiner tree problem, we consider for X not just all subsets of C but all subsets $S \subset V$ with $|S \setminus C| \leq 1$, hence remembering one Steiner vertex in order to possibly append more subtrees to it. Trying all partitions into two non-empty subsets of X as subtrees and all vertices $v \in X$ to append as degree-one vertex in $V \setminus L$ covers all possibilities to build the best tree for X and inductively computes the smallest Steiner tree for C with leaves from L. The correctness of the algorithm follows inductively along the same lines as a proof of the correctness of the Dreyfus-Wagner algorithm that can be found in any textbook. We therefore refrain from giving more details.

Note that we use this algorithm for the specific case that the set of Steiner vertices is an independent set which means that when combining two subtrees by an edge (u, w), as done in Algorithm 1, u and w cannot both be Steiner vertices. Hence it is enough to only remember one Steiner vertex in our case. For a more general scenario with non-independent Steiner vertices, one has to keep track of two Steiner vertices but this does not affect the asymptotic running time in $O^*(3^{|C|})$ of the algorithm which is dominated by checking all partitions of subsets of C.

At last, forbidding the set of vertices N in case $N \neq \emptyset$ from being used in building a tree spanning the vertices from C is done by simply running Algorithm 1 for the graph $G[V \setminus N]$. □

We note that there are faster known algorithms for Steiner Tree than Dreyfus-Wagner, for example the algorithm due to Nederlof [27] which runs in time $O^*(2^k)$ and polynomial space. Our modifications, however, do not seem to be trivially extensible to these other algorithms. Combining Lemmas 1 and 2 yields the following result.

Theorem 9. *Let (G, C, L, N) be a tuple which satisfies the match-cover property and k be the parameter. It is possible to decide in time $O^*(3^{|C|})$ and space*

$O^*(2^{|C|})$ *whether* $\mathrm{IT}_G(C, L, N)$ *contains an independency tree with at most* k *internal vertices.*

So far, we can efficiently check whether a set $\mathrm{IT}_G(C, L, N)$ contains an independency tree with at most k internal vertices. Furthermore we know that every independency tree is contained in at least one set $\mathrm{IT}_G(C, L, N)$. We could solve the problem by iterating over all possible choices for C, L, N and return yes if any independency tree in $\mathrm{IT}_G(C, L, N)$ has at most k internal vertices. Although we can enumerate all minimal vertex covers of size at most k of a graph in $O^*(2^k)$ (see [15] and [8, Theorem 4]), there are too many possibilities to check for the set N if we simply brute-force through all subsets of $V \setminus C$.

If for fixed sets C and L, there exists some set N such that $\mathrm{IT}_G(C, L, N)$ contains an independency tree of size at most k, then we know that a set N of cardinality $|L|$ is sufficient, as N is only needed to block the matching leaves of the independency tree from falsely being chosen as internal vertices. On the other hand, at most $k - |C|$ vertices from $V \setminus N$ are needed as internal vertices to build an independency tree in $\mathrm{IT}_G(C, L, N)$. Hence we only need to consider sets N which select from $V \setminus C$ a set that contains the $|L|$ vertices needed as leaves but does not include the $k - |C|$ vertices needed as internal vertices. Sets with this property are contained in a $(|V \setminus C|, |L| + k - |C|)$-universal set for $V \setminus C$, where a (n, l)-universal set for some ground set X of cardinality n is family Φ_l of subsets of X such that $2^S = \{A \cap S \colon A \in \Phi_l\}$ for any $S \subseteq X$ with $|S| \leq l$. It is possible to construct an (n, l)-universal set in time $O^*(2^l)$ (see [26]), which gives the following result.

Lemma 3. *Let* C *be a vertex cover of* G*. It can be decided in time* $O^*(9^k)$ *and space* $O^*(2^k)$ *whether* G *admits an independency tree* T *with at most* k *internal vertices such that all vertices in* C *are internal vertices of* T*.*

Proof. If $|C| > k$, then no such tree exists, so from now on we assume $|C| \leq k$. Let Φ_l be an (n, l)-universal set [26] over V with $l = |L| + k - |C|$. This means that Φ_l is a family of subsets of V such that $2^S = \{A \cap S \colon A \in \Phi_l\}$ for any $S \subseteq V$ with $|S| \leq l$. Consider the following algorithm: Let $\mathcal{T}$ be the set of tuples (L, N) such that $L \subseteq C$ and $N \in \Phi_{|L|+k-|C|}$ such that there exists a perfect matching from L to N. The algorithm proceeds by iterating over all tuples $(L, N) \in \mathcal{T}$ and uses Theorem 9 to decide whether $\mathrm{IT}_G(C, L, N)$ contains an independency tree with at most k internal vertices. It accepts if one set $\mathrm{IT}_G(C, L, N)$ contains such an independency tree, otherwise it rejects.

We show that the algorithm is correct by proving that, for every independency tree T^* with at most k internal vertices such that all vertices in C are internal vertices of T, there exists a tuple $(L^*, N^*) \in \mathcal{T}$ with $T^* \in \mathrm{IT}_G(C, L^*, N^*)$. We choose $(L^*, N^*) \in \mathcal{T}$ as follows: Let L^* be the set of internal vertices of T^* which have only leaves as children. Suppose that there exists a vertex $x \in L^* \setminus C$. The vertex x is internal, thus it has at least two neighbors y, z in T^*. Since x is not part of the vertex cover C, $y, z \in C$. Thus, y, z are internal vertices and $x \notin L^*$. This is a contradiction. It follows that $L^* \subseteq C$. Furthermore, let X be the set of internal vertices of T^* not contained in C and let M be a set which contains exactly one leaf of every vertex in L^*. Let $l^* := |L| + k - |C|$. It holds that $X \cap M = \emptyset$ and $|X| + |M| \leq l^*$. We choose $N^* \in \Phi_{l^*}$ such that $X \cap N^* = \emptyset$ and $M \subseteq N^*$. Thus, no internal vertex of T^* is contained in N^* and there exists a perfect matching from L^* to N^*. In summary, it holds that:

- C is a vertex cover of G and all vertices of C are internal vertices of T^*.
- $L^* \subseteq C$ and every vertex of T^* which has only leaves as children is contained in L^*.
- There exists a perfect matching from L^* to N^* and no internal vertex of T^* is contained in N^*.

We can conclude that (G, C, L^*, N^*) fulfills the match-cover property and $T^* \in \mathrm{IT}_G(C, L^*, N^*)$. Thus, the algorithm is correct.

We now consider the run time. We iterate over every subset $L \in 2^C$. By a result from Naor, Schulman and Srinivasan [26], one can construct an (n, l)-universal set of size $O^*(2^l)$ in time $O^*(2^l)$. For sets C, L, we need a family of $(n, |L| + k - |C|)$-universal sets. The total number of tuples is bounded by

$$\sum_{l=0}^{|C|} \binom{|C|}{l} \cdot O^*(2^{l+k-|C|}) = O^*(2^{k-|C|} \cdot 3^{|C|}) = O^*(2^k \cdot 1.5^{|C|})$$

Furthermore, it takes $O^*(3^{|C|})$ time to decide whether $\mathrm{IT}_G(C, L, N)$ contains an independency tree with at most k internal vertices. Since $|C| \leq k$, we arrive at a total running time of $O^*(9^k)$. □

Applying Lemma 3, we can provide the claimed fpt-algorithm for Max Leaf IT (internal).

Theorem 10. Max Leaf IT (internal) *can be decided in time* $O^*(18^k)$ *and space* $O^*(2^k)$.

Proof. Assume G admits an independency tree T with at most k internal vertices. Since the leaf set of T is an independent set, the internal vertices form a vertex cover. Thus, there exists a minimal vertex cover C of G of size at most k such that the internal vertices of T are a superset of C.

We enumerate all minimal vertex covers C of size at most k and use Lemma 3 to accept if there exists an independency tree with at most k internal vertices such that its set of internal vertices is a superset of C. Otherwise we reject.

There are at most 2^k minimal vertex covers of size at most k, which can be enumerated in $O^*(2^k)$ time; see [15] and [8, Theorem 4]. Furthermore, according to Lemma 3 we can decide in time $O^*(9^k)$ whether G admits an independency tree T with at most k internal vertices such that all vertices in C are internal vertices of T. The total running time is $O^*(18^k)$. □

In the following we show that Cliquy Tree can be reduced to solving Independency Tree. Instead of looking at vertex covers in G we look at vertex covers C in the complement graph $\overline{G}$. At first, it seems that an equivalent construction as in Lemma 1 is possible, but since C is no longer a vertex cover in G we cannot assume that every vertex has edges into this set. For that reason, we reduce it to independency tree by exchanging edges.

For the next statements, we denote for a graph $G = (V, E)$ and any set $X \subseteq V$ by $G|_X$ the graph constructed from G by deleting every edge (u, v) with $u, v \notin X$.

Lemma 4. *Let $G = (V, E)$ be a graph. Let C be a vertex cover of $\overline{G}$ of size at most k. There exists a cliquy tree with at most k internal vertices whose internal vertices are a superset of C if and only if (i) or (ii) holds:*

(i) There exists an independency tree in $G|_C$ whose internal vertices are exactly the vertices in C.

(ii) There exists an $x \in V \setminus C$ and an independency tree with at most k internal vertices in $G|_{C \cup \{x\}}$ whose internal vertices are a superset of $C \cup \{x\}$.

Proof. Assume (i) or (ii) is true. There exists an independency tree T in $G|_C$ or $G|_{C\cup\{x\}}$ such that its internal vertices are a superset of C. Hence, its leaves are a subset of $G \setminus C$. Since C is a vertex cover in $\overline{G}$, $G \setminus C$ forms a clique in G. Thus, T is a cliquy tree in G. Furthermore, T contains at most k internal vertices and its internal vertices are a superset of C.

For the other direction, assume that T is a cliquy tree with at most k internal vertices whose internal vertices are a superset of C. If the internal vertices are exactly C, then T is an independency tree on $G|_C$ and we are done. Otherwise, let $x \notin C$ be an internal vertex of T. Let $v, w \in V \setminus (C \cup \{x\})$ such that (v, w) is an edge in T. We can assume that there is a path from x to w in T which does not touch v (otherwise exchange v and w). We remove (v, w) from T and add (x, v). This is possible since x and v are part of a clique. Since w and x already were internal vertices, the resulting tree does not have fewer leaves. We repeat this procedure until T induced on $V \setminus C$ forms a star with root x. Now T is an independency tree of size at most k in $G|_{C\cup\{x\}}$ whose internal vertices are a superset of $C \cup \{x\}$. □

We are now ready to extend the statement of Theorem 10 to Max Leaf CT (internal).

Theorem 11. Max Leaf CT (internal) *can be decided in time $O^*(18^k)$ and space $O^*(2^k)$.*

Proof. Assume G admits a cliquy tree T with at most k internal vertices. Since the leaf set of T forms an independent set in $\overline{G}$, the internal vertices form a vertex cover in $\overline{G}$. Thus, there exists a minimal vertex cover C of $\overline{G}$ of size at most k such that the internal vertices of T are a superset of C.

We enumerate all minimal vertex covers C of $\overline{G}$ of size at most k. If there exists an independency tree in $G|_C$ whose internal vertices are exactly C, then we accept. Otherwise, we iterate over all vertices $x \in V \setminus C$ and accept if there exists an independency tree of size at most k in $G|_{C\cup\{x\}}$ whose internal vertices are a superset of $C \cup \{x\}$. According to Lemma 4, we accept if and only if there exists a cliquy tree with at most k internal vertices whose internal vertices are a superset of C.

There are at most 2^k minimal vertex covers of size at most k in $\overline{G}$, which can be enumerated in $O^*(2^k)$ time; see [15] and [8, Theorem 4]. Furthermore, the sets C and $C \cup \{x\}$ form a vertex cover in $G|_C$ and $G|_{C\cup\{x\}}$, respectively. Thus, according to Lemma 3, we can decide in time $O^*(9^k)$ whether $G|_{C\cup\{x\}}$ or $G|_X$ admit an appropriate independency tree. The total running time is $O^*(18^k)$. □

4.3. Fpt-Algorithm and 2k-Kernel for Min Leaf IT (internal)

In this section, we show that Min Leaf IT (internal) is in FPT. We do so by showing a strong relation between independency trees with a minimal number of leaves and spanning trees with a minimal number of leaves. The problem of deciding whether a graph contains a spanning tree with at most k leaves, parameterized by the number of internal vertices, is known as Max-Internal Spanning Tree. This problem is in FPT with current best known running time $O^*(4^k)$ [24]. We use this result to decide whether a graph contains an independency tree with at most k leaves, parameterized by the number of internal vertices, i.e., we solve Min Leaf IT (internal). For this, we need the following lemma:

Lemma 5. *For any graph G that has an independency tree, the minimal number of leaves of a spanning tree of G equals the minimal number of leaves of an independency tree of G.*

Proof. Let $G = (V, E)$ be any graph which admits an independency tree. Let T be a spanning tree of G with a minimal number of leaves. We distinguish between two cases:

First case: T has two leaves, i.e., T is a Hamiltonian path. According to Böhme et al. [2], every Hamiltonian graph which admits an independency tree also admits a Hamiltonian path with non-adjacent endpoints. Such a path is an independency tree with two leaves.

Second case: T has at least three leaves. Assume there are leaves x, y of T which are adjacent in G. There exists a vertex z of degree at least three which connects x and y in T. Let z' be the first vertex along the path from z to x. We create a new spanning tree T' from T by removing the edge (z, z') and adding the edge (x, y). The leaves x, y become internal vertices in T' and the internal vertex z' becomes a leaf in T'. The tree T' has one leaf less than T. However, T was assumed to be a spanning tree with a minimal number of leaves. Thus, the assumption that some leaves of T were adjacent in G was false, and hence, T is an independency tree.

For every minimal spanning tree we have found an independency tree with the same number of leaves. This independency tree is minimal, because every independency tree is also a spanning tree. □

This connection allows to transfer the $2k$ vertex kernel for Max-Internal Spanning Tree [24].

Theorem 12. *There is a kernel of at most $2k$ vertices for* Min Leaf IT (internal).

Proof. Let (G, k) be the input for the problem, where G is a connected graph and k is an integer. We describe the kernelization procedure in the following. First, we check if G is isomorphic to C_n, K_n or $K_{n/2,n/2}$. If this is the case, then we output a trivial no-instance, because G has no independency tree [2]. Otherwise, Lemma 5 shows that (G, k) is a yes-instance to Min Leaf IT (internal) if and only if G has a spanning tree with at least k internal vertices. Now, we employ the kernel for Max-Internal Spanning Tree by Li et al. [24], which is a $2k$ vertex kernel. If the resulting graph G' is isomorphic to C_n, K_n or $K_{n/2,n/2}$, then we can solve Max-Internal Spanning Tree in polynomial-time on this Max-Internal Spanning Tree-kernel and output a trivial yes-

or no-instance for Min Leaf IT (internal). Otherwise, we output G' (and k') without any further change. The correctness follows directly from Lemma 5. □

We can now also use Lemma 5 to transfer results from Max-Internal Spanning Tree to Min Leaf IT (internal).

Lemma 6. Min Leaf IT (internal) *can be solved in time* $O^*(4^k)$ *and polynomial space.*

Proof. Let (G, k) be the input for the problem, where G is a connected graph and k an integer. First, we check if G is isomorphic to C_n, K_n or $K_{n/2,n/2}$. If this is the case, then we output a trivial no-instance, because G has no independency tree [2]. Otherwise, an independency tree exists, and we use an $O^*(4^k)$ algorithm [24] to solve Max-Internal Spanning Tree for (G, k). By Lemma 5, this algorithm is successful in computing a spanning tree with at least k internal vertices if and only if there exists an independency tree in G with at least k internal vertices. □

If we want to give a corresponding independency tree in case of a yes-instance, then we start with the spanning tree T computed for Max-Internal Spanning Tree. If any two leaves of T are not independent, then we successively create a tree with strictly less leaves with the procedure used in the argument of the proof for Lemma 5. This way, we either arrive at an independency tree with even more than k internal vertices or at a Hamiltonian cycle. Since G is not isomorphic to C_n, K_n, or $K_{n/2,n/2}$, there exists a Hamiltonian path in G which is not a cycle [2]. The argument used for this result gives a quadratic number of possible Hamiltonian paths in G of which at least one has to be an independency tree.

5. An Exact Exponential-Time Algorithm for All Variants

In this section we present an exact exponential algorithm for Min/Max Leaf IT and Min/Max Leaf CT. To be precise, we will construct a more general algorithm for finding spanning trees where the leaf set is maximal or minimal and satisfies some properties. The running time will depend on the time required to check the property.

The basic idea is to enumerate all possible partitions into internal vertices and leaves. Since we do not care about the internal structure it is not necessary to enumerate all spanning trees. We have to check if for a given partition some spanning tree exists. This subproblem, i. e., given a graph $G = (V, E)$ and $L \subseteq V$, check whether G has a spanning tree with leaf set L, is already a hard problem by reduction from Hamiltonian Path: A graph G has a Hamiltonian path from vertex x to y if and only if G has a spanning tree with leaf set $\{x, y\}$. We will iteratively compute partial solutions in parallel for all possible leaf-sets via dynamic programming: At any given time, A_i contains all possible partitions into internal vertices and leaves of a spanning tree of any subgraph with i vertices. The set A_{i+1} is then constructed from A_i by checking if the trees of A_i can be extended with a new vertex. The final table A_n contains all possible partitions into internal vertices and leaves of a spanning tree of the whole graph. For any graph $G = (V, E)$ and i, $1 \leq i \leq |V|$, let A_i be equal to

$$\{(K, L) \colon K, L \subseteq V, |K| + |L| = i, G[K \cup L] \text{ has a spanning tree with leaf set } L\}.$$

We will now show how we can extend A_{i-1} to A_i by adding a vertex.

Lemma 7. *$(K, L) \in A_i$ if and only if there exist $k \in K$, $l \in L$ and $(k, l) \in E$, such that either*

(i) $(K \setminus \{k\}, (L \setminus \{l\}) \cup \{k\}) \in A_{i-1}$, or

(ii) $(K, L \setminus \{l\}) \in A_{i-1}$.

Proof. Let T be a spanning tree of $G[K \cup L]$ with non-empty leaf set L. Let l be any leaf in T with parent vertex k. Let T' be the spanning tree of $G[(K \cup L) \setminus \{l\}]$ obtained by removing l from T. If k is a leaf in T' then (i) holds, otherwise (ii).

Assume (i) or (ii) holds for some $k \in K$, $l \in L$ which are adjacent in G. There exists a spanning tree T' on $G[K \cup L']$ with leaf set $L' = L \setminus \{l\}$. We obtain a spanning tree T on $G[K \cup L]$ by adding the vertex l and the edge (k, l) to T'. The vertex k is not a leaf in T. If (i) holds, then k is a leaf in T'. The leaf set of T equals $(L' \cup \{l\}) \setminus \{k\} = L$. If (ii) holds, then k is not a leaf in T'. The leaf set of T equals $L' \cup \{l\} = L$. So T is a spanning tree on $G[K \cup L]$ with leaf set L. □

We can now apply Lemma 7 to prove Theorem 1. The basic idea for this result is to compute the set A_n in $O(3^n \cdot (n + m))$ time, where m is the number of edges, using dynamic programming and Lemma 7. We then simply filter the entries of A_n, only keeping those where the leaf set satisfies the property P, and select the entry which maximizes/minimizes the leaf set.

Theorem 1 (restated)**.** *Let G be a graph of order n. There exists an algorithm running in time $O(3^n \cdot f(n))$ and space $O(3^n)$ that constructs a spanning tree with a maximal/minimal number of leaves, where the leaves satisfy a property P that can be checked in time $O(f(n))$.*

Proof. We prove this for graphs with one component and at least one edge. For $i \in \{2, \ldots, n\}$ let $A_i \in 2^V \times 2^V$ be the set of all tuples (K, L) such that

- $K \cap L = \emptyset$
- $|K| + |L| = i$
- there exists a spanning tree on $G[K \cup L]$ with leaf set L.

The desired output can easily be derived from A_n in polynomial time. We iteratively construct A_i for increasing values of i. For a graph of size two the spanning tree consists of a single edge, thus the base case is given by $A_2 = \{(\emptyset, \{v, u\}) \mid (v, u) \in E\}$. We proceed by computing A_{i+1} from A_i. We iterate over all tuples $(K, L) \in 2^V \times 2^V$ with $K \cap L = \emptyset$ and $|K| + |L| = i + 1$. If $L = \emptyset$ we can skip the tuple, since every tree needs to have at least one leaf. Otherwise, for every $l \in L$, we add (K, L) to A_{i+1} if for some $k \in N(l) \cap K$, one of the following holds:

(i) $(K \setminus \{k\}, (L \setminus \{l\}) \cup \{k\}) \in A_i$, or
(ii) $(K, L \setminus \{l\}) \in A_i$.

From Lemma 7, we know that this is enough to find all entries in A_{i+1}. During the course of computing $A_1, \ldots, A_n$ we iterate over $\sum_{i=1}^{n} \binom{n}{i} 2^i = 3^n - 1$ many (K, L) tuples. For each such (K, L) with $|K| + |L| = i$ we pick a leaf $l \in L$ and evaluate (i) and (ii) by checking A_i at at most $|N(l)| + 1$ positions. Thus, A_n can be constructed in $O(3^n \cdot (n + m))$, where m is the number of edges. In the last step we filter the entries of A_n, only keeping those where the leaf set satisfies the property P. We then select the entry which maximizes/minimizes the leaf set. We then construct the optimal spanning tree via backtracking. It is easy to see that we only ever need to keep A_i and A_{i-1} in memory, which leads to the claimed space requirement. □

Since checking if a set is an independent set or a clique can be done in polynomial time, we arrive at the following corollary.

Corollary 1. *There exists an $O^*(3^n)$ time and $O(3^n)$ space algorithm for graphs of order n to construct an independency (cliquy) tree of G with a maximal/minimal number of leaves.*

Remark 1. *While it might be possible (though challenging) to design algorithms that solve these problems, say, in time $O^*(2^n)$, we can rule out algorithms with running times of $2^{o(n)}$ under the Exponential Time Hypothesis (ETH). Theorem 7 was proven by a reduction from RBDS to* Max Leaf IT *which only requires a linear blowup in the number of vertices of the input graph. As* Dominating Set *is, quite obviously, linearly reducible to RBDS, this also shows that there exists no $2^{o(n)}$ exact algorithm for* Max Leaf IT*, unless ETH fails [25]. The same holds for* Max Leaf CT *with the reduction used for Theorem 8.* Min Leaf IT *and* Min Leaf CT *are for $k = 2$ equivalent to* Hamiltonian Cycle*; see Proposition 1. This shows the claim by looking at classical reductions that show* NP*-hardness of* Hamiltonian Cycle *[21, 25].*

6. Conclusions

In Theorem 10 we showed that Max Leaf IT (internal) can be decided by an algorithm which runs in time $O^*(18^k)$ and exponential space. This algorithm needs to compute a slight modification of a Steiner tree as a subroutine. We currently do this using a modified version of the classical Dreyfus-Wagner algorithm, which runs, just like the original algorithm, in time $O^*(3^k)$ and exponential space, where k is the number of terminal nodes. This is also the only part in our algorithm that needs exponential space. Using more advanced methods however, one can solve the Steiner tree problem in time $O^*(2^k)$ and polynomial space [27]. The running time of this subroutine is a multiplicative factor in the running time of our algorithm. It would be interesting to check if it is possible to adapt these advanced methods for solving Steiner Tree to our problem, and thereby solve Max Leaf IT (internal) in polynomial space or obtain a faster single-exponential time (we would obtain $O^*(12^k)$ time by using the best known algorithm).

We only considered parameters intrinsic to the problem. A question for further research is to parameterize the problem by structural parameters like treewidth or vertex cover number. Especially vertex cover has a deep connection to the independency tree versions, since if the vertex cover number is high, there is no independency tree with many leaves. The only part of our algorithm for

Max Leaf IT (internal) that requires the number of internal vertices k and not just the vertex cover size as parameter is the iteration over all universal sets.

We have developed exact algorithms for constructing spanning trees assuming that the set of leaves satisfies a certain well-computable property P. It would be interesting to study this type of problems in the framework of parameterized complexity. For which properties P can we prove fpt results, and for which $\mathsf{W}[1]$-hardness results?

Finally, we hope this paper spurs interest in faster fpt algorithms for the problems studied in this paper, and steady improvement is obtained just like for other spanning tree variants, such as Max-Leaf Spanning Tree and Max-Internal Spanning Tree.

Acknowledgements

We are very grateful for discussions with Van Bang Le. He was so kind to send us the construction on which Theorem 3 is based. Previously, we only found examples that refuted Conjecture 1 in a weaker form. The first author was subsidized by DFG grant FE 560/6-1. We are also grateful to the reviewers whose comments helped improve the presentation of our results.

References

[1] D. Binkele-Raible, H. Fernau, S. Gaspers, and M. Liedloff. Exact and parameterized algorithms for Max Internal Spanning Tree. *Algorithmica*, 65:95–128, 2013.

[2] T. Böhme, H. Broersma, F. Göbel, A. V. Kostochka, and M. Stiebitz. Spanning trees with pairwise nonadjacent endvertices. *Discrete Mathematics*, 170(1-3):219–222, 1997.

[3] P. M. Camerini, G. Galbiati, and F. Maffioli. Complexity of spanning tree problems: Part I. *European Journal of Operational Research*, 5(5):346–352, 1980.

[4] S. Chen, I. Ljubic, and S. Raghavan. The regenerator location problem. *Networks*, 55(3):205–220, 2010.

[5] M. Cygan. Deterministic parameterized connected vertex cover. In F. V. Fomin and P. Kaski, editors, *Algorithm Theory – SWAT 2012 – 13th Scandinavian Symposium and Workshops*, volume 7357 of *LNCS*, pages 95–106. Springer, 2012.

[6] M. Cygan, F. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015.

[7] J. Daligault, G. Gutin, E. J. Kim, and A. Yeo. FPT algorithms and kernels for the directed k-leaf problem. *Journal of Computer and System Sciences*, 76(2):144–152, 2010.

[8] P. Damaschke. Parameterized enumeration, transversals, and imperfect phylogeny reconstruction. *Theoretical Computer Science*, 351(3):337–350, 2006.

[9] R. Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, 2000.

[10] M. Dom, D. Lokshtanov, and S. Saurabh. Kernelization lower bounds through colors and IDs. *ACM Trans. Algorithms*, 11(2):13:1–13:20, October 2014.

[11] R. J. Douglas. NP-completeness and degree restricted spanning trees. *Discrete Mathematics*, 105(1-3):41–47, 1992.

[12] R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

[13] S. E. Dreyfus and R. A. Wagner. The Steiner problem in graphs. *Networks*, 1(3):195–207, 1971.

[14] M. R. Fellows, D. K. Friesen, and M. A. Langston. On finding optimal and near-optimal lineal spanning trees. *Algorithmica*, 3:549–560, 1988.

[15] H. Fernau. On parameterized enumeration. In O. H. Ibarra and L. Zhang, editors, *Computing and Combinatorics, Proceedings COCOON 2002*, volume 2383 of *LNCS*, pages 564–573. Springer, 2002.

[16] M. J. Flores, J. A. Gámez, and S. Moral. The independency tree model: a new approach for clustering and factorisation. In M. Studený and J. Vomlel, editors, *Third European Workshop on Probabilistic Graphical Models*, pages 83–90, 2006.

[17] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006.

[18] Internet forum "*mathoverflow*". Graphs in which every spanning tree is an independency tree. `https://mathoverflow.net/questions/141355/graphs-in-which-every-spanning-tree-is-an-independency-tree`.

[19] Internet forum "*mathoverflow*". On "super connected" graphs. `https://mathoverflow.net/questions/140200/on-super-connected-graphs`.

[20] Internet forum "*stackexchange*". Completeness spanning trees. `https://cstheory.stackexchange.com/questions/18875/completeness-spanning-trees`.

[21] M. R. Garey and D. S. Johnson. *Computers and Intractability*. New York: Freeman, 1979.

[22] A. Itai, C. H. Papadimitriou, and J. L. Szwarcfiter. Hamiltonian paths in grid graphs. *SIAM Journal on Computing*, 11(4):676–686, 1982.

[23] M. Knauer and J. Spoerhase. Better approximation algorithms for the maximum internal spanning tree problem. *Algorithmica*, 71(4):797–811, 2015.

[24] W. Li, Y. Cao, J. Chen, and J. Wang. Deeper local search for parameterized and approximation algorithms for maximum internal spanning tree. *Information and Computation*, 252:187–200, 2017.

[25] D. Lokshtanov, D. Marx, and S. Saurabh. Lower bounds based on the Exponential Time Hypothesis. *EATCS Bulletin*, 105:41–72, 2011.

[26] M. Naor, L. J. Schulman, and A. Srinivasan. Splitters and near-optimal derandomization. In *36th Annual Symposium on Foundations of Computer Science, FOCS*, pages 182–191. IEEE Computer Society, 1995.

[27] J. Nederlof. Fast polynomial-space algorithms using inclusion-exclusion; improving on Steiner tree and related problems. *Algorithmica*, 65(4):868–884, 2013.

[28] E. Prieto and C. Sloper. Either/or: Using vertex cover structure in designing FPT-algorithms—the case of k-internal spanning tree. In F. K. H. A. Dehne, J.-R. Sack, and M. H. M. Smid, editors, *Proceedings of WADS 2003, Workshop on Algorithms and Data Structures*, volume 2748 of *LNCS*, pages 465–483. Springer, 2003.

[29] G. Salamon. Approximating the maximum internal spanning tree problem. *Theoretical Computer Science*, 410:5273–5284, 2009.

[30] G. Salamon. Vulnerability bounds on the number of spanning tree leaves. *Ars Mathematica Contemporanea*, 2:77–92, 2009.