PARAMETERIZED COMPLEXITY OF DIRECTED STEINER TREE ON SPARSE GRAPHS*

MARK JONES†, DANIEL LOKSHTANOV‡, M. S. RAMANUJAN‡, SAKET SAURABH‡,§, AND ONDŘEJ SUCHÝ¶

Abstract. We study the parameterized complexity of the directed variant of the classical Steiner Tree problem on various classes of directed sparse graphs. While the parameterized complexity of Steiner Tree parameterized by the number of terminals is well understood, not much is known about the parameterization by the number of non-terminals in the solution tree. All that is known for this parameterization is that both the directed and the undirected versions are W[2]-hard on general graphs, and hence unlikely to be fixed parameter tractable (FPT). The undirected Steiner Tree problem becomes FPT when restricted to sparse classes of graphs such as planar graphs, but the techniques used to show this result break down on directed planar graphs.

In this article we precisely chart the tractability border for Directed Steiner Tree (DST) on sparse graphs parameterized by the number of non-terminals in the solution tree. Specifically, we show that the problem is fixed parameter tractable on graphs excluding a topological minor, but becomes W[2]-hard on graphs of degeneracy 2. On the other hand we show that if the subgraph induced by the terminals is acyclic then the problem becomes FPT on graphs of bounded degeneracy.

We further show that our algorithm achieves the best possible asymptotic running time dependence on the solution size and degeneracy of the input graph, under standard complexity theoretic assumptions. Using the ideas developed for DST, we also obtain improved algorithms for Dominating Set on sparse undirected graphs. These algorithms are asymptotically optimal.

Key words. Algorithms and Data Structures. Graph Algorithms. Parameterized Algorithms. Steiner Tree problem. Sparse Graph Classes.

AMS subject classifications. G.2.2, F.2.2

1. Introduction.

In the Steiner Tree problem we are given as input an n-vertex graph $G = (V, E)$ and a set $T \subseteq V$ of terminals. The objective is to find a subtree ST of G spanning T that minimizes the number of vertices in ST. Steiner Tree is one of the most intensively studied graph problems in Computer Science. Steiner trees are important in various applications such as VLSI routing [31], phylogenetic tree reconstruction [28] and network routing [34]. We refer to the book of Prömel and Steger [43] for an overview of the results on, and applications of the Steiner Tree problem. The Steiner Tree problem is known to be NP-hard [22], and remains hard even on planar graphs [21]. The minimum number of non-terminals can be approximated to within $O(\ln t)$, but cannot be approximated to within $(1-\varepsilon)\ln t$, for any $\varepsilon > 0$, where t is the number of terminals, unless NP $\subseteq$ DTIME$[n^{\text{polylog } n}]$ (see [32]). Furthermore the edge-weighted variant of Steiner Tree remains APX-complete, even when the graph is complete and all edge costs are either 1 or 2 (see [3]).

*An extended abstract of this paper previously appeared in the Proceedings of the 21st European Symposium on Algorithms, ESA 2013 [30].

†Department of Computer Science, Royal Holloway University of London, United Kingdom, markj@cs.rhul.ac.uk

‡University of Bergen, Norway, {daniello, Ramanujan.Sridharan}@ii.uib.no

§The Institute of Mathematical Sciences, Chennai, India, saket@imsc.res.in. The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement no. 306992.

¶Faculty of Information Technology, Czech Technical University in Prague, Czech Republic, ondrej.suchy@fit.cvut.cz. Part of the work done while with Universität des Saarlandes, Saarbrücken, Germany. Partially supported by the DFG Cluster of Excellence on Multimodal Computing and Interaction (MMCI), DFG project DARE (GU 1023/1-2), the Indo-German Max Planck Center for Computer Science (IMPECS), and by the Czech Science Foundation project 14-13017P.

In this paper we study a natural generalization of Steiner Tree to directed graphs, from the perspective of parameterized complexity. The goal of parameterized complexity is to find ways of solving NP-hard problems more efficiently than by brute force. The aim is to restrict the combinatorial explosion in the running time to a parameter that is much smaller than the input size for many input instances occurring in practice. Formally, a *parameterization* of a problem is an assignment of an integer k to each input instance and we say that a parameterized problem is *fixed-parameter tractable* (FPT) if there is an algorithm that solves the problem in time $f(k) \cdot |I|^{O(1)}$, where $|I|$ is the size of the input instance and f is an arbitrary computable function depending only on the parameter k. Above FPT, there exists a hierarchy of complexity classes, known as the W-hierarchy. Just as NP-hardness is used as an evidence that a problem is probably not polynomial time solvable, showing that a parameterized problem is hard for one of these classes gives evidence that the problem is unlikely to be fixed-parameter tractable. The main classes in this hierarchy are:

$$\text{FPT} \subseteq \text{W[1]} \subseteq \text{W[2]} \subseteq \cdots \subseteq \text{W[P]} \subseteq \text{XP}$$

The principal analogue of the classical intractability class NP is W[1]. In particular, this means that an FPT algorithm for any W[1]-hard problem would yield a $O(f(k)n^c)$ time algorithm for every problem in the class W[1]. XP is the class of all problems that are solvable in time $O(n^{g(k)})$. Here, g is some (usually computable) function. For more background on parameterized complexity the reader is referred to the monographs [8, 13, 17, 39].

We consider the following directed variant of Steiner Tree.

Directed Steiner Tree (DST) **Parameter:** k
Input: A directed graph $D = (V, A)$, a root vertex $r \in V$, a set $T \subseteq V \setminus \{r\}$ of terminals and an integer $k \in \mathbb{N}$.
Question: Is there a set $S \subseteq V \setminus (T \cup \{r\})$ of at most k vertices such that the digraph $D[S \cup T \cup \{r\}]$ contains a directed path from r to every terminal $t \in T$?

The DST problem is well studied in approximation algorithms, as the problem generalizes several important connectivity and domination problems on undirected as well as directed graphs [6, 12, 25, 27, 44, 45]. These include Group Steiner Tree, Node Weighted Steiner Tree, TSP and Connected Dominating Set. However, this problem has so far largely been ignored in the realm of parameterized complexity. The aim of this paper is to fill this gap.

It follows from the reduction presented in [37] that DST is W[2]-hard on general digraphs. Hence we do not expect FPT algorithms to exist for these problems, and, therefore, we turn our attention to classes of *sparse* digraphs. Our results give a nearly complete picture of the parameterized complexity of DST on sparse digraphs. Specifically, we prove the following results. We use the O^* notation to suppress factors polynomial in the input size.

1. There is an $O^*(2^{O(hk)})$-time algorithm for DST on digraphs excluding K_h as a minor[1]. Here K_h is a clique on h vertices.
2. There is an $O^*(f(h)^k)$-time algorithm for DST on digraphs excluding K_h as a topological minor.

[1]When we say that a digraph excludes a fixed (undirected) graph as a minor or a topological minor, or that the digraph has degeneracy d we mean that the statement is true for the underlying undirected graph.

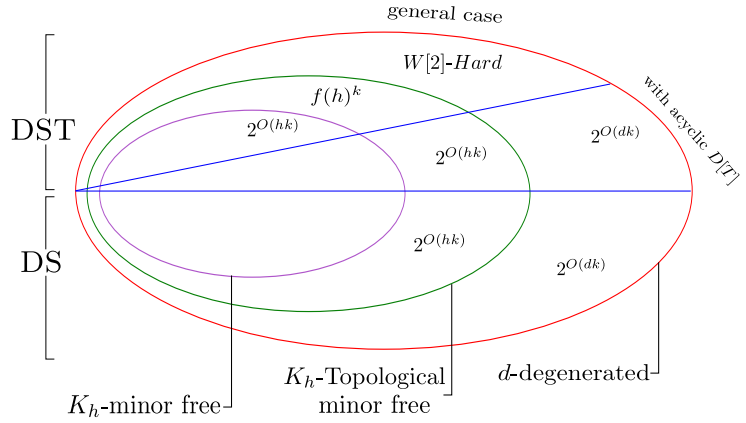

FIG. 1. *A summary of the main results in the paper*

3. There is an $O^*(2^{O(hk)})$-time algorithm for DST on digraphs excluding K_h as a topological minor if the graph induced on terminals is acyclic.
4. DST is W[2]-hard on 2-degenerate digraphs if the graph induced on terminals is allowed to contain directed cycles.
5. There is an $O^*(2^{O(dk)})$-time algorithm for DST on d-degenerate graphs if the graph induced on terminals is acyclic, implying that DST is FPT parameterized by k on $o(\log n)$-degenerate graph classes. This yields the first FPT algorithm for STEINER TREE on *undirected* d-degenerate graphs.
6. For any constant $c > 0$, there is no $f(k)n^{o(\frac{k}{\log k})}$-time algorithm for DST on graphs of degeneracy at most $c \log n$ even if the graph induced on terminals is acyclic, unless the Exponential Time Hypothesis [29] (ETH) fails.

Our algorithms for DST hinge on a novel branching which exploits the domination-like nature of the DST problem. The branching is based on a new measure which seems useful for various connectivity and domination problems on both directed and undirected graphs of bounded degeneracy. We demonstrate the versatility of the new branching by applying it to the DOMINATING SET problem on graphs excluding a topological minor and more generally, graphs of bounded degeneracy. The well-known DOMINATING SET problem is defined as follows.

DOMINATING SET	**Parameter:** k
Input: An undirected graph $G = (V, E)$, and an integer $k \in \mathbb{N}$.	
Question: Is there a set $S \subseteq V$ of at most k vertices such that every vertex in G is either in S or adjacent to a vertex in S?	

Our $O^*(2^{O(dk)})$-time algorithm for DOMINATING SET on d-degenerate graphs improves over the $O^*(k^{O(dk)})$ time algorithm by Alon and Gutner [2]. It turns out that our algorithm is essentially optimal – we show that, assuming the ETH, the running time dependence of our algorithm on the degeneracy of the input graph and solution size k cannot be significantly improved. Using these ideas we also obtain a polynomial time d^2 factor approximation algorithm for DOMINATING SET on d-degenerate graphs. We give survey of existing literature on DOMINATING SET and the results for it in Section 4. We believe that our new branching and corresponding measure will turn out to be useful for several other problems on sparse (di)graphs.

Related Results. Though the parameterized complexity of DST has so far been largely ignored, it has not been left completely unexplored. In particular the classical dynamic programming algorithm by Dreyfus and Wagner [14] from 1972 solves Steiner Tree in time $O^*(3^t)$ where t is the number of terminals in the input graph. The algorithm can also be used to solve DST within the same running time, and may be viewed as an FPT algorithm for Steiner Tree and DST if the number of terminals in the instance is the parameter. Fuchs et al. [20] improved the algorithm of Dreyfus and Wagner and obtained an algorithm with running time $O^*((2+\epsilon)^t)$, for any constant $\epsilon > 0$, where the degree of the polynomial factor depends on ϵ. More recently, Björklund, Husfeldt, Kaski, and Koivisto [4] obtained an $O^*(2^t)$ time algorithm for the cardinality version of Steiner Tree. Finally, Nederlof [38] obtained an algorithm running in $O^*(2^t c)$ and polynomial space for the case of positive integer weights bounded by c. Recently, Fomin et al. [18] obtained a polynomial space algorithm for weighted version of Steiner Tree running in time $O^*(7.97^t)$. All of these algorithms can also be modified to work for DST.

For most hard problems, the most frequently studied parameter in parameterized complexity is the size or quality of the solution. For Steiner Tree and DST, however, this is not the case. The non-standard parameterization of the problem by the number of terminals is well-studied, while the standard parameterization by the number of non-terminals in the solution tree has been left unexplored, aside from the simple W[2]-hardness proofs [37]. Steiner-type problems in directed graphs from parameterized perspective were studied in [26] in arc-weighted setting, but the paper focuses more on problems in which the required connectivity among the terminals is more complicated than just a tree.

For Steiner Tree parameterized by the solution size k, there is a simple (folklore) FPT algorithm on planar graphs. The algorithm is based on the fact that planar graphs have the diameter-treewidth property [16], the fact that Steiner Tree can be solved in polynomial time on graphs of bounded treewidth [9] along with a simple preprocessing step. In this step, one contracts adjacent terminals to single vertices and removes all vertices at distance at least $k+1$ from any terminal. For DST, however, this preprocessing step breaks down. Thus, prior to this work, nothing was known about the standard parameterization of DST aside from the W[2]-hardness result on general graphs.

Recently Steiner Tree in sparse graphs was considered with respect to the total size of the constructed tree $k+t$. Pilipczuk, Pilipczuk, Sankowski, and van Leeuwen [41] showed that there is a subexponential algorithm for the problem on planar graphs running in $O\left(2^{O\left(((k+t)\log(k+t))^{2/3}\right)}n\right)$ time. Later they improved the running time to $O\left(2^{O\left(\sqrt{(k+t)\log(k+t)}\right)}+(k+t)^{142}n\right)$ and, more importantly, presented a kernel for Steiner Tree in planar graphs of size $O\left((k+t)^{142}\right)$ [42].

2. Preliminaries. Given a digraph $D=(V,A)$, for each vertex $v \in V$, we define $N^+(v)=\{w \in V | (v,w) \in A\}$ and $N^-(v)=\{w \in H | (w,v) \in A\}$. In other words, the sets $N^+(v)$ and $N^-(v)$ are the set of out-neighbors and in-neighbors of v, respectively.

Degeneracy of an undirected graph $G=(V,E)$ is defined as the least number d such that every subgraph of G contains a vertex of degree at most d. Degeneracy of a digraph is defined to be the degeneracy of the underlying undirected graph. We say that a class of (di)graphs $\mathcal{C}$ is $o(\log n)$-*degenerate* if there is a function $f(n)=o(\log n)$ such that every (di)graph $G \in \mathcal{C}$ is $f(|V(G)|)$-degenerate.

In a directed graph, we say that a vertex u *dominates* a vertex v if there is an arc (u, v) and in an undirected graph, we say that a vertex u dominates a vertex v if there is an edge (u, v) in the graph.

Given a vertex v in a directed graph D, we define the operation of *short-circuiting* across v as follows. We add an arc from every vertex in $N^-(v)$ to every vertex in $N^+(v)$ and delete v.

For a set of vertices $X \subseteq V(G)$ such that $G[X]$ is connected we denote by G/X the graph obtained by contracting edges of a spanning tree of $G[X]$ in G.

Given an instance (D, r, T, k) of DST, we say that a set $S \subseteq V \setminus (T \cup \{r\})$ of at most k vertices is *a solution* to this instance if in the digraph $D[S \cup T \cup \{r\}]$ there is a directed path from r to every terminal $t \in T$.

Minors and Topological Minors. For a graph $G = (V, E)$, a graph H is a *minor* of G if H can be obtained from G by deleting vertices, deleting edges, and contracting edges. We denote that H is a minor of G by $H \preceq G$. A mapping $\varphi : V(H) \to 2^{V(G)}$ is a model of H in G if for every $u, v \in V(H)$ with $u \neq v$ we have $\varphi(u) \cap \varphi(v) = \emptyset$, $G[\varphi(u)]$ is connected, and, if (u, v) is an edge of H, then there are $u' \in \varphi(u)$ and $v' \in \varphi(v)$ such that $(u', v') \in E(G)$. It is known, that $H \preceq G$ if and only if H has a model in G.

A *subdivision* of a graph H is obtained by replacing each edge of H by a non-trivial path. We say that H is a *topological minor* of G if some subgraph of G is isomorphic to a subdivision of H and denote it by $H \preceq_T G$. In this paper, whenever we make a statement about a directed graph having (or being) a minor of another graph, we mean the underlying undirected graph. A graph G *excludes graph H as a (topological) minor* if H is not a (topological) minor of G. We say that a class of graphs $\mathcal{C}$ *excludes $o(\log n)$-sized (topological) minors* if there is a function $f(n) = o(\log n)$ such that for every graph $G \in \mathcal{C}$ we have that $K_{f(|V(G)|)}$ is not a (topological) minor of G.

Tree Decompositions. A *tree decomposition* of a graph $G = (V, E)$ is a pair (M, β) where M is a rooted tree and $\beta : V(M) \to 2^V$, such that:

1. $\bigcup_{t \in V(M)} \beta(t) = V$.
2. For each edge $(u, v) \in E$, there is a $t \in V(M)$ such that both u and v belong to $\beta(t)$.
3. For each $v \in V$, the nodes in the set $\{t \in V(M) \mid v \in \beta(t)\}$ form a connected subtree of M.

The following notations are the same as that in [24]. Given a tree decomposition of graph $G = (V, E)$, we define mappings $\sigma, \gamma, \alpha : V(M) \to 2^V$ by letting for all $t \in V(M)$,

$$\sigma(t) = \begin{cases} \emptyset & \text{if } t \text{ is the root of } M \\ \beta(t) \cap \beta(s) & \text{if } s \text{ is the parent of } t \text{ in } M \end{cases}$$

$$\gamma(t) = \bigcup\nolimits_{u \text{ is a descendant of } t} \beta(u)$$

$$\alpha(t) = \gamma(t) \setminus \sigma(t).$$

Let (M, β) be a tree decomposition of a graph G. The *width* of (M, β) is $\max\{|\beta(t)| - 1 \mid t \in V(M)\}$, and the *adhesion* of the tree decomposition is $max\{|\sigma(t)| \mid t \in V(M)\}$. For every node $t \in V(M)$, the *torso* at t is the graph

$$\tau(t) := G[\beta(t)] \cup E(K[\sigma(t)]) \cup \bigcup\nolimits_{u \text{ child of } t} E(K[\sigma(u)]),$$

where $K[V']$ is a complete graph on vertex set V'.

Again, by a tree decomposition of a directed graph, we mean a tree decomposition for the underlying undirected graph.

3. DST **on sparse graphs.** In this section, we introduce our main idea and use it to design algorithms for the Directed Steiner Tree problem on classes of sparse graphs. We begin by giving an $O^*(2^{O(hk)})$ algorithm for DST on K_h-minor free graphs. Following that, we give an $O^*(f(h)^k)$ algorithm for DST on K_h-topological minor free graphs for some f. Then, we show that in general, even in 2-degenerate graphs, we cannot expect to have an FPT algorithm for DST parameterized by the solution size. Finally, we show that when the graph induced on the terminals is acyclic, then our ideas are applicable and we can give a $O^*(2^{O(hk)})$ algorithm on K_h-topological minor free graphs and a $O^*(2^{O(dk)})$ algorithm on d-degenerate graphs.

3.1. DST **on minor free graphs.** We begin with a polynomial time preprocessing which will allow us to identify a *special* subset of the terminals with the property that it is enough for us to find an arborescence from the root to these terminals.

Rule 1. *Given an instance (D, r, T, k) of* DST*, let C be a strongly connected component in the graph $D[T]$ with at least 2 vertices. Then, contract C to a single vertex c, to obtain the graph D' and return the instance $(D', r, T' = (T \setminus C) \cup \{c\}, k)$.*

Lemma 3.1. *Rule 1 is sound, i.e., the instance (D', r, T', k) produced by the rule is a yes-instance of* DST *if and only if (D, r, T, k) is a yes-instance of* DST*.*

Proof. Suppose S is a solution to (D, r, T, k). Then there is a directed path from r to every terminal $t \in T$ in the digraph $D[S \cup T \cup \{r\}]$. Contracting the vertices of C will preserve this path. Hence, S is also a solution for (D', r, T', k).

Conversely, suppose S is a solution for (D', r, T', k). If the path P from r to some $t \in T'$ in $D'[S \cup T' \cup \{r\}]$ contains c, then there must be a path from r to some vertex x of C and a path (possibly trivial) from some vertex $y \in C$ to t in $D[S \cup T \cup \{r\}]$. As there is a path between any x and y in $D[C]$, concatenating these three paths results in a path from r to t in $D[S \cup T \cup \{r\}]$. Hence, S is also a solution to (D, r, T, k). □

Observation 3.2. *Given an undirected graph $G = (V, E)$ which excludes K_h as a minor for some h, and a vertex subset $X \subseteq V$ inducing a connected subgraph of G, the graph G/X also excludes K_h as a minor.*

Proof. Suppose G/X contains K_h as a minor. Then K_h can be obtained from G/X by deleting vertices, deleting edges, and contracting edges. Since G/X can be obtained from G by contracting edges, K_h is a minor of G, a contradiction. □

We call an instance *reduced* if Rule 1 cannot be applied to it. Given an instance (D, r, T, k), we first apply Rule 1 exhaustively to obtain a reduced instance. Since the resulting graph still excludes K_h as a minor (by Proposition 3.2), we have not changed the problem and hence, for ease of presentation, we denote the reduced instance also by (D, r, T, k). We call a terminal vertex $t \in T$ a *source-terminal* if it has no in-neighbors in $D[T]$. We use T_0 to denote the set of all source-terminals. Since for every terminal, the graph $D[T]$ contains a path from some source terminal to this terminal, we have the following observation.

Observation 3.3. *Let (D, r, T, k) be a reduced instance and let $S \subseteq V$. Then the digraph $D[S \cup T \cup \{r\}]$ contains a directed path from r to every terminal $t \in T$ if and only if it contains a directed path from r to every source-terminal $t \in T_0$.*

In particular, notice that if S is a solution, then for every $t \in T_0$ there is a vertex in S dominating it.

We also need the following observation about short-circuiting across a vertex.

OBSERVATION 3.4. *Let (D, r, T, k) be a reduced instance and T_0 the set of source terminals. Let $v \in T \setminus T_0$, D' be obtained from D by short-circuiting across v and $S \subseteq V$. Then the digraph $D[S \cup T \cup \{r\}]$ contains a directed path from r to every terminal $t \in T$ if and only if $D'[S \cup (T \setminus \{v\}) \cup \{r\}]$ contains a directed path from r to every terminal $t \in T \setminus \{v\}$.*

The following is an important subroutine of our algorithm.

LEMMA 3.5. *Let D be a digraph, $r \in V(D)$, $T \subseteq V(D) \setminus \{r\}$ and $T_0 \subseteq T$. There is an algorithm which can find a minimum size set $S \subseteq V(D)$ such that there is path from r to every $t \in T_0$ in $D[T \cup \{r\} \cup S]$ in time $O^*(2^{|T_0|})$.*

Proof. Nederlof [38] gave an algorithm to solve the STEINER TREE problem on undirected graphs in time $O^*(2^t)$ where t is the number of terminals. Misra et al. [36] observed that the same algorithm can be easily modified to solve the DST problem in time $O^*(2^t)$ with t being the number of terminals. In our case, we create an instance of the DST problem by taking the same graph, defining the set of terminals as T_0 and for every vertex $t \in T \setminus T_0$, short-circuiting across this vertex. By Observation 3.4, a k-sized solution to this instance gives a k-sized solution to the original problem. To actually find the set of minimum size, we can first find its size by a binary search and then delete one by one the non-terminals, if their deletion does not increase the size of the minimum solution. □

We call the algorithm from Lemma 3.5, NEDERLOF(D, r, T, T_0).

We also need the following structural claim regarding the existence of low degree vertices in graphs excluding K_h as a topological minor.

LEMMA 3.6. *Let $G = (V, E)$ be an undirected graph excluding K_h as a topological minor*[2] *and let $X, Y \subseteq V$ be two disjoint vertex sets. If every vertex in X has at least $h-1$ neighbors in Y, then there is a vertex in Y with at most ch^4 neighbors in $X \cup Y$ for some constant c.*

Proof. It was proved in [5, 33], that there is a constant a such that any graph that does not contain K_h as a topological minor is $d = ah^2$-degenerate. Consider the graph $H_0 = G[X \cup Y] \setminus E(X)$. We construct a sequence of graphs $H_0, \ldots, H_l$, starting from H_0 and repeating an operation which ensures that any graph in the sequence excludes K_h as a topological minor. The operation is defined as follows. In graph H_i, pick a vertex $x \in X$. As it has degree at least $h-1$ in Y and there is no K_h topological minor in H_i, it has two neighbors y_1 and y_2 in Y, which are non-adjacent. Remove x from H and add the edge (y_1, y_2) to obtain the graph H_{i+1}. By repeating this operation, we finally obtain a graph H_l where the set X is empty. As the graph H_l still excludes K_h as a topological minor, it is d-degenerate, and hence it has at most $d|Y|$ edges. In the sequence of operations, every time we remove a vertex from X, we added an edge between two previously non-adjacent vertices of Y. Hence, the number of vertices in X in H_0 is bounded by the number of edges within Y in H_l, which is at most $d|Y|$. As H_0 is also d-degenerate, it has at most $d(|X|+|Y|) = d(d+1)|Y|$ edges. Therefore, there is a vertex in Y incident on at most $2d(d+1) = 2ah^2(ah^2+1) \leq ch^4$ edges where $c = 4a^2$. This concludes the proof of the lemma. □

Let (D, r, T, k) be a reduced instance of DST, $Y \subseteq V \setminus (T \cup \{r\})$ be a set of non-terminals representing a partial solution and d_b be some fixed positive integer. We define the following sets of vertices (see Fig. 2).

- $T_1 = T_1(Y)$ is the set of source terminals dominated by Y.

[2] As a graph G which excludes K_h as a minor, also excludes K_h as a topological minor, the lemma also applies in the former case. While a stronger bound can be given for this case, stating the lemma this way allows us to use it in further sections and does not hurt the asymptotic running time.

- $B_h = B_h(Y, d_b)$ is the set of non-terminals not in Y which dominate at least $d_b + 1$ terminals in $T_0 \setminus T_1$.
- $B_l = B_l(Y, d_b)$ is the set of non-terminals not in Y which dominate at most d_b terminals in $T_0 \setminus T_1$.
- $W_h = W_h(Y, d_b)$ is the set of terminals in $T_0 \setminus T_1$ which are dominated by B_h.
- $W_l = W_l(Y, d_b) = T_0 \setminus (T_1 \cup W_h)$ is the set of source terminals which are not dominated by Y or B_h.

Note that the sets $Y, T_1, B_h, B_l, W_h, W_l$ are pairwise disjoint. The constant d_b is introduced to describe the algorithm in a more general way so that we can use it in further sections of the paper. Throughout this section, we will have $d_b = h - 2$.

LEMMA 3.7. *Let (D, r, T, k) be a reduced instance of* DST*, $Y \subseteq V \setminus T$, $d_b \in \mathbb{N}$, and T_1, B_h, B_l, W_h, and W_l as defined above. If $|W_l| > d_b(k - |Y|)$, then the given instance does not admit a solution containing Y.*

Proof. This follows from the fact that any non-terminal from $V \setminus (B_h \cup Y)$ in the solution, which dominates a vertex in W_l can dominate at most d_b of these vertices. Since the solution contains at most $k - |Y|$ such non-terminals, at most $d_b(k - |Y|)$ of these vertices can be dominated. Since every source terminal must be dominated in a solution, this completes the proof. □

LEMMA 3.8. *Let (D, r, T, k) be a reduced instance of* DST*, $Y \subseteq V \setminus T$, $d_b \in \mathbb{N}$, and T_1, B_h, B_l, W_h, and W_l as defined above. If B_h is empty, then there is an algorithm which can test if this instance has a solution containing Y in time $O^*(2^{d_b(k-|Y|)+|Y|})$.*

Proof. We use Lemma 3.5 and test whether $|\textsc{Nederlof}(D, r, T \cup Y, Y \cup (T_0 \setminus T_1))| \leq k - |Y|$. We know that $|Y| \leq k$ and, by Lemma 3.7, we can assume that $|T_0 \setminus T_1| \leq d_b(k - |Y|)$. Therefore, the size of $Y \cup (T_0 \setminus T_1)$ is bounded by $|Y| + d_b(k - |Y|)$, implying that we can solve the DST problem on this instance in time $O^*(2^{d_b(k-|Y|)+|Y|})$. This completes the proof of the lemma. □

We now proceed to the main algorithm of this subsection.

THEOREM 3.9. DST *can be solved in time $O^*(3^{hk+o(hk)})$ on graphs excluding K_h as a minor.*

Proof. Let T_0 be the set of source terminals of this instance. The algorithm we describe takes as input a reduced instance (D, r, T, k), a vertex set Y and a positive integer d_b and returns a smallest solution for the instance which contains Y if such a solution exists. If there is no solution, then the algorithm returns a dummy symbol S_∞. To simplify the description, we assume that $|S_\infty| = \infty$. The algorithm is a recursive algorithm and at any stage of the recursion, the corresponding recursive step returns the smallest set found in the recursions initiated in this step. We start with Y being the empty set.

By Lemma 3.7, if $|W_l| > d_b(k - |Y|)$, then there is no solution containing Y and hence we return S_∞ (see Algorithm 3.1). If B_h is empty, then we apply Lemma 3.8 to solve the problem in time $O^*(2^{d_b(k-|Y|)})$. If B_h is non-empty, then we find a vertex $v \in W_h$ with the least in-neighbors in B_h. Suppose it has d_w of them.

We then branch into $d_w + 1$ branches described as follows. In the first d_w branches, we move a vertex u of B_h which is an in-neighbor of v, to the set Y. Each of these branches is equivalent to picking one of the in-neighbors of v from B_h in the solution. We then recourse on the resulting instance. In the last of the $d_w + 1$ branches, we delete from the instance non-terminals in B_h which dominate v and recourse on the resulting instance. Note that in the resulting instance of this branch, we have v in $W_l(Y)$.

Input : An instance (D, r, T, k) of DST, degree bound d_b, set Y
Output: A smallest solution of size at most k and containing Y for the instance (D, r, T, k) if it exists and S_∞ otherwise

Compute the sets B_h, B_l, W_h, W_l
if $|W_l| > d_b(k - |Y|)$ **then return** S_∞
else if $B_h = \emptyset$ **then**
  $S \leftarrow \text{NEDERLOF}(D, r, T \cup Y, W_l \cup Y) \cup Y$.
  if $|S| > k$ **then** $S \leftarrow S_\infty$
  return S
end
else
  $S \leftarrow S_\infty$
  Find vertex $v \in W_h$ with the least in-neighbors in B_h.
  for $u \in B_h \cap N^-(v)$ **do**
    $Y' \leftarrow Y \cup \{u\}$,
    $S' \leftarrow \text{DST-SOLVE}((D, r, T, k), d_b, Y')$.
    if $|S'| < |S|$ **then** $S \leftarrow S'$
  end
  $D' \leftarrow D \setminus (B_h \cap N^-(v))$
  $S' \leftarrow \text{DST-SOLVE}((D', r, T, k), d_b, Y)$.
  if $|S'| < |S|$ **then** $S \leftarrow S'$
  return S
end

Algorithm 3.1: Algorithm DST-SOLVE for DST on graphs excluding K_h as a minor

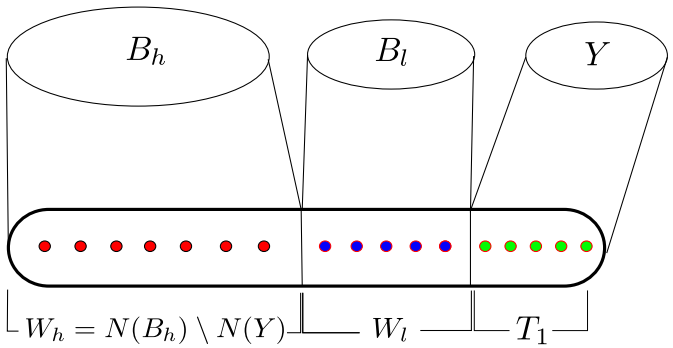

FIG. 2. *An illustration of the sets defined in Theorem 3.9*

Correctness. At each node of the recursion tree, we define a measure $\mu(I) = d_b(k - |Y|) - |W_l|$. We prove the correctness of the algorithm by induction on this measure. In the base case, when $d_b(k - |Y|) - |W_l| < 0$, then the algorithm is correct (by Lemma 3.7). Now, we assume as induction hypothesis that the algorithm is correct on instances with measure less than some $\mu \geq 0$. Consider an instance I such that $\mu(I) = \mu$. Since the branching is exhaustive, it is sufficient to show that the algorithm is correct on each of the child instances. To show this, it is sufficient to show that for each child instance I', $\mu(I') < \mu(I)$. In the first d_w branches, the size

of the set Y increases by 1, and the size of the set W_l does not decrease. Hence, in each of these branches, $\mu(I') \leq \mu(I) - d_b$. In the final branch, though the size of the set Y remains the same, the size of the set W_l increases by at least 1. Hence, in this branch, $\mu(I') \leq \mu(I) - 1$. Thus, we have shown that in each branch, the measure drops, hence completing the proof of correctness of the algorithm.

Analysis. Since D excludes K_h as a minor, Lemma 3.6, combined with the fact that we set $d_b = h - 2$, implies that $d_w^{\max} = ch^4$, for some c, is an upper bound on the maximum d_w which can appear during the execution of the algorithm. We first bound the number of leaves of the recursion tree as follows. The number of leaves is bounded by $\sum_{i=0}^{d_b k} \binom{d_b k}{i} (d_w^{\max})^{k - \frac{i}{d_b}}$. To see this, observe that each branch of the recursion tree can be described by a length-$d_b k$ vector as shown in the correctness paragraph. We then select i positions of this vector on which the last branch was taken. Finally for $k - \frac{i}{d_b}$ of the remaining positions, we describe which of the first at most $d_w^{\max}$ branches was taken. Any of the first d_w^{max} branches can be taken at most $k - \frac{i}{d_b}$ times if the last branch is taken i times.

The time taken along each root to leaf path in the recursion tree is polynomial, while the time taken at a leaf for which the last branch was taken i times is $O^*(2^{d_b(k-(k-\frac{i}{d_b}))+k-\frac{i}{d_b}}) = O^*(2^{i+k})$ (see Lemmata 3.7 and 3.8). Hence, the running time of the algorithm is

$$O^*\left(\sum_{i=0}^{d_b k} \binom{d_b k}{i} (d_w^{\max})^{k-\frac{i}{d_b}} \cdot 2^{i+k}\right) = O^*\left((2d_w^{\max})^k \cdot \sum_{i=0}^{d_b k} \binom{d_b k}{i} \cdot 2^i\right)$$
$$= O^*\left((2d_w^{\max})^k \cdot 3^{d_b k}\right).$$

For $d_b = h - 2$ and $d_w^{\max} = ch^4$ this is $O^*(3^{hk+o(hk)})$. This completes the proof of the theorem. □

LEMMA 3.10. *For every two function $g(n)$ and $h(k)$ with $g(n) = o(\log n)$, there is a function $f(k)$ such that for every k and n we have $2^{g(n)h(k)} \leq f(k) \cdot n$.*

Proof. We know that there is a function $f'(k)$ such that for every $n > f'(k)$ we have $g(n) < (\log n)/h(k)$. Now let $f(k)$ be the function defined as $f(k) = \max_{1 \leq n \leq f'(k)}\{2^{g(n)h(k)}\}$. Then, for every k we have that if $n \leq f'(k)$ then $2^{g(n)h(k)} \leq \max_{1 \leq n \leq f'(k)}\{2^{g(n)h(k)}\} = f(k)$ while for $n > f'(k)$ we have

$$2^{g(n)h(k)} \leq 2^{(\log n / h(k)) \cdot h(k)} = 2^{\log n} = n.$$

Hence, indeed, $2^{g(n)h(k)} \leq f(k) \cdot n$ for every n and k. □

Theorem 3.9 along with Lemma 3.10 has the following corollary.

COROLLARY 3.11. *If $\mathcal{C}$ is a class of digraphs excluding $o(\log n)$-sized minors, then* DST *parameterized by k is FPT on $\mathcal{C}$.*

3.2. DST **on graphs excluding topological minors.** We begin by observing that on graphs excluding K_h as a *topological minor*, we cannot apply Rule 1 since contractions may create new topological minors. Hence, we do not have the notion of a source terminal, which was crucial in designing the algorithm for this problem on graphs excluding minors. However, we will use a decomposition theorem of Grohe and Marx ([24], Theorem 4.1) to obtain a number of subproblems where we will be able to apply all the ideas developed in the previous subsection, and finally use a dynamic programming approach over this decomposition to combine the solutions to

the subproblems.

THEOREM 3.12. (*Global Structure Theorem*, [24]) *For every $h \in \mathbb{N}$, there exists constants $a(h)$, $b(h)$, $c(h)$, $d(h)$, and $e(h)$, such that the following holds. Let H be a graph on h vertices. Then, for every graph G with $H \not\preceq_T G$, there is a tree decomposition (M, β) of adhesion at most $a(h)$ such that for all $t \in V(M)$, one of the following three conditions is satisfied:*

1. *$|\beta(t)| \leq b(h)$.*
2. *$\tau(t)$ has at most $c(h)$ vertices of degree larger than $d(h)$.*
3. *$K_{e(h)} \not\preceq \tau(t)$.*

Furthermore, there is an algorithm that, given graphs G and H of sizes n and h, respectively, in time $f(h)n^{O(1)}$ for some computable function f, computes either such a decomposition (M, β) or a subdivision of H in G.

Let (M, β) be a tree decomposition given by the above theorem. Without loss of generality we assume that for every $t \in V(M)$ we have $r \in \beta(t)$. This might increase $a(h)$, $b(h)$, $c(h)$, and $e(h)$ by at most one. For the rest of this subsection we work with this tree decomposition.

THEOREM 3.13. DST *can be solved in time $O^*(q(h)^k)$ on graphs excluding K_h as a topological minor, for some computable function q.*

Proof. Our algorithm is based on dynamic programming over the tree decomposition (M, β) obtained as described above. For $t \in V(M)$ let $T_{\sigma(t)} = (T \cup \{r\}) \cap \sigma(t)$ and $T_{\gamma(t)} = (T \cup \{r\}) \cap \gamma(t)$. For every $t \in V(M)$ we have one table Tab_t indexed by (R, F), where $T_{\sigma(t)} \subseteq R \subseteq \sigma(t)$ and F is a set of arcs on R. The index of a table represents the way a possible solution tree can cross the cut-set $\sigma(t)$. In particular, the set R represents the intersection of the final solution tree $D[T \cup S \cup \{r\}]$ with $\sigma(t)$.

To explain the role of F consider the following situation. It may happen that for some vertex v in the considered part of the graph the path from r to v in the solution is partially contained in the considered part of the graph and partially outside it. Namely, the path can go through some vertex u of $\sigma(t)$ and the part between r and u can be within the considered part of the graph, but the part between u and v can be outside it. To model this situation we introduce the set F, which describes the connections "provided from outside", i.e., that we will be able to get from u to v.

Therefore, in the considered subgraph, we look for a set $S \subseteq \alpha(t)$ such that in the digraph $D[T_{\gamma(t)} \cup R \cup S] \cup F$ there is a directed path from r to every $t' \in T_{\gamma(t)} \cup R$. That is, the partial solution set S is required to provide a directed path from r to all terminals which are inside the considered part of the graph and to the vertices of R, but it can use the arcs of F as a help. If in the digraph $D[T_{\gamma(t)} \cup R \cup S] \cup F$ there is a directed path from r to every $t' \in T_{\gamma(t)} \cup R$, then we say that S is *good* for t, R, F.

For each index R, F we store in $\mathsf{Tab}_t(R, F)$ one good set S of minimum size. If no such set exists, or $|S| > k$ for any such set, we set $|\mathsf{Tab}_t(R, F)| = \infty$ and use the dummy symbol S_∞ in place of the set. Naturally, $S_\infty \cup S = S_\infty$ for any set S. Furthermore, if $\mathsf{Tab}_t(R, F) = S \neq S_\infty$ we let $\kappa_t(R, F)$ be the set of arcs on R such that $(u, v) \in \kappa_t(R, F)$ if and only if there is a directed path from u to v in $D[T_{\gamma(t)} \cup R \cup \mathsf{Tab}_t(R, F)]$. Note also, that if $|\mathsf{Tab}_t(R, F)| = 0$ then $\kappa_t(R, F)$ only depends on R, not on F.

Let us now explain the role of $\kappa_t(R, F)$. We cannot afford to require the solution outside the current subgraph to actually provide exactly the connections in F as we guessed. This would no longer resemble an instance of DST but rather of DIRECTED STEINER NETWORK, which is a more general problem. Therefore, we use $\kappa_t(R, F)$ to represent the connections provided to the outside by a (particular) solution for the

current subgraph for given R and F. One would expect that $\kappa_t(R, F)$ would be in certain sense complementary to F. Any intersection of these two sets show certain inefficiency of the partial solution (or rather of F guessed).

As σ of the root node of M is $\emptyset$, the only entry of Tab for root is an optimal Steiner tree in D. Let us denote by $g(h)$ the maximum number of entries of the table Tab_t over $t \in V(M)$. It is easy to see that $g(h) \leq 2^{a(h)+a(h)^2}$.

The algorithm to fill the tables proceeds bottom-up along the tree decomposition and we assume that by the time we start filling the table for t, the tables for all its proper descendants have already been filled. We now describe the algorithm to fill the table for t, distinguishing three cases, based on the type of node t (see Theorem 3.12).

3.2.1. Case 1: $\tau(t)$ has at most $c(h)$ vertices of degree larger than $d(h)$. In this case we use Algorithm 3.2. Let us first describe it on high level. For each R and F it first removes the irrelevant parts of the graph and then branches on the non-terminal vertices of high degree. Following that, it invokes Algorithm 3.3 which does the main work.

The aim would now be to guess R_s and F_s for each child s of t and replace the subgraph corresponding to s and its descendants by $\kappa_s(R_s, F_s)$. In particular, among the vertices in $\sigma(s)$ we would only keep vertices in R_s and make R_s part of the set of terminals. However, note that, since t can have an unbounded number of children in M we cannot afford to make the guess for each of them. Hence SatisfyChildrenSD only branches (i.e. makes the guess) for the children which need at least one private vertex of the solution. Furthermore, the selected solution for one child might contain a vertex of $\sigma(s)$ (i.e., $\sigma(s) \setminus R_s \neq \emptyset$) which may actually help the solution also in other children.

After a solution is selected for all children for which we can afford to do so, we are left with a graph with vertex set formed by $\beta(t)$ and $\alpha(s)$ for the children s which do not need any additional private vertex of the solution. We apply Rule 1 exhaustively on it. Although the parts of the graph corresponding to the children might contain vertices of high degree, we show that each of these parts only contain at most $|\sigma(s)|$ source terminals. Hence, if the number of obtained source terminals is too big, then there is no solution for the branch. Otherwise we use the modified algorithm of Nederlof as described in Lemma 3.8.

Let us now describe the pseudocodes in more detail, starting with Algorithm 3.2. On line 3 we restrict the graph to the parts that we are allowed to use. On line 4 we set the initial solution, to later take the minimum over many branches. Line 5 identifies the vertices of high degree, except for those that are already surely part of the solution. On line 6 we branch on vertices of B which should be taken into the solution and on line 7 we remove those not selected from the graph. Lines 8–10 adjust the tree decomposition to form a tree decomposition of the digraph D''. Line 11 is the actual call to Algorithm 3.3. The first four arguments represent an instance of DST, while the last two represent a tree decomposition for the graph. In the instance we put the selected high degree vertices and the vertices of $R \setminus \{r\}$ into the set of terminals, as we only want to consider solutions containing them in this branch. Thus, we also decrease the size of the solution by $|Y|$ as we add the set Y to the result of the call. Next on line 13 we compare the new solution set to the one obtained previously, and if the new one is better we store it. Finally, on line 15 we save the best solution to the table.

Algorithm 3.3 first checks the obvious constraint on line 1. Then on line 2 it checks, whether there is a child (remaining) that needs some private vertex of the

```
1  foreach R with T_{σ(t)} ⊆ R ⊆ σ(t) do
2  foreach F ⊆ R^2 do
3  |  D' ← D[α(t) ∪ R] ∪ F.
4  |  S ← S_∞.
5  |  B ← {v | v ∈ (β(t) ∩ V(D')) \ (T ∪ R) & deg_{τ(t)}(v) > d(h)}.
6  |  foreach Y ⊆ B with |Y| ≤ k  do
7  |  |  D'' ← D' \ (B \ Y).
8  |  |  M' ← subtree of M rooted at t.
9  |  |  Let β' : V(M') → 2^{V(D'')} be such that β'(s) = β(s) ∩ V(D'')
10 |  |                for every s in V(M').
11 |  |  S' ← SatisfyChildrenSD(D'', r, T_{γ(t)} ∪ Y ∪ R \ {r}, k − |Y|, M', β')
12 |  |                ∪Y.
13 |  |  if |S'| < |S| then S ← S'.
14 |  end
15 |  Tab_t(R, F) ← S.
16 end
```

Algorithm 3.2: Algorithm SmallDeg to fill Tab_t if all but few vertices of the bag have small degrees.

solution. This is checked by inspecting the table for R_s equal to the intersection of the current terminal set with $\sigma(s)$ (i.e. we add no vertices of $\sigma(s)$ to the solution) and F_s providing connections to all vertices in R_s. If this makes the child "satisfied", i.e., the stored solution for this case is an empty set, we cannot afford to branch on this child. In certain sense, this index represents the most we can provide from outside without expenses. See Observation 3.16 for an explanation why this is the right index to inspect.

If we can branch on the child, then we set the initial solution on line 3 and then branch into all possible R' and F'. We want to add the solution stored in $\mathsf{Tab}_s(R', F')$ and the vertices in $(R' \setminus T'_{\sigma(s)})$ to the solution we get from the recursive call. Therefore, we check on line 6 whether these two sets together are not too big, and otherwise we skip the recursive call.

On lines 7–10 we prepare the instance for the recursive call. Namely, on line 7 we remove the child—more precisely $\alpha(s)$ and the unselected vertices from $\sigma(s)$. On line 8 we add the connections provided by the solution in the child. Note that this is the only place where the set F' (at least indirectly) influences the instance for the recursive call. On line 9 we update the terminal set by removing terminals in the child and adding the vertices that we selected to be in the solution. Then on line 10 the budget is adjusted. Lines 11–13 adjust the tree decomposition to form a tree decomposition of the digraph D''. The line 14 constitutes the actual recursive call and on line 15 we add the vertices selected for the child to the solution obtained. If no child needs its private vertex in the solution, then the else branch on lines 19–22 is executed, as described earlier. The correctness of the check on line 21 is justified by Lemma 3.18.

For the proof of the correctness of the algorithm, we need several observations and lemmas.

The first observation shows that we can model taking certain vertices into a solution by adding them into the terminal set.

Input : An instance (D', r, T', k) of DST, a tree decomposition (M, β) rooted at t

Output: A smallest solution to the instance or S_∞ if all solutions are larger than k

if $k < 0$ **then return** S_∞
if $\exists s$ *child of* t *such that* $|\mathsf{Tab}_s(T'_{\sigma(s)}, \{r\} \times (T' \cap \sigma(s))| > 0$ **then**
  $S \leftarrow S_\infty$.
  foreach R' *s.t.* $T'_{\sigma(s)} \subseteq R' \subseteq \sigma(s)$ **do**
  foreach $F' \subseteq (R')^2$ **do**
  if $|\mathsf{Tab}_s(R', F') \cup (R' \setminus T'_{\sigma(s)})| \leq k$ **then**
    $\hat{D} \leftarrow D' \setminus (\gamma(s) \setminus R')$.
    $D'' \leftarrow \hat{D} \cup \kappa_s(R', F')$.
    $T'' \leftarrow (T' \cap V(D'')) \cup (R' \setminus T'_{\sigma(s)})$.
    $k' \leftarrow k - |\mathsf{Tab}_s(R', F')| - |R' \setminus T'_{\sigma(s)}|$.
    $M' \leftarrow M$ with the subtree rooted at s removed.
    Let $\beta' : V(M') \to 2^{V(D'')}$ be such that $\beta'(s) = \beta(s) \cap V(D'')$
      for every s in $V(M')$.
    $S' \leftarrow$ SatisfyChildrenSD $(D'', r, T'', k', M', \beta') \cup$
      $\mathsf{Tab}_s(R', F') \cup (R' \setminus T'_{\sigma(s)})$.
    if $|S'| < |S|$ **then** $S \leftarrow S'$.
  end
else
  Apply Rule 1 exhaustively to (D', r, T', k) to obtain (D'', r, T'', k).
  Denote by T_0 the source terminals in (D'', r, T'', k).
  if $|T_0| > k \cdot \max\{d(h), a(h)\}$ **then** $S \leftarrow S_\infty$.
  else $S \leftarrow$ Nederlof (D'', r, T'', T_0).
end
if $|S| > k$ **then** $S \leftarrow S_\infty$.
return S.

Algorithm 3.3: Function SatisfyChildrenSD (D', r, T', k, M, β) doing the main part of the work of Algorithm 3.2.

Observation 3.14. *Let (D, r, T, k) be an instance of DST. For every x in $V \setminus (T \cup \{r\})$ we have that S is a solution for $(D, r, T \cup \{x\}, k-1)$ if and only if $S \cup \{x\}$ is a solution for (D, r, T, k) and x is reachable from r in $D[T \cup S \cup \{x, r\}]$. In particular, if $S \cup \{x\}$ is a minimal solution for (D, r, T, k), then S is a minimal solution for $(D, r, T \cup \{x\}, k-1)$.*

Proof. On both sides of the equivalence we ask for the vertices of $T \cup \{x\}$ to be reachable from r in $D[T \cup S \cup \{x, r\}]$. If x is not reachable from r in $D[T \cup S \cup \{x, r\}]$, then $S \cup \{x\}$ is not minimal. Finally, if S is not minimal, i.e., $S \setminus \{y\}$ is also a solution for $(D, r, T \cup \{x\}, k-1)$, then $S \cup \{x\} \setminus \{y\}$ is also a solution for (D, r, T, k) and $S \cup \{x\}$ is not minimal. □

The following lemma shows how to obtain a solution for the current instance using the solution returned from a recursive call.

Lemma 3.15. *Let (D', r, T', k) be an instance of DST and M, β be a tree decomposition for D rooted at t. Let s be a child of t for which $|\mathsf{Tab}_s(T'_{\sigma(s)}, \{r\} \times (T' \cap \sigma(s))| > 0$*

(i.e., the condition on line 2 of Algorithm 3.3 is satisfied). Let R' be some subset of $\sigma(s)$ such that $T'_\sigma \subseteq R'$ and F' be some set of arcs on R'. Let (D'', r, T'', k') be such that $D'' = (D' \setminus (\gamma(s) \setminus R')) \cup \kappa_s(R', F')$, $T'' = (T' \cap V(D'')) \cup (R' \setminus T'_{\sigma(s)})$, and $k' = k - |\mathsf{Tab}_s(R', F')| - |R' \setminus T'_{\sigma(s)}|$ (i.e., the instance formed by lines 7–10 of the algorithm on s for R' and F'). If S is a solution for (D'', r, T'', k') then $S' = S \cup \mathsf{Tab}_s(R', F') \cup (R' \setminus T'_{\sigma(s)})$ is a solution for (D', r, T', k).

Proof. Assume S is a solution for (D'', r, T'', k'). As $k' = k - |\mathsf{Tab}_s(R', F')| - |R' \setminus T'_{\sigma(s)}|$ and $|S| \leq k'$ it is $|S'| \leq k$. We have to show that every vertex t' in T' is reachable from r in $D'[T' \cup \{r\} \cup S']$. Note that, since $(T' \cap \sigma(s)) \subseteq R'$ and, thus, $T'' = (T' \cap V(D'')) \cup (R' \setminus T'_{\sigma(s)}) \supseteq (T' \setminus (\gamma(s) \setminus R')) = (T' \setminus \alpha(s))$, we have $T' \subseteq T'' \cup (T' \cap \alpha(s))$.

For a vertex t' in T'' there is a path from r to t' in $D''[T'' \cup \{r\} \cup S]$ as S is a solution for (D'', r, T'', k'). If this path contains an arc (u, v) in $\kappa_s(R', F')$, then there is a path from u to v in $D[T'_{\gamma(s)} \cup R' \cup \mathsf{Tab}_s(R', F')]$ by the definition of $\kappa_s(R', F')$ and we can replace the arc (u, v) with this path, obtaining (possibly after shortcutting) a path in $D'[T' \cup \{r\} \cup S']$.

For a vertex t' in $T' \cap \alpha(s)$ there is a path P from r to t' in $D[T'_{\gamma(s)} \cup R' \cup \mathsf{Tab}_s(R', F')] \cup F'$ as Tab_s was filled correctly and, thus, $\mathsf{Tab}_s(R', F')$ is good for s, R', F'. Let r' be the last vertex of R' on P (note that $r \in R'$ and $\alpha(s) \cap R' = \emptyset$). Since $R' \subseteq T''$ we can replace the part of P from r to r' by a path in $D'[T' \cup \{r\} \cup S']$ obtained in the previous step (possibly trivial). This way we get (possibly after shortcutting) a path from r to t' in $D'[T' \cup \{r\} \cup S']$ as required. □

The next observation partially explains, why we select the particular index to check on line 2 of Algorithm 3.3. Furthermore, it shows that adding more vertices to the set R does not increase the size of the solution (for the particular type of set F).

Observation 3.16. *If s is a child of t and there are some R' and F' such that $|\mathsf{Tab}_s(R', F')| = 0$, then every vertex in $T' \cap \gamma(s)$ is reachable from some vertex in R' in $D'[T'_{\gamma(s)} \cup R']$. Furthermore, if there is an F' such that $|\mathsf{Tab}_s(R', F')| = 0$, then $|\mathsf{Tab}_s(R', \{r\} \times (R' \setminus \{r\})| = 0$, and also $|\mathsf{Tab}_s(R'', \{r\} \times (R'' \setminus \{r\})| = 0$ for every $R' \subseteq R'' \subseteq \sigma(s)$.*

Proof. By definition of Tab_s, if $|\mathsf{Tab}_s(R', F')| = 0$, then $\emptyset$ is good for s, R', F'. This means that in the digraph $D[T_{\gamma(s)} \cup R'] \cup F'$ there is a directed path from r to every $t' \in T_{\gamma(s)} \cup R'$. As F' is a set of arcs on R', every vertex in $T' \cap \gamma(s)$ is reachable from some vertex in R' in $D'[T'_{\gamma(s)} \cup R']$, proving the first part. But then, if $R' \subseteq R'' \subseteq \sigma(s)$ and $F'' = \{r\} \times (R'' \setminus \{r\})$, then in the digraph $D[T_{\gamma(s)} \cup R''] \cup F''$ there is a directed path from r to every $t' \in T_{\gamma(s)} \cup R''$, proving the second part. □

The following observation shows that if the condition on line 2 of Algorithm 3.3 is not satisfied, then every source terminal is represented within $\beta(t)$.

Observation 3.17. *Suppose the condition on line 2 of Algorithm 3.3 is not satisfied (i.e., for every child s of t we have $|\mathsf{Tab}_s(T'_{\sigma(s)}, \{r\} \times (T' \cap \sigma(s))| = 0$), (D'', r, T'', k) is obtained from (D', r, T', k) by exhaustive application of Rule 1 and T_0 is the set of source terminals. Let $t_0 \in T_0$ be obtained by contracting a strongly connected component C of $D'[T']$. If there is a vertex $u \in C \cap \gamma(s)$ for some child s of t, then there is a vertex $v \in C \cap \sigma(s)$.*

Proof. Since the condition is not satisfied, it follows from Observation 3.16 that there is a path from some $v \in \sigma(s)$ to u in $D'[T' \cup \{r\}]$. Since t_0 is a source terminal, this path has to be fully contained in C. □

The next lemma gives a bound on the number of source terminals dominated by

a single vertex of the graph, once the condition is not satisfied.

LEMMA 3.18. *Suppose the condition on line 2 of Algorithm 3.3 is not satisfied (i.e., for every child s of t we have $|\mathsf{Tab}_s(T'_{\sigma(s)}, \{r\} \times (T' \cap \sigma(s))| = 0$), (D'', r, T'', k) is obtained from (D', r, T', k) by exhaustive application of Rule 1 and T_0 is the set of source terminals. A vertex x in $\alpha(s) \setminus (T' \cup \{r\})$ for some child s of t can dominate at most $a(h)$ vertices of T_0. A vertex x in $\beta(t) \setminus (T' \cup \{r\})$ can dominate at most $deg_{\tau(t)}(x)$ vertices of T_0.*

Proof. If $x \in V(D') \setminus T'$ dominates a vertex $t_0 \in T_0$, then t_0 was obtained by contracting some strongly connected component C of $D'[T']$ and there is a vertex $y \in C \cap N^+_{D'}(x)$. If x is in $\alpha(s)$, then $N^+_{D'}(x) \subseteq \gamma(s)$, C contains a vertex of $\sigma(s)$ due to Observation 3.17, and, hence, there can be at most $a(h)$ such t_0's. If x is in $\beta(t)$, then either y is also in $\beta(t)$, in which case the edge xy is in $\tau(t)$, or y is in $\alpha(s)$ for some child s of t. In this case x is in $\sigma(s)$, C contains a vertex y' of $\sigma(s)$ due to Observation 3.17, and we can account t_0 to the edge xy' of $\tau(t)$. □

The following lemma shows that, if (D', r, T', k) is a yes-instance and the condition is satisfied, then at least one recursive call will return a solution.

LEMMA 3.19. *Let (D', r, T', k) be an instance of DST and (M, β) be a tree decomposition for D' rooted at t. Let s be a child of t for which $|\mathsf{Tab}_s(T'_{\sigma(s)}, \{r\} \times (T' \cap \sigma(s))| > 0$ (i.e., the condition on line 2 of Algorithm 3.3 is satisfied). Let S be a solution for (D', r, T', k), $R' = (T' \cup S \cup \{r\}) \cap \sigma(s)$ and F' be the set of arcs (u, v) on R such that v is reachable from u in $D'[(T' \cup S \cup \{r\}) \setminus \alpha(s)]$. Let (D'', r, T'', k') be such that $D'' = (D' \setminus (\gamma(s) \setminus R')) \cup \kappa_s(R', F')$, $T'' = (T' \cap V(D'')) \cup (R' \setminus T'_{\sigma(s)})$, and $k' = k - |\mathsf{Tab}_s(R', F')| - |R' \setminus T'_{\sigma(s)}|$ (i.e., the instance formed by lines 7–10 of Algorithm 3.3 on s for R' and F'). Then $(S \setminus \alpha(s)) \setminus (R' \setminus T'_{\sigma(s)})$ is a solution for (D'', r, T'', k'), while $(S \cap \alpha(s))$ is good for s, R', F'.*

Proof. We start by the second claim. As S is a solution for (D', r, T', k), in $D'[T' \cup S \cup \{r\}]$ there is a path from r to every $t' \in (T' \cup S) \cap \gamma(s) \supseteq T'_{\gamma(s)} \cup R' \setminus \{r\}$. If such a path contains a part formed by vertices in $V(D') \setminus \gamma(s)$, then there is an arc in F' between the last vertex of $R' = (T' \cup S \cup \{r\}) \cap \sigma(s)$ before the part and the first vertex of R' after the part and we can replace the part of the path by the arc. Therefore, there is a path from r to every $t' \in T'_{\gamma(s)} \cup R' \setminus \{r\}$ in $D[T'_{\gamma(s)} \cup R' \cup (S \cap \alpha(s))] \cup F'$. Hence, indeed, $(S \cap \alpha(s))$ is good for s, R', F' and $|S \cap \alpha(s)| \geq |\mathsf{Tab}_s(R', F')|$. It also follows that $|(S \setminus \alpha(s)) \setminus (R' \setminus T'_{\sigma(s)})| = |S| - |S \cap \alpha(s)| - |R' \setminus T'_{\sigma(s)}| \leq k - |\mathsf{Tab}_s(R', F')| - |R' \setminus T'_{\sigma(s)}| = k'$.

Next we show that every vertex of $(T' \cup S) \setminus \alpha(s)$ is reachable from r in $\widetilde{D} = D'[(T' \cup S \cup \{r\}) \setminus \alpha(s)] \cup \kappa_s(R', F')$. For every vertex $t' \in R'$ there is a path from r to t' in $D'[T'_{\gamma(s)} \cup R' \cup \mathsf{Tab}_s(R', F')] \cup F'$ since Tab_s is correctly filled. Replacing the parts of this path in $\alpha(s)$ by arcs of $\kappa_s(R', F')$ and arcs of F' by paths in $D'[(T' \cup S \cup \{r\}) \setminus \alpha(s)]$ one obtains a path in $\widetilde{D}$ to every vertex of R'. Now for every vertex t' in $(T' \cup S) \setminus \alpha(s)$ there is a path P from r to t' in $D'[T' \cup S \cup \{r\}]$. Let r' be the last vertex of P in R' (note that r is in R'). Then concatenating the path from r to r' (possibly trivial) obtained in the previous step with the part of P between r' and t' we get a path from r to t' in $\widetilde{D}$.

As $D''[(T' \setminus \alpha(s)) \cup (S \setminus \alpha(s)) \cup \{r\}] = D''[(T' \cup S \cup \{r\}) \setminus \alpha(s)] = ((D' \setminus (\gamma(s) \setminus R')) \cup \kappa_s(R', F'))[(T' \cup S \cup \{r\}) \setminus \alpha(s)] = D'[(T' \cup S \cup \{r\}) \setminus \alpha(s)] \cup \kappa_s(R', F') = \widetilde{D}$, we have shown that $S \setminus \alpha(s)$ is a solution for $(D'', r, T' \setminus \alpha(s), k - |\mathsf{Tab}_s(R', F')|)$. Moreover each vertex of $(R' \setminus T'_{\sigma(s)})$ is reachable from r in $\widetilde{D}$. Also we have $T'' =$

$(T' \cap V(D'')) \cup (R' \setminus T'_{\sigma(s)}) = (T' \setminus (\gamma(s) \setminus R')) \cup (R' \setminus T'_{\sigma(s)}) = (T' \setminus \alpha(s)) \cup (R' \setminus T'_{\sigma(s)})$, since $(T' \cap \sigma(s)) \subseteq R'$. Therefore, it remains to move the vertices of $(R' \setminus T'_{\sigma(s)})$ one by one into the terminal set using Observation 3.14 to show that $(S \setminus \alpha(s)) \setminus (R' \setminus T'_{\sigma(s)})$ is a solution for (D'', r, T'', k'). □

The next lemma shows the correctness of Algorithm 3.3 (SatisfyChildrenSD).

Lemma 3.20. *Let (D', r, T', k) be an instance of DST and let (M, β) be a tree decomposition for D'. If there is a solution S of size at most k for (D', r, T', k), then the invocation of* SatisfyChildrenSD(D', r, T', k, M, β) *returns a set S' not larger than S which is a solution for (D', r, T', k).*

Proof. We prove the claim by induction on the depth of the recursion. Note that the depth is bounded by the number of children of t in M. Suppose first that the condition on line 2 is not satisfied and $|T_0| > k \cdot \max\{d(h), a(h)\}$. As no vertex can dominate more than $\max\{d(h), a(h)\}$ vertices of T_0 by Lemma 3.18, there is a vertex of T_0 not dominated by S, which is a contradiction. If $|T_0| \leq k \cdot \max\{d(h), a(h)\}$, it follows from the optimality of the modified Nederlof's algorithm (see Lemma 3.5) and Observation 3.3 that $|S'| \leq |S|$ and S' is a solution for (D', r, T', k).

Now suppose that the condition on line 2 is satisfied and let s be a child of t for which $|\mathsf{Tab}_s(T'_{\sigma(s)}, \{r\} \times (T' \cap \sigma(s))| > 0$. Let R', F' and (D'', r, T'', k') be as in Lemma 3.19. Then $S \setminus (\alpha(s) \cup (R' \setminus T'_{\sigma(s)}))$ is a solution for (D'', r, T'', k'), while $S \cap \alpha(s)$ is good for s, R', F'. Therefore $|S \cap \alpha(s)| \geq |\mathsf{Tab}_s(R', F')|$ as Tab_s is filled correctly by assumption. Moreover SatisfyChildrenSD$(D'', r, T'', k', M, \beta)$ will return a set S' with $|S'| \leq |S \setminus (\alpha(s) \cup (R' \setminus T'_{\sigma(s)}))| \leq |S| - |\mathsf{Tab}_s(R', F')| - |(R' \setminus T'_{\sigma(s)})|$ due to the induction hypothesis. Together we get that $|S' \cup \mathsf{Tab}_s(R', F') \cup (R' \setminus T'_{\sigma(s)})| \leq |S|$ and therefore also the set returned by SatisfyChildrenSD is not larger than S. It follows from Lemma 3.15 and the induction hypothesis that the set returned is a solution for (D', r, T', k). □

Now we are ready to prove the correctness of Algorithm 3.2. Let D', D'', M', and β' be as in Algorithm 3.2. We first show that if the algorithm stores a set $S \neq S_\infty$ in $\mathsf{Tab}_t(R, F)$, then $S \subseteq \alpha(t)$ and in the digraph $D[T_{\gamma(t)} \cup R \cup S] \cup F$ there is a directed path from r to every $t' \in T_{\gamma(t)} \cup R$. The set S' returned by SatisfyChildrenSD$(D'', r, T_{\gamma(t)} \cup Y \cup R \setminus \{r\}, k - |Y|, M', \beta')$ is a solution for $(D'', r, T_{\gamma(t)} \cup Y \cup R \setminus \{r\}, k - |Y|)$ by Lemma 3.20. We can move the vertices of Y one by one from $T_{\gamma(t)} \cup Y \cup R \setminus \{r\}$ to S' using Observation 3.14 to show that $S' \cup Y$ is a solution for $(D'', r, T_{\gamma(t)} \cup R \setminus \{r\}, k)$. But this means that in $D''[T_{\gamma(t)} \cup R \cup S' \cup Y] = (D' \setminus (B \setminus Y))[T_{\gamma(t)} \cup R \cup S' \cup Y] = D'[T_{\gamma(t)} \cup R \cup S' \cup Y] = (D[\alpha(t) \cup R] \cup F)[T_{\gamma(t)} \cup R \cup S' \cup Y] = D[T_{\gamma(t)} \cup R \cup S' \cup Y] \cup F$ there is a path from r to every $t' \in T_{\gamma(t)} \cup R \setminus \{r\}$ as required.

In order to prove that the set stored is minimal assume that there is a set $S \subseteq \alpha(t)$ of size at most k which is good for t, R, F. Let $D' = D[\alpha(t) \cup R] \cup F$, $B = \{v \mid v \in (\beta(t) \cap V(D')) \setminus (T \cup R) \& \deg_{\tau(t)}(v) > d(h)\}$, $Y = S \cap B$ and $D'' = D' \setminus (B \setminus Y)$. Without loss of generality we can assume that S is minimal and, therefore, $S \setminus Y$ is a solution for $(D'', r, T \cup Y \cup R \setminus \{r\}, k - |Y|)$ by Observation 3.14. Hence SatifyChildrenSD$(D'', r, T \cup Y \cup R \setminus \{r\}, k - |Y|, M', \beta')$ returns a set S' not larger than $S \setminus Y$ due to Lemma 3.20 and the set stored in $\mathsf{Tab}_t(R, F)$ is not larger than $|S' \cup Y| \leq |S \setminus Y| + |Y| = |S|$ finishing the proof of correctness.

As for the time complexity, observe first, that the bottleneck of the running time of Algorithm 3.2 are the at most $2^{c(h)}$ calls of Algorithm 3.3. Therefore, we focus our attention on the running time of Algorithm 3.3. Note that in each recursive call of SatisfyChildrenSD, by Observation 3.16, as the condition on line 2 is satisfied,

either $|\mathsf{Tab}_s(R', F')| > 0$ or $|R' \setminus T'_{\sigma(s)}| > 0$ and thus, $k' < k$. There are at most $g(h)$ recursive calls for one call of the function. The time spent by SatisfyChildrenSD on instance with parameter k is at most the maximum of $g(h)$ times the time spent on instances with parameter $k-1$ and the time spent by the modified algorithm of Nederlof on an instance with at most $k \cdot \max\{d(h), a(h)\}$ source terminals. As the time spent for $k < 0$ is constant, we conclude that the running time in Case 1 can be bounded by $O^*((\max\{g(h), 2^{\max\{d(h),a(h)\}}\})^k)$.

3.2.2. Case 2: $K_{e(h)} \not\preceq \tau(t)$. The overall strategy in this case is similar to that in the previous case. Nevertheless, this time there are no special vertices that we would have to deal with beforehand. Therefore, basically all the work is done by Algorithm 3.4 (SatisfyChildrenMF()), which is a slight modification of the function SatisfyChildrenSD. The modification is limited to the else branch of the condition on line 2, that is, to lines 19–22, where Algorithm 3.1 (developed in Section 3.1) is used instead of the modified version of Nederlof's algorithm. Again the parts of the graph corresponding to the children which do not need any additional private vertex of the solution can contain $K_{e(h)}$ as a minor. However, we show that the relevant part of the graph which is used to quide the branching in Algorithm 3.1 is a minor of $\tau(t)$ and, hence, the running time bound of Algorithm 3.1 applies.

For every R and F we now simply store in $|\mathsf{Tab}_t(R, F)|$ the result of SatisfyChildrenMF$(D', r, T_{\gamma(t)} \cup R \setminus \{r\}, k, M', \beta')$, where $D' = D[\alpha(t) \cup R] \cup F$, that is, the parts of the graph that we are allowed to use, M' is the subtree of M rooted at t and $\beta' : V(M') \to 2^{V(D')}$ is such that $\beta'(s) = \beta(s) \cap V(D')$ for every s in $V(M')$.

```
18 else
19 |  Apply Rule 1 exhaustively to (D', r, T', k) to obtain (D'', r, T'', k).
20 |  S ← DST-solve((D'', r, T'', k), max{e(h) − 2, a(h)}, ∅).
21 end
22 return S
```

Algorithm 3.4: Part of the function SatisfyChildrenMF (D', r, T', k, M, β) which differs from the appropriate part of the function SatisfyChildrenSD.

For the analysis, we need most of the lemmata proved for Case 1. To prove a running time upper bound we also need the following lemma.

Lemma 3.21. *Suppose the condition on line 2 of Algorithm 3.4 is not satisfied, (D'', r, T'', k) is obtained from (D', r, T', k) by exhaustive application of Rule 1, T_0 is the set of source terminals, $B_h = B_h(\emptyset, \max\{e(h)-2, a(h)\})$ is the set of non-terminals with degree at least $\max\{e(h)-1, a(h)+1\}$ in T_0 and $W_h = W_h(\emptyset, \max\{e(h)-2, a(h)\})$ is the set of source terminals with an in-neighbor in B_h. Then $D''[B_h \cup W_h]$ is a minor of $\tau(t)$.*

Proof. As proven in Lemma 3.18, the non-terminals in $\gamma(t) \setminus \beta(t)$ can only dominate at most $a(h)$ vertices in T_0. Therefore $B_h \subseteq \beta(t)$. By Lemma 3.17, each vertex t' in T_0 was obtained by contracting a strongly connected component $C(t')$ which contains at least one vertex of $\beta(t)$. Finally, if $b \in B_h$ dominates $w \in W_h$, but there is no edge between b and $\beta(t) \cap C(w)$ in D', then there is a child s of t such that $b \in \sigma(s)$, $C(w) \cap \alpha(s) \neq \emptyset$, thus there is $y \in C(w) \cap \sigma(s)$ and by is an edge of $\tau(t)$ as $\sigma(s)$ is a clique in $\tau(t)$. By the same argument $C(w) \cap \tau(t)$ is connected for every $w \in W_h$ and $D''[B_h \cup W_h]$ has a model in $\tau(t)$. □

The proof of correctness of the algorithm is similar to that in Case 1.

We first prove by induction on the depth of the recursion that if SatisfyChildrenMF(D', r, T', k, M, β) returns a set $S \neq S_\infty$ then S is a smallest solution for (D', r, T', k). If the condition on line 2 is not satisfied (and hence there is no recursion) the claim follows from the correctness of the algorithm DST-solve proved in Section 3.1. If the condition is satisfied, it follows from Lemma 3.15 and the induction hypothesis that the set returned is indeed a solution. The minimality in this case is proved exactly the same way as in Lemma 3.20.

It remains to prove that being a solution for $(D', r, T_{\gamma(t)} \cup R \setminus \{r\}, k)$ is the same as being good for t, R, F. A set $S \subseteq \alpha(t)$ is a solution for $(D', r, T_{\gamma(t)} \cup R \setminus \{r\}, k)$ if there is a path from r to every $t' \in T \cup R \setminus \{r\}$ in $D'[T_{\gamma(t)} \cup R \cup S] = D[(\alpha(t) \cup R) \cap (T_{\gamma(t)} \cup R \cup S)] \cup F = D[T_{\gamma(t)} \cup R \cup S] \cup F$. This is, however, exactly the definition of being good for t, R, F.

As for the time complexity, let us first find a bound $d_w^{\max}$ for DST-solve in the case the condition on line 2 is not satisfied. Lemma 3.21 implies that $K_{e(h)} \not\preceq D''[B_h \cup W_h]$ in this case and the sets B_h and W_h only get smaller during the execution of DST-solve. Using Lemma 3.6, we can derive an upper bound $d_w^{\max} \leq c \cdot e(h)^4$ for some constant c. It follows then from the proof in Section 3.1 that DST-solve runs in $O^*((2^{O(d_b)} \cdot d_w^{\max})^k)$ time and, as we use the algorithm with $d_b = \max\{e(h) - 2, a(h)\}$, there is a constant $g'(h)$, such that the running time of DST-solve can be bounded by $O^*((g'(h))^k)$. From this, similarly as in Case 1, it is easy to conclude, that the running time of the overall algorithm for Case 2 can be bounded by $O^*((\max\{g(h), g'(h)\})^k)$.

3.2.3. Case 3: $|\beta(t)| \leq b(h)$. If $|\beta(t)| \leq b(h)$, then no vertex in $\tau(t)$ has degree larger than $b(h) - 1$, and $K_{b(h)+1} \not\preceq \tau(t)$. Therefore, in this case, either of the two approaches described above can be used. This completes the proof of Theorem 3.13. □

3.3. DST **on d-degenerate graphs.** Since DST has an $O^*(f(k, h))$ algorithm on graphs excluding minors and topological minors, a natural question is if DST has a $O^*(f(k, d))$ algorithm on d-degenerate graphs. However, we show that in general, we cannot expect an algorithm of this form even for an arbitrary 2-degenerate graph.

Theorem 3.22. DST *parameterized by k is W[2]-hard on 2-degenerate graphs.*

Proof. The proof is by a parameterized reduction from Set Cover. Given an instance $(\mathcal{U}, \mathcal{F} = \{F_1, \ldots, F_m\}, k)$ of Set Cover, we construct an instance of DST as follows. Corresponding to each set F_i, we have a vertex f_i and corresponding to each element $u \in \mathcal{U}$, we add a directed cycle C_u of length l_u where l_u is the number of sets in $\mathcal{F}$ which contain u (see Fig. 3). For each cycle C_u, we add an arc from each of the sets containing u, to a unique vertex of C_u. Since C_u has l_u vertices, this is possible. Finally, we add another directed cycle C of length $m+1$ and for each vertex f_i, we add an arc from a unique vertex of C to f_i. Again, since C has length $m+1$, this is possible. Finally, we set as the root r, the only remaining vertex of C which does not have an arc to some f_i and we set as terminals all the vertices involved in a directed cycle C_u for some u and all the vertices in the cycle C except the root r. It is easy to see that the resulting digraph has degeneracy 3. Finally, we subdivide every edge which lies in a cycle C_u for some u, or on the cycle C and add the new vertices to the terminal set. This results in a digraph D of degeneracy 2. Let T be the set of terminals as defined above. This completes the construction. We claim that $(\mathcal{U}, \mathcal{F}, k)$ is a Yes instance of Set Cover if and only if (D, r, T, k) is a Yes instance of DST.

Suppose that $(\mathcal{U}, \mathcal{F}, k)$ is a Yes instance and let $F \subseteq \mathcal{F}$ be a solution. Consider the set $F_v = \{f_i | F_i \in F\}$. Clearly, $|F| \leq k$ and F is a solution for the instance (D, r, T, k) as all the terminals are reachable from r in $D[F \cup T \cup \{r\}]$.

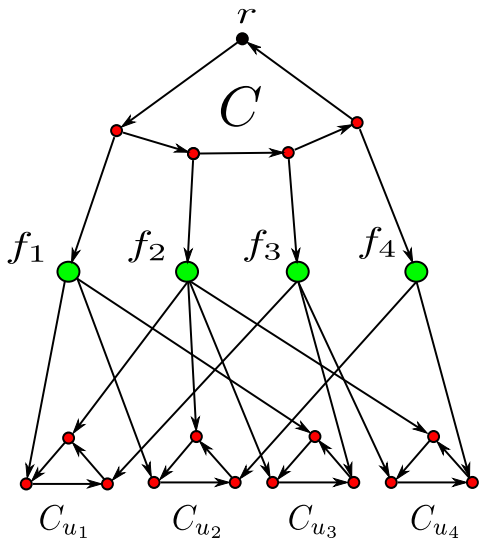

FIG. 3. *An instance of* SET COVER *reduced to an instance of* DST. *The red vertices are the terminals and the green vertices are the non-terminals.*

Conversely, suppose that F_v is a solution for (D, r, T, k). Since the only non-terminals are the vertices corresponding to the sets in $\mathcal{F}$, we define a set $F \subseteq \mathcal{F}$ as $F = \{F_i | f_i \in F_v\}$. Clearly $|F| \leq k$. We claim that F is a solution for the SET COVER instance $(\mathcal{U}, \mathcal{F}, k)$. Since there are no edges between the cycles C_u or C in the instance of DST, for every u, it must be the case that F_v contains some vertex f_i which has an arc to a vertex in the cycle C_u. But the corresponding set F_i will cover the element u and we have defined F such that $F_i \in F$. Hence, F is indeed a solution for the instance $(\mathcal{U}, \mathcal{F}, k)$. This completes the proof. □

In the instance of DST obtained in the above reduction, it seems that the presence of directed cycles in the subgraph induced by the terminals plays a major role in the *hardness* of this instance. We formally show that this is indeed the case by presenting an FPT algorithm for DST for the case the digraph induced by the terminals is acyclic.

THEOREM 3.23. DST *can be solved in time* $O^*(2^{O(dk)})$ *on* d*-degenerate graphs if the digraph induced by the terminals is acyclic.*

Proof. As the digraph induced by terminals is acyclic, Rule 1 does not apply and the instance is reduced. Therefore we can directly execute the algorithm DST-SOLVE on it. We set the degree bound to $d_b = d$. Note that if the set B_h and W_h created by the algorithm fulfill the invariants, then, as the digraph induced by $W_h \cup B_h$ is d-degenerate and the degree of every vertex in B_h is at least $d_b + 1 = d + 1$, there must be a vertex $v \in W_h$ with at most d (in-)neighbors in B_h. Therefore we have $d_w^{\max} = d$ and according to the analysis from Section 3.1, the algorithm runs in $O^*((2^{O(d_b)} \cdot d_w^{\max})^k) = O^*(2^{O(d)k})$ time. □

Theorem 3.23 combined with Lemma 3.10 results in the following corollary.

COROLLARY 3.24. *If* $\mathcal{C}$ *is an* $o(\log n)$*-degenerate class of digraphs, then* DST *parameterized by* k *is* FPT *on* $\mathcal{C}$ *if the digraph induced by terminals is acyclic.*

Before concluding this section, we also observe that analogous to the algorithms in Theorems 3.9 and 3.23, we can show that in the case when the digraph induced by terminals is acyclic, the DST problem admits an algorithm running in time $O^*(2^{O(hk)})$

on graphs excluding K_h as a topological minor.

THEOREM 3.25. DST *can be solved in time* $O^*(2^{O(hk)})$ *on graphs excluding* K_h *as a topological minor if the digraph induced by terminals is acyclic.*

Combined with Lemma 3.10, Theorem 3.25 has the following corollary.

COROLLARY 3.26. *If* $\mathcal{C}$ *is a class of digraphs excluding* $o(\log n)$*-sized topological minors, then* DST *parameterized by* k *is* FPT *on* $\mathcal{C}$ *if the digraph induced by terminals is acyclic.*

3.4. Hardness of DST**.** In this section, we show that the algorithm given in Theorem 3.23 is essentially the best possible with respect to the dependency on the degeneracy of the graph and the solution size. We begin by proving a lower bound on the time required by any algorithm for DST on graphs of degeneracy $O(\log n)$.

Our starting point is the known result for the following problem.

PARTITIONED SUBGRAPH ISOMORPHISM (PSI)
Input: Undirected graphs $H = (V_H, E_H)$ and $G = (V_G = \{g_1, \ldots, g_l\}, E_G)$ and a coloring function $col : V_H \to [l]$.
Question: Is there an injection $\phi : V_G \to V_H$ such that for every $i \in [l]$, $col(\phi(g_i)) = i$ and for every $(g_i, g_j) \in E_G$, $(\phi(g_i), \phi(g_j)) \in E_H$?

We need the following lemma by Marx [35].

LEMMA 3.27 ([35, Corollary 6.3]). PARTITIONED SUBGRAPH ISOMORPHISM *cannot be solved in time* $f(k)n^{o(\frac{k}{\log k})}$ *where* f *is an arbitrary function and* $k = |E_G|$ *is the number of edges in the smaller graph* G *unless* ETH *fails.*

Using the above lemma, we will first prove a similar kind of hardness for a restricted version of SET COVER (Lemma 3.28). Following that, we will reduce this problem to an instance of DST to prove the hardness of the problem on graphs of degeneracy $O(\log n)$.

LEMMA 3.28. *There is a constant* γ *such that* SET COVER *with size of each set bounded by* $\gamma \log m$ *cannot be solved in time* $f(k)m^{o(\frac{k}{\log k})}$*, unless* ETH *fails, where* k *is the size of the solution and* m *is the size of the family of sets.*

Proof. Let $(H = (V_H, E_H), G = (V_G, E_G), col)$ be an instance of PARTITIONED SUBGRAPH ISOMORPHISM where $|V_G| = l$ and the function $col : V_H \to [l]$ is a coloring (not necessarily proper) of the vertices of H with colors from $[l]$. We call the set of vertices of H which have the same color, a *color class*. We assume without loss of generality that there are no isolated vertices in G. Let n be the number of vertices of H. For each vertex of color i in H, we assign a $\log n$-sized subset of $2 \log n$. Since $\binom{2 \log n}{\log n} \geq n$, this is possible. Let this assignment be represented by the function $id : V_H \to 2^{[2 \log n]}$.

Recall that the vertices of G are numbered $g_1, \ldots, g_l$ and we are looking for a colorful subgraph of H isomorphic to G such that the vertex from color class i is mapped to the vertex g_i.

We will list the sets of the SET COVER instance and then we will define the set of elements contained in each set. For each pair (i, j) such that there is an edge between g_i and g_j, and for every edge between vertices u and v in V_H such that $col(u) = i$, $col(v) = j$, we have a set F^{ij}_{uv}. For each $i \in [l]$, for each $v \in V_H$ such that $col(v) = i$, we have a set F^{ii}_{vv}. The notation is chosen is such way that we can think of the sets as placed on a $l \times l$ grid, where the sets F^{ij}_{uv} for a fixed i and j are placed at the position (i, j) (see Fig. 4). Observe that many sets can be placed at a position and it may also be the case that some positions of the grid do not have a set placed on them. Let $\mathcal{F}$

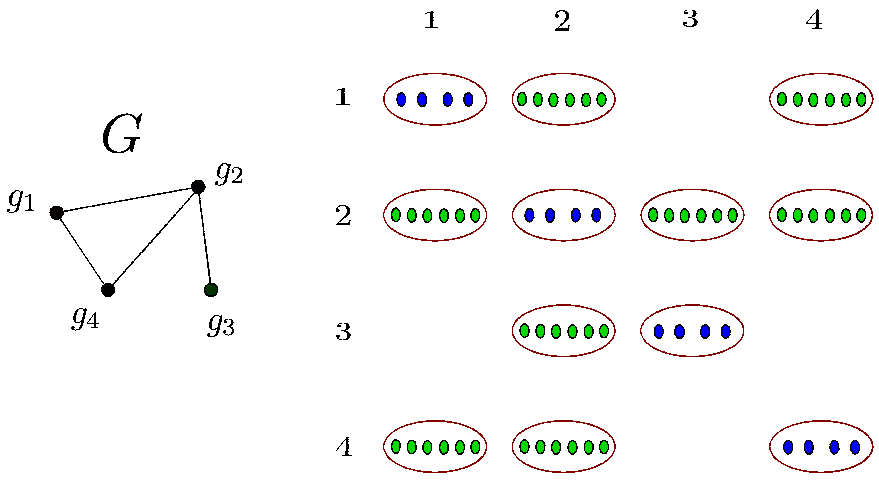

FIG. 4. *An illustration of the sets in the reduced instance, corresponding to the graph G (on the left). The blue sets at position (i,i) correspond to vertices of H with color i and the green sets at position (i,j) correspond to edges of H between color classes i and j.*

be the family of sets defined as above.

A position (i,j) which has a set placed on it is called non-empty and empty otherwise. Without loss of generality, we assume that if there are $i \neq j$ such that there is an edge between g_i and g_j in G, then the position (i,j) is non-empty. Two non-empty positions (i,j) and (i',j) are said to be *consecutive* if there is no non-empty position (i'',j) where $i < i'' < i'$. Similarly, two non-empty positions (i,j) and (i,j') are said to be consecutive if there is no non-empty position (i,j'') where $j < j'' < j'$. Note that consecutive positions are only defined along the same row or column.

We now define the universe $\mathcal{U}$ as follows. For every non-empty position (i,j), we have an element $s^{(i,j)}$. For every (i_1,j_1) and (i_2,j_2) such that they are consecutive, we have a set $\mathcal{U}^{(i_1,j_1)(i_2,j_2)}$ of $2\log n$ elements $\{u_1^{(i_1,j_1)(i_2,j_2)} \ldots, u_{2\log n}^{(i_1,j_1)(i_2,j_2)}\}$. An element $u_a^{(i_1,j_1)(i_2,j_2)}$ is said to *correspond* to $id(u)$ for some vertex u if $a \in id(u)$.

We will now define the elements contained within each set. For each non-empty position (i,j), add the element $s^{(i,j)}$ to every F_{uv}^{ij} for all (possible) u, v. Now, fix $1 \leq i \leq l$. Let (i,j_1) and (i,j_2) be consecutive positions where $j_1 < j_2$. For each set $F_{uv}^{ij_1}$, we add the elements $\{u_a^{(i,j_1)(i,j_2)} | a \notin id(u)\}$ and for each set $F_{uv}^{ij_2}$, we add the the elements $\{u_a^{(i,j_1)(i,j_2)} | a \in id(u)\}$.

Similarly, fix $1 \leq j \leq l$. Let (i_1,j) and (i_2,j) be consecutive positions where that $i_1 < i_2$. For each set $F_{uv}^{i_1 j}$, we add the elements $\{u_a^{(i_1,j)(i_2,j)} | a \notin id(u)\}$ and for each set $F_{uv}^{i_2 j}$, we add the the elements $\{u_a^{(i_1,j)(i_2,j)} | a \in id(u)\}$.

This completes the construction of the SET COVER instance. We first prove the following lemma regarding the constructed instance, which we will then use to show the correctness of the reduction.

CLAIM 1. *Suppose (i,j_1) and (i,j_2) are two consecutive positions where $j_1 < j_2$ and (i_1,j) and (i_2,j) are two consecutive positions where $i_1 < i_2$.*

1. *The elements in $\mathcal{U}^{(i,j_1)(i,j_2)}$ can be covered by precisely one set from (i,j_1) and one set from (i,j_2) if and only if the two sets are of the form $F_{uv}^{ij_1}$ and $F_{uv'}^{ij_2}$.*
2. *The elements in $\mathcal{U}^{(i_1,j)(i_2,j)}$ can be covered by precisely one set from (i_1,j)*

and one set from (i_2, j) *if and only if the two sets are of the form* $F^{i_1 j}_{uv}$ *and* $F^{i_2 j}_{u'v}$.

Proof. We prove the first statement. The proof of the second is analogous. Observe that, by the construction, the only sets which can cover elements in $\mathcal{U}^{(i,j_1)(i,j_2)}$ are sets from (i, j_1) and (i, j_2).

Suppose that the elements in $\mathcal{U}^{(i,j_1)(i,j_2)}$ are covered by precisely one set from (i, j_1) and one from (i, j_2) and the two sets are of the form $F^{ij_1}_{uv}$ and $F^{ij_2}_{u'v'}$ where $u \neq u'$. By the construction, the set $F^{ij_1}_{uv}$ covers the elements of $\mathcal{U}^{(i,j_1)(i,j_2)}$ which *do not* correspond to $id(u)$ and the set $F^{ij_2}_{u'v'}$ covers the elements of $\mathcal{U}^{(i,j_1)(i,j_2)}$ which correspond to $id(u')$. Since $col(u) = col(u')$ (implied by the construction), $id(u) \neq id(u')$. Since $|id(u)| = |id(u')|$, it must be the case that there is an element of $[2 \log n]$, say x, which is in $id(u)$ but not in $id(u')$. But then, it must be the case that the element $u^{(i,j_1)(i,j_2)}_x$ is left uncovered by both $F^{ij_1}_{uv}$ and $F^{ij_2}_{u'v'}$, a contradiction.

Conversely, consider two sets of the form $F^{ij_1}_{uv}$ and $F^{ij_2}_{uv'}$. We claim that these two sets together cover the elements in $\mathcal{U}^{(i,j_1)(i,j_2)}$. But this is true since $F^{ij_1}_{uv}$ covers the elements of $\mathcal{U}^{(i,j_1)(i,j_2)}$ which do not correspond to $id(u)$ and $F^{ij_2}_{uv'}$ covers the elements of $\mathcal{U}^{(i,j_1)(i,j_2)}$ which do correspond to $id(u)$. This completes the proof. □

Now that we have proved Claim 1, the proof of Lemma 3.28 continues. We claim that the instance (H, G, col) is a YES instance of PSI if and only if the instance $(\mathcal{U}, \mathcal{F}, k')$ is a YES instance of SET COVER, where $k' = 2|E_G| + |V_G|$. Suppose that (H, G, col) is a YES instance, ϕ is its solution, and let $v_i = \phi(g_i)$. We claim that the sets $F^{ij}_{v_i v_j}$, where (i, j) is a non-empty position, form a solution for the SET COVER instance. Since we have picked a set from every non-empty position (i, j), the elements $s^{(i,j)}$ are all covered. But since the sets we picked from any two consecutive positions match premise of Claim 1, the elements corresponding to the consecutive positions are also covered.

Conversely, suppose that the SET COVER instance is a YES instance and let $\mathcal{F}'$ be a solution. Since we must pick at least one set from each non-empty position (we have to cover the vertices $s^{(i,j)}$), and the number of non-empty positions equals k', we must have picked exactly one set from each non-empty position. Let v_i be the vertex corresponding to the set picked at position (i, i). We define the function ϕ as $\phi(g_i) = v_i$. Clearly, ϕ is an injection with $col(\phi(g_i)) = i$. It remains to show that for every g_i, g_j, if $(g_i, g_j) \in E_G$, then there is an edge between v_i and v_j. To show this, we need to show that the set picked from position (i, j) has to be exactly $F^{ij}_{v_i v_j}$. By Claim 1, the sets picked from row i are of the form $F^{ij}_{v_i,v}$, for any j and v and the sets picked from column j are of the form F^{ij}_{v,v_j}, for any i and v. Hence, the set picked from position (i, j) can only be $F^{ij}_{v_i v_j}$. Thus, there is an edge between v_i and v_j in H and ϕ is indeed a homomorphism. This completes the proof of equivalence of the two instances.

Since G contains no isolated vertex, we have $l = O(k)$ and, thus, $k' = \Theta(k)$. Observe that the number of sets m in the SET COVER instance is $|V_H| + 2|E_H|$, that is, $n \leq m$ and $m = O(n^2)$. Observe that each set contains at most $4 \log n + 1$ elements, one of the form $s^{(i,j)}$ and $\log n$ for each of the at most four consecutive positions the set can be a part of. Since the number of sets m is at least n, there is a constant γ such that the number of elements in each set is bounded by $\gamma \log m$. Finally, since $m = O(n^2)$, an algorithm for SET COVER of the form $f(k)m^{o(\frac{k}{\log k})}$ implies an algorithm of the form $f(k)n^{o(\frac{k}{\log k})}$ for PSI. This completes the proof of the lemma. □

Now we are ready to prove the main theorem of this section.

THEOREM 3.29. DST *cannot be solved in time* $f(k)n^{o(\frac{k}{\log k})}$ *on* $c \log n$*-degenerate graphs for any constant* $c > 0$ *even if the digraph induced by terminals is acyclic, where* k *is the solution size and* f *is an arbitrary function, unless* ETH *fails.*

Proof. The proof is by a reduction from the restricted version of SET COVER shown to be hard in Lemma 3.28. Fix a constant $c > 0$ and let $(\mathcal{U} = \{u_1, \ldots, u_n\}, \mathcal{F} = \{F_1, \ldots, F_m\}, k)$ be an instance of SET COVER, where the size of any set is at most $\gamma \log m$, for some constant γ. For each set F_i, we have a vertex f_i. For each element u_i, we have a vertex x_i. If an element u_i is contained in set F_j, then we add an arc (f_j, x_i). Further, we add another vertex r and add arcs (r, f_i) for every i. Finally, we add $m^{2\gamma/c}$ isolated vertices. This completes the construction of the digraph D. We set $T = \{x_1, \ldots, x_n\}$ as the set of terminals and r as the root.

We claim that $(\mathcal{U}, \mathcal{F}, k)$ is a YES instance of SET COVER if and only if (D, r, T, k) is a YES instance of DST. Suppose that $\{F_1, \ldots, F_k\}$ is a set cover for the given instance. It is easy to see that the vertices $\{f_1, \ldots, f_k\}$ form a solution for the DST instance.

Conversely, suppose that $\{f_1, \ldots, f_k\}$ is a solution for the DST instance. Since the only way that r can reach a vertex x_i is through some f_j, and the construction implies that $u_i \in F_j$, the sets $\{F_1, \ldots, F_k\}$ form a set cover for $(\mathcal{U}, \mathcal{F}, k)$. This concludes the proof of equivalence of the two instances.

We claim that the degeneracy of the graph D is $c \log n_1 + 1$, where n_1 is the number of vertices the graph has. First, we show that the degeneracy of the graph D is bounded by $\gamma \log m + 1$. This follows from that each vertex f_i has total degree at most $\gamma \log m + 1$ and if a subgraph contains none of these vertices, then it contains no edges. Now, n_1 is at least $m^{2\gamma/c}$. Hence, $\log n_1 \geq (2\gamma/c) \log m$ and the degeneracy of the graph is at most $\gamma \log m + 1 \leq c \cdot (2\gamma/c) \log m \leq c \log n_1$. Finally, since each vertex f_i is adjacent to at most $\gamma \log m + 1$ vertices, $n_1 = O(m \log m + m^{2\gamma/c})$ and, thus, it is polynomial in m. Hence, an algorithm for DST of the form $f(k)n_1^{o(\frac{k}{\log k})}$ implies an algorithm of the form $f(k)m^{o(\frac{k}{\log k})}$ for the SET COVER instance. This concludes the proof of Theorem 3.29. □

Combining the Theorem 3.29 with Lemma 3.10 we get the following corollary.

COROLLARY 3.30. *There are no two functions* f *and* g *such that* $g(d) = o(d)$ *and there is an algorithm for* DST *running in time* $O^*(2^{g(d)f(k)})$*, unless* ETH *fails.*

To examine the dependency on the solution size we utilize the following theorem.

THEOREM 3.31. ([29]) *There is a constant* c *such that* DOMINATING SET *does not have an algorithm running in time* $O^*(2^{o(n)})$ *on graphs of maximum degree* $\leq c$ *unless* ETH *fails.*

From Theorem 3.31, we can infer the following corollary.

COROLLARY 3.32. *There are no two functions* f *and* g *such that* $f(k) = o(k)$ *and there is an algorithm for* DST *running in time* $O^*(2^{g(d)f(k)})$*, unless* ETH *fails.*

Proof. We use the following standard reduction from DOMINATING SET to DST. Let $(G = (V, E), k)$ be an instance of DOMINATING SET with the maximum degree of G bounded by some constant c. We can assume that the number of vertices n of the graph G is at most $ck + k$, since otherwise, it is a trivial NO instance. Let $D = (V', A)$ be the digraph defined as follows. We set $V' = V \times \{1, 2\} \cup \{r\}$. There is an arc in A from $(u, 1)$ to $(v, 2)$ if either $u = v$ or there is an edge between u and v in E. Finally, there is an arc from r to $(v, 1)$ for every $v \in V$. We let $T = \{(v, 2) \mid v \in V\}$.

It is easy to check that $S \subseteq V$ is a solution to the instance (G, k) of DOMINATING SET if and only if $S \times \{1\}$ is a solution to the instance (D, r, T, k) of DST. As the

vertices $(v, 2)$ have degree at most $c+1$, D is $c+1$-degenerate. Since the reduction is polynomial time, preserves k and $k = \Theta(n)$, an algorithm for DST running in time $O^*(2^{g(d)f(k)})$ for some $f(k) = o(k)$ would solve DOMINATING SET on graphs of maximum degree $\leq c$ in time $O^*(2^{g(c+1)f(k)}) = O^*(2^{o(n)})$, and, hence, ETH fails by Theorem 3.31. □

4. Applications to DOMINATING SET. In this section, we adapt the ideas used in the algorithms for DST to design improved algorithms for the DOMINATING SET problem and some variants of it in subclasses of degenerate graphs.

4.1. Introduction for DOMINATING SET. On general graphs DOMINATING SET is W[2]-complete [13]. However, there are many interesting graph classes where FPT-algorithms exist for DOMINATING SET. The project of expanding the horizon where FPT algorithms exist for DOMINATING SET has produced a number of cutting-edge techniques of parameterized algorithm design. This has made DOMINATING SET a cornerstone problem in parameterized complexity. For an example the initial study of parameterized subexponential algorithms for DOMINATING SET, on planar graphs [1, 19] resulted in the development of bidimensionality theory characterizing a broad range of graph problems that admit efficient approximation schemes, subexponential time FPT algorithms and efficient polynomial time pre-processing (called *kernelization*) on minor closed graph classes [10, 11]. Alon and Gutner [2] and Philip, Raman, and Sikdar [40] showed that DOMINATING SET problem is FPT on graphs of bounded degeneracy and on $K_{i,j}$-free graphs, respectively.

Numerous papers also concerned the approximability of DOMINATING SET. It follows from [15] that DOMINATING SET on general graphs can approximated to within roughly $\ln(\Delta(G) + 1)$, where $\Delta(G)$ is the maximum degree in the graph G. On the other hand, it is NP-hard to approximate DOMINATING SET in bipartite graphs of degree at most B within a factor of $(\ln B - c \ln \ln B)$, for some absolute constant c [7]. Note that a graph of degree at most B excludes K_{B+2} as a topological minor, and, hence, the hardness also applies to graphs excluding K_h as a topological minor. While a polynomial time approximation scheme (PTAS) is known for K_h-minor-free graphs [23], we are not aware of any constant factor approximation for DOMINATING SET on graphs excluding K_h as a topological minor or d-degenerate graphs.

Based on the ideas from previous sections, we develop an algorithm for DOMINATING SET. Our algorithm for DOMINATING SET on d-degenerate graphs improves over the $O^*(k^{O(dk)})$ time algorithm by Alon and Gutner [2]. In fact, it turns out that our algorithm is essentially optimal – we show that, assuming the ETH, the running time dependence of our algorithm on the degeneracy of the input graph and solution size k cannot be significantly improved. Furthermore, we also give a factor $O(d^2)$ approximation algorithm for DOMINATING SET on d-degenerate graphs. A list of our results for DOMINATING SET is given below.

1. There is an $O^*(3^{hk+o(hk)})$-time algorithm for DOMINATING SET on graphs excluding K_h as a topological minor.
2. There is an $O^*(3^{dk+o(dk)})$-time algorithm for DOMINATING SET on d-degenerate graphs. This implies that DOMINATING SET is FPT on $o(\log n)$-degenerate classes of graphs.
3. For any constant $c > 0$, there is no $f(k)n^{o(\frac{k}{\log k})}$-time algorithm on graphs of degeneracy $c \log n$ unless ETH fails.
4. There are no two functions f and g such that $g(d) = o(d)$ and there is an

algorithm for Dominating Set running in time $O^*(2^{g(d)f(k)})$, unless ETH fails.

5. There are no two functions f and g such that $f(k) = o(k)$ and there is an algorithm for Dominating Set running in time $O^*(2^{g(d)f(k)})$, unless ETH fails.
6. There is an $O(dn \log n)$ time factor $O(d^2)$ approximation algorithm for Dominating Set on d-degenerate graphs.

4.2. Dominating Set **on graphs of bounded degeneracy.** We begin by giving an algorithm for Dominating Set running in time $O^*(3^{hk+o(hk)})$ in graphs excluding K_h as a topological minor. This improves over the $O^*(2^{O(kh \log h)})$ algorithm of [2]. Though the algorithm we give here is mainly built on the ideas developed for the algorithm for DST, the algorithm has to be modified slightly in certain places in order to fit this problem. We also stress this as an example of how these ideas, with some modifications, can be made to fit problems other than DST.

We begin by appropriately defining the sets appearing during the run of the recursive algorithm based on partial solution Y and a degree bound d_b (see Fig. 5):

- $B = B(Y)$ is the set of vertices (other than Y) dominated by Y.
- $W = W(Y)$ is the set of vertices not dominated by Y.
- $B_h = B_h(Y, d_b)$ is the set of vertices of B which dominate at least $d_b + 1$ vertices of W.
- $B_l = B_l(Y, d_b)$ is the set of vertices of B which dominate at most d_b vertices of W.
- $W_h = W_h(Y, d_b)$ is the set of vertices of W which have a neighbor in $B_h \cup W$.
- $W_l = W_l(Y, d_b)$ are the remaining vertices of W.

Note that the sets B_h, B_l, W_h, W_l and Y are pairwise disjoint. We will also maintain the invariant that the set Y is of size at most k.

We continue by proving lemmata required for the correctness of the base cases of our algorithm.

Lemma 4.1. *Let (G, k) be an instance of* Dominating Set. *Let $Y \subseteq V$ be a set of vertices, d_b a positive constant, and let B_h, B_l, W_h, W_l be as defined above. If $|W_l| > d_b(k - |Y|)$, then the given instance does not admit a solution which contains Y.*

Proof. Since any vertex in W_l does not have neighbors in $B_h \cup W \cup Y$, $G[W_l]$ is an independent set, and the only vertices which can dominate a vertex in W_l are either itself, or a vertex of B_l. Any vertex of W_l can only dominate itself (vertices of B_l are already dominated by Y) and any vertex of B_l can dominate at most d_b vertices of W. Hence, if $|W_l| > d_b(k - |Y|)$, W_l cannot be dominated by adding $k - |Y|$ vertices to Y. This completes the proof. □

Lemma 4.2. *Let (D, k) be an instance of* Dominating Set. *Let $Y \subseteq V$ be a set of vertices, d_b a positive constant, and let B_h, B_l, W_h, W_l be as defined above. If $B_h \cup W_h$ is empty, then there is an algorithm which can test whether this instance has a solution containing Y in time $O^*(2^{d_b(k-|Y|)})$.*

Proof. The premises of the lemma imply that the only potentially non-empty sets are Y, B_l and $W = W_l$. If W is empty, we are done. Suppose W is non-empty. We can assume that $|Y| \leq k$ and, by Lemma 4.1, that $|W| \leq d_b(k - |Y|)$. Since W is an independent set, the only vertices the vertices of W can dominate (except for vertices dominated by Y), are themselves. Thus it is never worse to take a neighbor of them in the solution if they have one. The isolated vertices from W can be simply moved to Y, as we have to take them into the solution. Now, it remains to find a set of vertices of B_l of the appropriate size such that they dominate the remaining

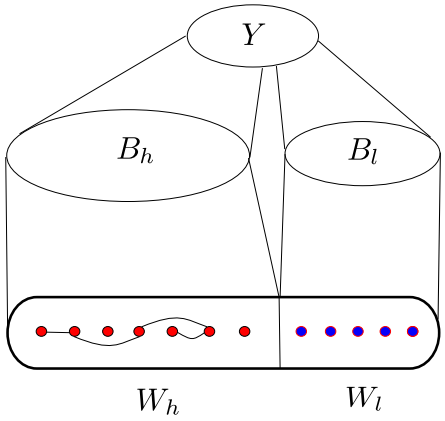

FIG. 5. *An illustration of the sets defined in Theorem 4.3*

vertices of W. But in this case, we can reduce it to an instance of DST and apply the algorithm from Lemma 3.5. The reduction is as follows. Consider the graph $G[B_l \cup W]$. Remove all edges between vertices in B_l. Let this graph be G_{dst}. Simply add a new vertex r and add directed edges from r to every vertex in B_l. Also orient all edges between B_l and W, from B_l to W. Now, W is set as the terminal set. Now, since $|W| \leq d_b(k-|Y|)$, it is easy to see that the algorithm NEDERLOF(G_{dst}, r, W, W) runs in time $O^*(2^{d_b(k-|Y|)})$. This completes the proof of the lemma. □

THEOREM 4.3. DOMINATING SET *can be solved in time* $O^*(3^{hk+o(hk)})$ *on graphs excluding* K_h *as a topological minor.*

Proof. The algorithm we describe takes as input an instance (G, k), a degree bound d_b, and a vertex set Y and returns a smallest solution for the instance, which contains Y if such a solution exists. If there is no such solution, then the algorithm returns a dummy symbol S_∞. To simplify the description, we assume that $|S_\infty| = \infty$. The algorithm is a recursive algorithm and at any stage of recursion, the corresponding recursive step returns the smallest set found in the recursions initiated in this step. For the purpose of this proof, we will always have $d_b = h - 2$. Initially, the set Y is empty.

Observe that, by Lemma 4.1, if $|W_l| > d_b(k - |Y|)$, then there is no solution containing Y and hence we return S_∞ (see Algorithm 4.1). If $B_h \cup W_h$ is empty, then we apply Lemma 4.2 to solve the problem in time $O^*(2^{d_b(k-|Y|)})$. If $B_h \cup W_h$ is non-empty, then we find a vertex $v \in W_h$ with the least number of neighbors in $B_h \cup W_h$. Let N be the set of these neighbors and let $|N| = d_w$.

We then branch into $d_w + 2$ branches described as follows. In the first $d_w + 1$ branches, we move a vertex u of $N \cup \{v\}$, to the set Y, and recourse on the resulting instance. In the last of the $d_w + 2$ branches, that is, the branch where we have guessed that none of the vertices of N are in the dominating set, delete the vertices of $N \cap B_h$ and the edges from v to $N \cap W_h$, to obtain the graph G' and recourse on the resulting instance. Note that computing the sets for this instance, we will have $v \in W_l$.

Correctness. At each node of the recursion tree, we define a measure $\mu(I) = d_b(k - |Y|) - |W_l|$. We prove the correctness of the algorithm by induction on this measure. In the base case, when $d_b(k - |Y|) - |W_l| < 0$, the algorithm is correct (by Lemma 4.1). Now, we assume as induction hypothesis that the algorithm is correct on instances with measure less than some $\mu \geq 0$. Consider an instance I such that

Input : An instance (G, k) of Dominating Set, degree bound d_b, set $Y \subset V(G)$
Output: A smallest solution of size at most k and containing Y for the instance (G, k) if it exists and S_∞ otherwise

Compute the sets B_h, B_l, W_h, W_l.
if $|W_l| > d_b(k - |Y|)$ **then return** S_∞.
else if $B_h = \emptyset$ **then**
  $S \leftarrow \text{Nederlof}(G_{dst}, r, W, W) \cup Y$.
  if $|S| > k$ **then** $S \leftarrow S_\infty$.
  return S.
end
else
  $S \leftarrow S_\infty$.
  Find vertex $v \in W_h$ with the least number of neighbors in $B_h \cup W_h$.
  for $u \in ((B_h \cup W_h) \cap N[v])$ **do**
    $Y = Y \cup \{u\}$.
    $S' \leftarrow$ DS-solve$((G, k), d_b, Y)$.
    if $|S'| < |S|$ **then** $S \leftarrow S'$.
  end
  $G' \leftarrow G \setminus (N[v] \cap B_h) - \{vu \mid u \in N(v) \cap W_h\}$.
  $S' \leftarrow$ DS-solve$((G', k), d_b, Y)$.
  if $|S'| < |S|$ **then** $S \leftarrow S'$.
  return S.
end

Algorithm 4.1: Algorithm DS-solve for Dominating Set

$\mu(I) = \mu$. Since the branching is exhaustive it is sufficient to show that the algorithm is correct on each of the child instances. To show this, it is sufficient to show that for each child instance I', $\mu(I') < \mu(I)$. In the first $d_w + 1$ branches, the size of the set Y increases by 1, and the size of the set W_l does not decrease (W_l has no neighbors in $B_h \cup W$). Hence, in each of these branches, $\mu(I') \leq \mu(I) - d_b$. In the final branch, though the size of the set Y remains the same, the size of the set W_l increases by at least 1. Hence, in this branch, $\mu(I') \leq \mu(I) - 1$. Thus, we have shown that in each branch, the measure drops, hence completing the proof of correctness of the algorithm.

Analysis. Since D excludes K_h as a topological minor, Lemma 3.6, combined with the fact that we set $d_b = h - 2$, implies that $d_w^{\max} = ch^4$, for some c, is an upper bound on the maximum d_w which can appear during the execution of the algorithm. We first bound the number of leaves of the recursion tree as follows. The number of leaves is bounded by $\sum_{i=0}^{d_b k} \binom{d_b k}{i} (d_w^{\max} + 1)^{k - \frac{i}{d_b}}$. To see this, observe that each branch of the recursion tree can be described by a length-$d_b k$ vector as shown in the correctness paragraph. We then select i positions of this vector on which the last branch was taken. Finally for $k - \frac{i}{d_b}$ of the remaining positions, we describe which of the first at most $(d_w^{\max} + 1)$ branches was taken. Any of the first $d_w^{max} + 1$ branches can be taken at most $k - \frac{i}{d_b}$ times if the last branch is taken i times.

The time taken along each root to leaf path in the recursion tree is polynomial, while the time taken at a leaf for which the last branch was taken i times is

$O^*(2^{d_b(k-(k-\frac{i}{d_b}))}) = O^*(2^{i+k})$ (see Lemmata 4.1 and 4.2). Hence, the running time of the algorithm is

$$O^*\left(\sum_{i=0}^{d_b k}\binom{d_b k}{i}(d_w^{\max}+1)^{k-\frac{i}{d_b}}\cdot 2^{i+k}\right) = O^*\left((2d_w^{\max}+2)^k\cdot\sum_{i=0}^{d_b k}\binom{d_b k}{i}\cdot 2^i\right)$$
$$= O^*\left((2d_w^{\max}+2)^k\cdot 3^{d_b k}\right).$$

For $d_b = h-2$ and $d_w^{\max} = ch^4$ this is $O^*(3^{hk+o(hk)})$. This completes the proof of the theorem. □

Observe that, when Algorithm 4.1 is run on a graph of degeneracy d, setting $d_b = d$, combined with the simple fact that $d_w^{max} \leq d$ gives us the following theorem.

THEOREM 4.4. DOMINATING SET *can be solved in time* $O^*(3^{dk+o(dk)})$ *on graphs of degeneracy* d.

From the above two theorems and Lemma 3.10, we have the following corollary.

COROLLARY 4.5. *If* $\mathcal{C}$ *is a class of graphs excluding* $o(\log n)$*-sized topological minors or an* $o(\log n)$*-degenerate class of graphs, then* DOMINATING SET *parameterized by* k *is* FPT *on* $\mathcal{C}$.

4.3. Hardness. In this section we give complementary lower bounds on the running time of algorithms solving DOMINATING SET.

THEOREM 4.6. DOMINATING SET *cannot be solved in time* $f(k)n^{o(\frac{k}{\log k})}$ *on* $c\log n$*-degenerate graphs for any constant* $c > 0$*, where* k *is the solution size and* f *is an arbitrary function, unless* ETH *fails.*

Proof. The proof is by a reduction from the restricted version of SET COVER shown to be hard in Lemma 3.28. Fix a constant $c > 0$ and let $(\mathcal{U} = \{u_1, \ldots, u_n\}, \mathcal{F} = \{F_1, \ldots, F_m\}, k)$ be an instance of SET COVER, where the size of any set is at most $\gamma \log m$ for some constant γ. For each set F_i, we have a vertex f_i. For each element u_i, we have a vertex x_i. If an element u_i is contained in the set F_j, then we add an edge (f_j, x_i). Further, we add two vertices r and p and add edges (r, f_i) for every i and an edge (r, p). Finally, we add a star of size $m^{2\gamma/c}$ centered in q disjoint from the rest of the graph. This completes the construction of the graph G.

We claim that $(\mathcal{U}, \mathcal{F}, k)$ is a YES instance of SET COVER if and only if $(G, k+2)$ is a YES instance of DOMINATING SET. Suppose that $\{F_1, \ldots, F_k\}$ is a set cover for the given instance. It is easy to see that the vertices $q, r, f_1, \ldots, f_k$ form a solution for the DOMINATING SET instance.

In the converse direction, since one of p and r and at least one vertex of the star must be in any dominating set, we assume without loss of generality that r and q are contained in the minimum dominating set. Also, since r dominates any vertex f_i, we may also assume that the solution is disjoint from the x_i's. This is because, if x_i was in the solution, we can replace it with an adjacent f_j to get another solution of the same size. Hence, we suppose that $\{q, r, f_1, \ldots, f_k\}$ is a solution for the DOMINATING SET instance. Since the only way that some vertex x_i can be dominated is by some f_j, and the construction implies that $u_i \in F_j$, the sets $\{F_1, \ldots, F_k\}$ form a set cover for $(\mathcal{U}, \mathcal{F}, k)$. This concludes the proof of equivalence of the two instances.

We claim that the degeneracy of the graph G is bounded by $c\log n_1$, where n_1 is the number of vertices in the graph G. First, we claim that the degeneracy of the graph G is bounded by $\gamma \log m + 1$. This follows from that each vertex f_i has total degree at most $\gamma \log m + 1$, each leaf of the star has degree 1 and if a subgraph contains none of these vertices, then it contains no edges. Now, n_1 is at least $m^{2\gamma/c}$.

Hence, $\log n_1 \geq (2\gamma/c)\log m$ and the degeneracy of the graph is at most $\gamma \log m + 1 \leq c \cdot (2\gamma/c)\log m \leq c \log n_1$. Finally, since each vertex f_i is incident to at most $\gamma \log m + 1$ vertices, $n_1 = O(m \log m + m^{2\gamma/c})$ and, thus, it is polynomial in m. Hence, an algorithm for Dominating Set of the form $f(k)n_1^{o(\frac{k}{\log k})}$ implies an algorithm of the form $f(k)m^{o(\frac{k}{\log k})}$ for the Set Cover instance. This concludes the proof of the theorem. □

As a corollary of Theorem 4.6, and Lemma 3.10, we have the following corollary.

Corollary 4.7. *There are no two functions f and g such that $g(d) = o(d)$ and there is an algorithm for* Dominating Set *running in time $O^*(2^{g(d)f(k)})$, unless ETH fails.*

From Theorem 3.31, we can infer the following corollary.

Corollary 4.8. *There are no two functions f and g such that $f(k) = o(k)$ and there is an algorithm for* Dominating Set *running in time $O^*(2^{g(d)f(k)})$, unless ETH fails.*

Proof. Suppose that there was an algorithm for Dominating Set running in time $O^*(2^{g(d)f(k)})$, where $f(k) = o(k)$. Consider an instance (G, k) of Dominating Set where G is a graph with maximum degree c. We can assume that the number of vertices of the graph is at most $ck + k$, since otherwise, it is a trivial No instance. Hence, $k = \Theta(n)$ and thus, an algorithm running in time $O^*(2^{g(d)f(k)})$ will run in time $O^*(2^{g(c)f(k)}) = O^*(2^{o(n)})$, and, by Theorem 3.31, ETH fails. □

Thus, Corollaries 4.7 and 4.8 together show that, unless ETH fails, our algorithm for Dominating Set has the best possible dependence on both the degeneracy and the solution size.

4.4. Approximating Dominating Set on graphs of bounded degeneracy. In this section, we adapt the ideas developed in the previous subsections, to design a polynomial-time d^2-approximation algorithm for the Dominating Set problem on d-degenerate graphs.

Theorem 4.9. *There is a $O(dn \log n)$-time d^2- approximation algorithm for the* Dominating Set *problem on d-degenerate graphs.*

Proof. The approximation algorithm is based on our FPT algorithm, but whenever the branching algorithm would branch, we take all candidates into the solution and cycle instead of recoursing. During the execution of the algorithm the partial solution is kept in the set Y and vertex sets B, W, B_h, B_l, W_h, and W_l are updated with the same meaning as in the branching algorithm. In the base case, all vertices of W_l are taken into the solution. This last step could be replaced by searching a dominating set for the vertices in W_l by some approximation algorithm for Set Cover. While this would probably improve the performance of the algorithm in practice, it does not improve the theoretical worst case bound.

Correctness. It is easy to see that the set Y output by the algorithm is a dominating set for G. Now let Q be an optimal dominating set for G. We want to show that $|Y|$ is at most d^2 times $|Q|$. In particular, we want to account every vertex of Y to some vertex of Q, which "should have been chosen instead to get the optimal set." Before we do that let us first observe how the vertices can move around the sets during the execution of the algorithm. Once a vertex is added to Y it is never removed. Hence, once a vertex is moved from W to B, it is never moved back. But then the vertices in B are only losing neighbors in W, and once they get to B_l they are never moved anywhere else. Thus, vertices in W_l can never get a new neighbor

Input : An undirected graph $G = (V, E)$ without isolated vertices
Output: A dominating set for G of size at most d^2 times the size of a minimum dominating set

1 $Y \leftarrow \emptyset$
2 $W_h \leftarrow V$
3 **while** $W_h \neq \emptyset$ **do**
4 *Find vertex $v \in W_h$ with the least number of neighbors in $B_h \cup W_h$.*
5 $Y \leftarrow Y \cup ((B_h \cup W_h) \cap N(v))$
6 $B \leftarrow$ vertices in $V \setminus Y$ with a neighbor in Y
7 $W \leftarrow V \setminus (Y \cup B)$
8 $B_h \leftarrow$ vertices in B with at least $d + 1$ neighbors in W
9 $B_l \leftarrow B \setminus B_h$
10 $W_h \leftarrow$ vertices in W with a neighbor in B_h or W
11 $W_l \leftarrow W \setminus W_h$
12 **end**
13 $Y \leftarrow Y \cup W_l$
14 **return** Y

Algorithm 4.2: Algorithm DS-APPROX for DOMINATING SET

in W or B_h and they also stay in W_l until Step 13. The vertices of B_h can get to Y or B_l and the vertices of W_h can get to any other set during the execution of the algorithm.

Now let v be the vertex found in Step 4 of Algorithm 4.2 and w be the vertex dominating it in Q. As in the branching algorithm, we know that $|(B_h \cup W_h) \cap N(v)| \leq d$ since the subgraph of G induced by $(B_h \cup W_h)$ is d-degenerate. Hence at most d vertices are added to Y in Step 5 of the algorithm. We charge the vertex w for these at most d vertices and make the vertex v responsible for that. Finally, in Step 13 let $v \in W_l$ and w be a vertex which dominates it in Q. We charge w for adding v to Y and make v responsible for it. Obviously, for each vertex added to Y, some vertex is responsible and some vertex of Q is charged. It remains to count for how many vertices can be a vertex of Q charged.

Observe that whenever a vertex is responsible for adding some vertices to Y, it is in W and after adding these vertices it becomes dominated and, hence, moved to B or Y. Therefore, each vertex becomes responsible for adding vertices only once and, thus, each vertex is responsible for adding at most d vertices to Y. Now let us distinguish in which set a vertex w of Q is, when it is first charged. If w is first charged in Step 5 of the algorithm, then a vertex $v \in W_h$ is responsible for that and w has to be either in W_h, B_h, or B_l, as vertices in W_h do not have neighbors elsewhere. In the first two cases, if $w \neq v$, then it is moved to Y and hence has no neighbors in W anymore. If $v = w$, then it is moved to B_l, and it has no neighbors in W, as all of them are moved to Y. Thus, in these cases, after the step is done, w has no neighbors in W and, hence, is never charged again. If w is in B_l, then it has at most d neighbors in W and, as each of them is responsible for adding at most d vertices to Y, w is charged for at most d^2 vertices. If a vertex w is first charged in Step 13, then it is B_l or W_l, has at most d neighbors in W_l, each of them being responsible for addition of exactly one vertex, so it is charged for addition of at most d vertices. It follows, that every vertex of Q is charged for addition of at most d^2 to Y and, therefore, $|Y|$ is at

most d^2 times $|Q|$.

Running time analysis. To see the running time, observe first that by the above argument, every vertex is added to each of the sets at most once. Also a vertex is moved from one set to another only if some of its neighbors is moved to some other set or it is selected in Step 4. Hence, we might think of a vertex sending a signal to all its neighbor, once it is moved to another set. There are only constantly many signals and each of them is sent at most once over each edge in each direction. Hence, all the updates to the sets can be done in $O(m) = O(dn)$-time, as the graph is d-degenerate.

Also each vertex in W_h can keep its number of neighbors in $W_h \cup B_h$ and update it whenever it receives a signal from some of its neighbors about being moved out of $W_h \cup B_h$. We can keep a heap of the vertices in W_h sorted by a degree in $W_h \cup B_h$ and update it in $O(\log n)$ time whenever the degree of some of the vertices change. This means $O(dn \log n)$ time to keep the heap through the algorithm. Using the heap, the vertex v in Step 4 can be found in $O(\log n)$-time in each iteration. As in each iteration at least one vertex is added to Y, there are at most n iterations and the total running time is $O(dn \log n)$. □

This algorithm can be also used for K_h-minor-free and K_h-topological-minor free graphs, yielding $O(h^2 \cdot \log h)$-approximation and $O(h^4)$-approximation, as these graphs are $O(h \cdot \sqrt{\log h})$ and $O(h^2)$-degenerate, respectively. As far as we know, this is also the first constant factor approximation for dominating set in K_h-topological-minor free graphs. Although PTAS is known for K_h-minor-free graphs [23], our algorithm can be still of interest due to its simplicity and competitive running time.

5. Conclusions. We gave the first FPT algorithms for the Steiner Tree problem on directed graphs excluding a fixed graph as a (topological) minor, and then extended the results to directed graphs of bounded degeneracy. We mention that the same approach also gives us FPT algorithms for Dominating Set and some of its variants, for instance Connected Dominating Set and Total Dominating Set. Finally, in the process of showing the optimality of our algorithm, we showed that for any constant c, DST is not expected to have an algorithm of the form $f(k)n^{o(\frac{k}{\log k})}$ on $c \log n$-degenerate graphs. It would be interesting to either improve this lower bound, or prove the tightness of this bound by giving an algorithm with a matching running time.

REFERENCES

[1] J. Alber, H. L. Bodlaender, H. Fernau, T. Kloks, and R. Niedermeier, *Fixed parameter algorithms for dominating set and related problems on planar graphs*, Algorithmica, 33 (2002), pp. 461–493. Cited on page 25.

[2] Noga Alon and Shai Gutner, *Linear time algorithms for finding a dominating set of fixed size in degenerated graphs*, Algorithmica, 54 (2009), pp. 544–556. Cited on pages 3, 25, and 26.

[3] Marshall W. Bern and Paul E. Plassmann, *The Steiner problem with edge lengths 1 and 2*, Inf. Process. Lett., 32 (1989), pp. 171–176. Cited on page 1.

[4] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto, *Fourier meets Möbius: fast subset convolution*, in Proc. STOC 2007, 2007, pp. 67–74. Cited on page 4.

[5] Béla Bollobás and Andrew Thomason, *Proof of a conjecture of Mader, Erdös and Hajnal on topological complete subgraphs*, Eur. J. Comb., 19 (1998), pp. 883–887. Cited on page 7.

[6] Moses Charikar, Chandra Chekuri, To-Yat Cheung, Zuo Dai, Ashish Goel, Sudipto Guha, and Ming Li, *Approximation algorithms for directed Steiner problems*, J. Algorithms, 33 (1999), pp. 73–91. Cited on page 2.

[7] M. Chlebík and J. Chlebíková, *Approximation hardness of dominating set problems in bounded degree graphs*, Inf. Comput., 206 (2008), pp. 1264–1275. Cited on page 25.

[8] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh, *Parameterized Algorithms*, Springer, 2015. Cited on page 2.

[9] Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk, *Solving connectivity problems parameterized by treewidth in single exponential time*, in Proc. FOCS 2011, 2011, pp. 150–159. Cited on page 4.

[10] Erik D. Demaine, Fedor V. Fomin, Mohammadtaghi Hajiaghayi, and Dimitrios M. Thilikos, *Subexponential parameterized algorithms on bounded-genus graphs and H-minor-free graphs*, J. ACM, 52 (2005), pp. 866–893. Cited on page 25.

[11] Erik D. Demaine and MohammadTaghi Hajiaghayi, *The bidimensionality theory and its algorithmic applications*, The Computer Journal, 51 (2008), pp. 292–302. Cited on page 25.

[12] Erik D. Demaine, Mohammad Taghi Hajiaghayi, and Philip N. Klein, *Node-weighted Steiner tree and group Steiner tree in planar graphs*, ACM Transactions on Algorithms, 10 (2014), p. 13. Cited on page 2.

[13] Rodney G. Downey and Michael R. Fellows, *Fundamentals of Parameterized Complexity*, Texts in Computer Science, Springer, 2013. Cited on pages 2 and 25.

[14] S. E. Dreyfus and R. A. Wagner, *The Steiner problem in graphs*, Networks, 1 (1971), pp. 195–207. Cited on page 4.

[15] Rong-chii Duh and Martin Fürer, *Approximation of k-set cover by semi-local optimization*, in STOC, Frank Thomson Leighton and Peter W. Shor, eds., ACM, 1997, pp. 256–264. Cited on page 25.

[16] David Eppstein, *Diameter and treewidth in minor-closed graph families*, Algorithmica, 27 (2000), pp. 275–291. Cited on page 4.

[17] Jörg Flum and Martin Grohe, *Parameterized Complexity Theory*, Springer-Verlag, Berlin, 2006. Cited on page 2.

[18] Fedor V. Fomin, Petteri Kaski, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh, *Parameterized single-exponential time polynomial space algorithm for steiner tree*, in Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I, vol. 9134 of Lecture Notes in Computer Science, Springer, 2015, pp. 494–505. Cited on page 4.

[19] Fedor V. Fomin and Dimitrios M. Thilikos, *Dominating sets in planar graphs: Branch-width and exponential speed-up*, SIAM J. Comput., 36 (2006), pp. 281–309. Cited on page 25.

[20] Bernhard Fuchs, Walter Kern, Daniel Mölle, Stefan Richter, Peter Rossmanith, and Xinhui Wang, *Dynamic programming for minimum Steiner trees*, Theory Comput. Syst., 41 (2007), pp. 493–500. Cited on page 4.

[21] M. R. Garey and D. S. Johnson, *The rectilinear Steiner tree problem is NP-complete*, SIAM J. Appl. Math., 32 (1977), pp. 826–834. Cited on page 1.

[22] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979. Cited on page 1.

[23] Martin Grohe, *Local tree-width, excluded minors, and approximation algorithms*, Combinatorica, 23 (2003), pp. 613–632. Cited on pages 25 and 32.

[24] Martin Grohe and Dániel Marx, *Structure theorem and isomorphism test for graphs with excluded topological subgraphs*, SIAM J. Comput., 44 (2015), pp. 114–159. Cited on pages 5, 10, and 11.

[25] Sudipto Guha and Samir Khuller, *Approximation algorithms for connected dominating sets*, Algorithmica, 20 (1998), pp. 374–387. Cited on page 2.

[26] J. Guo, R. Niedermeier, and O. Suchý, *Parameterized complexity of arc-weighted directed Steiner problems*, SIAM Journal on Discrete Mathematics, 25 (2011), pp. 583–599. Cited on page 4.

[27] Eran Halperin, Guy Kortsarz, Robert Krauthgamer, Aravind Srinivasan, and Nan Wang, *Integrality ratio for group Steiner trees and directed Steiner trees*, SIAM J. Comput., 36 (2007), pp. 1494–1511. Cited on page 2.

[28] F. K. Hwang, D. S. Richards, and P. Winter, *The Steiner Tree Problem*, North-Holland, Amsterdam, 1992. Cited on page 1.

[29] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane, *Which problems have strongly exponential complexity?*, J. Comput. Syst. Sci., 63 (2001), pp. 512–530. Cited on pages 3 and 24.

[30] Mark Jones, Daniel Lokshtanov, M. S. Ramanujan, Saket Saurabh, and Ondrej Suchý, *Parameterized complexity of directed Steiner tree on sparse graphs*, in Algorithms - ESA 2013 - 21st Annual European Symposium, Sophia Antipolis, France, September 2-4, 2013.

Proceedings, Hans L. Bodlaender and Giuseppe F. Italiano, eds., vol. 8125 of Lecture Notes in Computer Science, Springer, 2013, pp. 671–682. Cited on page 1.

[31] Andrew B. Kahng and Gabriel Robins, *On Optimal Interconnections for VLSI*, Kluwer Academic Publisher, 1995. Cited on page 1.

[32] P. Klein and R. Ravi, *A nearly best-possible approximation algorithm for node-weighted Steiner trees*, Journal of Algorithms, 19 (1995), pp. 104 – 115. Cited on page 1.

[33] János Komlós and Endre Szemerédi, *Topological cliques in graphs 2*, Combinatorics, Probability & Computing, 5 (1996), pp. 79–90. Cited on page 7.

[34] Bernhard Korte, Hans Jürgen Prömel, and Angelika Steger, *Steiner trees in VLSI-layout*, in Paths, Flows and VLSI-Layout, 1990, pp. 185–214. Cited on page 1.

[35] Dániel Marx, *Can you beat treewidth?*, Theory of Computing, 6 (2010), pp. 85–112. Cited on page 21.

[36] Neeldhara Misra, Geevarghese Philip, Venkatesh Raman, Saket Saurabh, and Somnath Sikdar, *FPT algorithms for connected feedback vertex set*, J. Comb. Optim., 24 (2012), pp. 131–146. Cited on page 7.

[37] Daniel Mölle, Stefan Richter, and Peter Rossmanith, *Enumerate and expand: Improved algorithms for connected vertex cover and tree cover*, Theory Comput. Syst., 43 (2008), pp. 234–253. Cited on pages 2 and 4.

[38] Jesper Nederlof, *Fast polynomial-space algorithms using inclusion-exclusion*, Algorithmica, 65 (2013), pp. 868–884. Cited on pages 4 and 7.

[39] Rolf Niedermeier, *Invitation to Fixed-Parameter Algorithms*, Oxford University Press, Oxford, 2006. Cited on page 2.

[40] Geevarghese Philip, Venkatesh Raman, and Somnath Sikdar, *Polynomial kernels for dominating set in graphs of bounded degeneracy and beyond*, ACM Trans. Algorithms, 9 (2012), pp. 11:1–11:23. Cited on page 25.

[41] Marcin Pilipczuk, Michal Pilipczuk, Piotr Sankowski, and Erik Jan van Leeuwen, *Subexponential-time parameterized algorithm for Steiner tree on planar graphs*, in 30th International Symposium on Theoretical Aspects of Computer Science, STACS 2013, February 27 - March 2, 2013, Kiel, Germany, vol. 20 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013, pp. 353–364. Cited on page 4.

[42] ———, *Network sparsification for Steiner problems on planar and bounded-genus graphs*, in 55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014, IEEE Computer Society, 2014, pp. 276–285. Cited on page 4.

[43] H. J. Prömel and A. Steger, *The Steiner Tree Problem; a Tour through Graphs, Algorithms, and Complexity*, Vieweg, 2002. Cited on page 1.

[44] Alexander Zelikovsky, *A series of approximation algorithms for the acyclic directed Steiner tree problem*, Algorithmica, 18 (1997), pp. 99–110. Cited on page 2.

[45] Leonid Zosin and Samir Khuller, *On directed Steiner trees*, in Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, David Eppstein, ed., ACM/SIAM, 2002, pp. 59–63. Cited on page 2.