To appear in Networks, 2009

Fixed-Parameter Tractability Results for Full-Degree Spanning Tree and Its Dual[*]

Jiong Guo[†] Rolf Niedermeier

Sebastian Wernicke[‡]

Institut für Informatik,
Friedrich-Schiller-Universität Jena,
Ernst-Abbe-Platz 2,
D-07743 Jena, Germany.
{jiong.guo, rolf.niedermeier}@uni-jena.de.

August 7, 2009

Abstract

We provide first-time fixed-parameter tractability results for the NP-hard problems Maximum Full-Degree Spanning Tree and Minimum-Vertex Feedback Edge Set. These problems are dual to each other: In Maximum Full-Degree Spanning Tree, the task is to find a spanning tree for a given graph that maximizes the number of vertices that preserve their degree. For Minimum-Vertex Feedback Edge Set, the task is to minimize the number of vertices that end up with a reduced degree. Parameterized by the solution size, we exhibit that Minimum-Vertex Feedback Edge Set is fixed-parameter tractable and has a problem kernel with the number of vertices linearly depending on the parameter k. Our main contribution for Maximum Full-Degree Spanning Tree, which is W[1]-hard, is a linear-size problem kernel when restricted to planar graphs. Moreover, we present a dynamic programming algorithm for graphs of bounded treewidth.

Keywords: Fixed-parameter tractability, Problem kernel, Data reduction, Graph algorithms, Planar graphs

[*]A preliminary version of this paper appears in the proceedings of the 2nd International Workshop on Parameterized and Exact Computation (IWPEC'06), Zürich, Switzerland, September 2006, volume 4169 of Lecture Notes in Computer Science, pages 203–214, Springer.

[†]Supported by the Deutsche Forschungsgemeinschaft, Emmy Noether research group PIAF (fixed-parameter algorithms), NI 369/4.

[‡]Supported by the Deutsche Telekom Stiftung.

1 Introduction

The NP-hard problem Maximum Full-Degree Spanning Tree (FDST) is defined as follows.

> **Input**: An undirected graph $G = (V, E)$ and an integer $k \geq 0$.
> **Task**: Find a spanning tree T of G (called *solution tree*) in which at least k vertices have the same degree as in G.

Referring to vertices that maintain their degree as *full-degree* vertices and to the remaining ones as *reduced-degree* vertices, the task basically is to maximize the number of full-degree vertices. FDST is motivated by applications in water distribution and electrical networks [4, 8, 25]. However, also biological applications concerning the interference of reaction rates in metabolic networks are conceivable [30, Chapter 8]. Let n denote the number of graph vertices. FDST is a notoriously hard problem and, as such, not polynomial-time approximable within a factor of $O(n^{1/2-\epsilon})$ for any $\epsilon > 0$ unless NP-complete problems have randomized polynomial-time algorithms [4]. The approximability bound is almost tight in that Bhatia et al. [4] provide an algorithm with an approximation ratio of $\Theta(n^{1/2})$. FDST remains NP-hard in planar graphs; however, polynomial-time approximation schemes (PTAS) are known here [4, 8]. Broersma et al. [8] present further tractability and intractability results for various special graph classes. Gaspers et al. [19] present an $O(1.9172^n)$-time exact algorithm for FDST. By way of contrast, we study the parameterized complexity [14, 18, 26] of FDST.

The complement (dual) problem of FDST, called Minimum-Vertex Feedback Edge Set (VFES), is to find a feedback edge set[1] in a given graph such that this set is incident to as few vertices as possible. For our purposes, the following equivalent definition is more convenient.

> **Input**: An undirected graph $G = (V, E)$ and an integer $k \geq 0$.
> **Task**: Find a spanning tree T of G (called *solution tree*) in which at most k vertices have a degree smaller than in G.

In other words, we want to minimize the number of reduced-degree vertices. VFES is motivated by an application of placing pressure-meters in fluid networks [23, 27, 28]. Khuller et al. [23] show that VFES is APX-hard and present a polynomial-time approximation algorithm with ratio $2 + \epsilon$ for any fixed $\epsilon > 0$. Moreover, they develop a PTAS for VFES in planar graphs. As with FDST, the parameterized complexity of VFES has so far not been explored.

Parameterized by the respective solution size k, this work provides first-time parameterized complexity results for FDST and its dual VFES. Somewhat analogously to the study of approximability properties, we observe that FDST seems to be the harder problem when compared to its dual VFES: Whereas VFES is fixed-parameter tractable, FDST is W[1]-hard.[2] More specifically, our findings are as follows:

[1] A feedback edge set is a set of edges whose deletion destroys all cycles in a graph.
[2] The reduction is from Independent Set and the same as the one in [4, 23].

- VFES has a problem kernel with less than $4k$ vertices and it can be solved in $O(4^k \cdot k^2 + m)$ time for an m-edge graph.
- FDST becomes fixed-parameter tractable in the case of planar graphs. In particular, as the main technical contribution of this paper, we apply the linear kernelization framework for planar graph problems introduced in [21] to FDST and prove a linear-size problem kernel for FDST when restricted to planar graphs.
- For planar graphs, both VFES and FDST are solvable in subexponential time. More specifically, we present a tree decomposition-based dynamic programming, which directly implies fixed-parameter algorithms.

We remark that, when restricted to planar graphs, this work amends the so far few examples where both a problem and its dual possess linear-size problem kernels. The only other examples we are aware of (again restricted to planar graphs) are Vertex Cover and its dual Independent Set, and Dominating Set and its dual Nonblocker [2, 9, 10, 11].

2 Preliminaries and Notation

This section gives a brief overview about parameterized complexity and the notation that we use.

Parameterized complexity provides an at least two-dimensional framework for studying the computational complexity of problems [14, 18, 26]. One dimension is the input size n (as in classical complexity theory), and the other one is the *parameter* k (usually a positive integer). A problem is called *fixed-parameter tractable* (fpt) if it can be solved in $f(k) \cdot n^{O(1)}$ time, where f is a computable function only depending on k. This means that when solving a combinatorial problem that is fpt, the combinatorial explosion can be confined to the parameter.

A core tool in the development of fixed-parameter algorithms is polynomial-time preprocessing by *data reductions* [16, 20]. Here, the goal is for a given problem instance x with parameter k to transform it into a new instance x' with parameter k' such that the size of x' is upper-bounded by some function only depending on k, the instance (x, k) has a solution iff (x', k') has a solution, and $k' \leq k$. The reduced instance, which must be computable in polynomial time, is called a *problem kernel*, and the whole process is called *reduction to a problem kernel* or simply *kernelization*.

Downey and Fellows [14] develop a formal framework to show *fixed-parameter intractability* by means of *parameterized reductions*. A parameterized reduction from a parameterized language L to another parameterized language L' is a function that, given an instance (x, k), computes in $f(k) \cdot n^{O(1)}$ time an instance (x', k') (with k' only depending on k) such that $(x, k) \in L \Leftrightarrow (x', k') \in L'$. The basic complexity class for fixed-parameter intractability is called $W[1]$ and there is good reason to believe that $W[1]$-hard problems are not fpt [14, 18, 26].

Concerning notation, we only consider *undirected* graphs $G = (V, E)$, where V is the set of vertices and E the set of edges. By default, we use n to denote the number of vertices and m to denote the number of edges of a given graph. A vertex that is endpoint of an edge is said to be *incident* to that edge and *adjacent* to the other endpoint. The *neighborhood* $N(v)$ of a vertex $v \in V$ is the set of vertices that are adjacent to v. For a set of vertices $V' \subseteq V$, the *induced subgraph* $G[V']$ is the graph over the vertex set V' with edge set $\{\{v, w\} \in E \mid v, w \in V'\}$. Given a vertex $v \in V$, we use "$G - v$" as a shorthand notation for the induced subgraph $G[V \setminus \{v\}]$. We implicitly assume all paths that we deal with in this work to be *simple paths*, that is, a vertex is contained at most once in a path. The length of a path is defined as the number of edges used by the path. The distance $d(u, v)$ between two vertices u, v is the length of a shortest path between u and v. The distance $d(e, w)$ between an edge $e = \{u, v\}$ and a vertex w is the minimum of $d(u, w)$ and $d(v, w)$. If a graph can be drawn in the plane without edge crossings, then it is called a *planar graph.* A *plane graph* is a planar graph with a fixed embedding in the plane.

3 Minimum-Vertex Feedback Edge Set

Minimum-Vertex Feedback Edge Set (VFES), the dual of Maximum Full-Degree Spanning Tree (FDST), appears to be better tractable from a parameterized point of view than FDST. We subsequently present a simple kernelization with a kernel graph whose number of vertices linearly depends on the parameter (Section 3.1) and, based on this, an efficient fixed-parameter algorithm (Section 3.2) for VFES.

3.1 Data Reduction and Problem Kernel

To reduce a given instance of VFES to a problem kernel, we make use of two very simple data reduction rules already used by Khuller et al. [23].

Rule 1. Remove all degree-1 vertices.
Rule 2. For any two neighboring degree-2 vertices that do not have a common neighbor, contract the edge between them.

The correctness and the linear running time of these two rules are easy to verify; we call an instance of VFES *reduced* if the reduction rules cannot be applied any more.

Theorem 3.1. *A reduced instance* $(G = (V, E), k)$ *of* VFES *only has a solution tree if* $|V| < 4k$.

Proof. Assume that G has a solution tree T. Let X denote the set of the reduced-degree vertices in T. We partition the vertices in V into three disjoint subsets according to their degree in T, namely $V_{=1}$ contains all degree-1 vertices, $V_{=2}$ contains all degree-2 vertices, and $V_{\geq 3}$ contains all vertices of degree

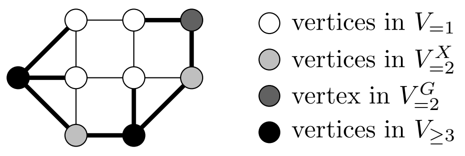

Figure 1: Given a spanning tree for a graph (bold edges), the proof of the linear kernel for VFES in Theorem 3.1 partitions them into four disjoint subsets as illustrated above (see proof for details).

higher than 2. Furthermore, let $V_{=2}^{X} := V_{=2} \cap X$ and $V_{=2}^{G} := V_{=2} \setminus V_{=2}^{X}$. The partition is illustrated in Figure 1. Since G does not contain any degree-1 vertices by Rule 1, every degree-1 vertex in T is a reduced-degree vertex. Hence, T can have at most k leaves and $|V_{=1}| \leq |X| \leq k$. Since T is a tree, this directly implies $|V_{\geq 3}| \leq k-2$.

As for $V_{=2}$, the vertices in $V_{=1} \cup V_{\geq 3}$ are either directly connected to each other or via a path P consisting of vertices from $V_{=2}$. Because Rule 2 contracts edges between two neighboring degree-2 vertices that have no common neighbor in the input graph, at least one of every two neighboring vertices of P has to be a reduced-degree vertex. Clearly, $V_{=2}^{X} \cup V_{=1} \subseteq X$. Since T is a tree, it follows that $|V_{=2}^{G}| \leq |V_{=1} \cup V_{\geq 3} \cup V_{=2}^{X}| - 1 \leq 2k-3$.

Overall, this shows that $|V| = |V_{=1} \cup V_{=2}^{X} \cup V_{=2}^{G} \cup V_{\geq 3}| < 4k$ as claimed. □

3.2 A Fixed-Parameter Algorithm

The problem kernel obtained in Theorem 3.1 suggests a simple fixed-parameter algorithm for Minimum-Vertex Feedback Edge Set: For $i = 1, \ldots, k$, we consider all $\binom{4k}{i}$ size-i subsets X of kernel vertices. For each of these subsets, all edges between the vertices in X are removed from the input graph, that is, they become reduced-degree vertices.[3] If the remaining graph is a forest, we have found a solution.[4] The correctness of this algorithm is obvious by its exhaustive nature, but on the running-time side it requires the consideration of $\sum_{i=1}^{k} \binom{4k}{i} > 9.45^k$ vertex subsets. The next theorem shows that we can do better because an exhaustive approach does not need to consider all vertices of the kernel but only those with degree at least three.

Theorem 3.2. *For an m-edge instance $(G = (V, E), k)$ of* VFES*, it can be decided in $O(4^k \cdot k^2 + m)$ time whether it has a solution tree.*

Proof. Given an instance $(G = (V, E), k)$ of VFES, we first perform the kernelization which needs $O(m)$ time. By Theorem 3.1, we know that the remaining

[3]Note that a vertex in X becomes a reduced-degree vertex only if it is adjacent to an other vertex in X.

[4]Since the input graph must be connected, we can add some edges from G that are between the vertices in X in order to reconnect connected components with each other and thus obtain a spanning tree.

graph only has a solution tree if it contains fewer than $4k$ vertices. We now partition the vertices in V according to their degree: vertices in $V_{=2}$ have degree two and $V_{\geq 3}$ contains all vertices with degree at least three. For every size-i subset $X_{\geq 3} \subseteq V_{\geq 3}$, $1 \leq i \leq k$, the following two steps are performed:

> **Step 1.** Remove all edges between vertices in $X_{\geq 3}$. Call the resulting graph $G' = (V, E')$.
> **Step 2.** For each edge $e \in E'$, assign it a weight $w(e) = m+1$ if it is incident to a vertex in $V_{\geq 3} \setminus X_{\geq 3}$ and a weight of 1, otherwise. Then, find a maximum-weight spanning tree for every connected component of G'. If the total weight of edges that are *not* in a spanning tree is at most $k-i$, then the original VFES instance $(G = (V, E), k)$ has a solution tree and the algorithm terminates; otherwise, the next subset $X_{\geq 3}$ is tried.

To justify Step 2, observe that the edges incident to vertices in $V_{\geq 3} \setminus X_{\geq 3}$ have to be in the solution tree because these vertices preserve their degree. Therefore, if a component in G' contains cycles we can only destroy these by removing edges between a vertex in $X_{\geq 3}$ and a vertex in $V_{=2}$. Moreover, removing such an edge results in exactly one additional reduced-degree vertex. Thus, searching for a maximum-weight spanning tree in the second step leads to a solution tree (the components can easily be reconnected by edges between vertices in $X_{\geq 3}$).

The running time of the algorithm is composed of the linear preprocessing time, the number of subsets that need to be considered, and the time needed for Steps 1 and 2. By Theorem 3.1, a reduced graph contains less than $4k$ vertices and, hence, $O(k^2)$ edges, upper-bounding the time needed for Steps 1 and 2 by $O(k^2)$. From the proof of Theorem 3.1, we know that $|V_{\geq 3}| < 2k$, and hence the overall running time is $O(m + \sum_{1 \leq i \leq k} \binom{2k}{i} \cdot k^2) = O(4^k \cdot k^2 + m)$. □

4 Maximum Full-Degree Spanning Tree in Planar Graphs

The parameterized reduction from Independent Set to Maximum Full-Degree Spanning Tree (FDST) that is given by Bhatia et al. [4] already shows that FDST is W[1]-hard with respect to the number k of full-degree vertices. For *planar* graphs, however, this does not hold; in this section, we show that there is a linear-size problem kernel for FDST in planar graphs. Basically, the proof for the problem kernel is achieved by contradiction; that is, we assume to be given a maximum-size solution $F \subseteq V$ to FDST on G and then show that either $|V| = O(|F|)$ holds or F cannot be of maximum size.[5] To this end, we show that the kernelization framework for planar graph problems [21] applies to FDST. More specifically, we prove that FDST admits the so-called *distance property* required by the framework (Section 4.1). Then, we define the

[5]This type of proof strategy appears first in work dealing with the Maximum Leaf Spanning Tree problem [15, 17].

regions and the corresponding region decomposition (Section 4.2) and introduce the data reduction rules (Section 4.3). Finally, we give a problem kernel size analysis in Section 4.4.

Note that the general framework from [21] only suggests a basic strategy for deriving linear kernelizations for planar graph problems. In other words, it provides a mathematical framework within which this task may be accomplished, but it completely leaves open the design and corresponding correctness proofs of concrete polynomial-time data reduction rules. For FDST, the design of data reduction rules and the kernel size analysis are highly problem-specific and require non-trivial combinatorial arguments. This is the main contribution we provide here. Throughout the next sections, we denote the planar graph that we are given as an instance of FDST by $G = (V, E)$; the maximum-size set of full-degree vertices in G is denoted F and we use k to denote its size. Finally, we assume that we are working with an arbitrary but fixed embedding of G in the plane; whenever this embedding is of relevance, we refer to G as being *plane* instead of planar.

4.1 Distance Property

This section shows that FDST admits the so-called distance property required by the framework [21] with $c_V = 2$ and $c_E = 2$, namely, for every vertex $u \in V$, there exists a full-degree vertex $v \in F$ with $d(u, v) \leq 2$, and, for every edge $e \in E$, there exists a full-degree vertex $v \in F$ with $d(e, v) \leq 2$. This follows from the following lemma.

Lemma 4.1. *Given a maximum-size set F of full-degree vertices, every reduced-degree vertex has distance at most two to at least one full-degree vertex.*

Proof. Assume for the purpose of contradiction that a reduced-degree vertex v has distance at least three to every vertex in F. Let T denote a spanning tree for G that corresponds to F, that is, a vertex has full degree in T if and only if it is in F. We now show how to iteratively transform T into a spanning tree T' that has one more full-degree vertex than T, namely v, which contradicts the maximum size of F. The transformation works as follows: Let e be an edge in G that has v as an endpoint and is not contained in T. To transform T into T', we add e to T. This induces a single cycle C in the resulting graph and we proceed by distinguishing two cases:

1. If C has length three, delete the cycle edge between the neighbors of v.
2. If C contains more than three edges, delete one of the two cycle edges in C that are incident to the two neighbors of v in the cycle, but not to v itself.

We now have a spanning tree T' for G where the vertex v has one more edge incident to it than in T. Also, all full-degree vertices of T remain full-degree in T' because v has distance at least three to every full-degree vertex whereas the transformation only affects the degree of vertices with distance at most two to v. After repeating the operation for all edges which are incident to v but not contained in T, we obtain a spanning tree for G with $k+1$ full-degree vertices, thus contradicting F being of maximum size. □

The following lemma can be shown in a similar way as Lemma 4.1 and will also be very useful for some of the later proofs.

Lemma 4.2. *Consider a maximum-size set F of full-degree vertices. From every reduced-degree vertex, there are at least two edge-disjoint length-at-most-2 paths into the set of full-degree vertices.*

Proof. Assume for the purpose of contradiction that there exists a reduced-degree vertex v for which every length-at-most-2 path from v to a full-degree vertex uses the same edge e. Let T be a spanning tree of the input graph corresponding to F. If we add to T all edges from the input graph that have v as an endpoint, then v becomes a full-degree vertex, but this also induces some cycles. Since we assumed that every length-at-most-2 path from v to a full-degree vertex uses the same edge e, the same argument as in the proof of Lemma 4.1 implies that each of the cycles contains a length-2 path v, u_1, u_2 where u_1 and u_2 are reduced-degree vertices. Hence, we can destroy all occurring cycles without changing the full-/reduced-degree status of any vertex by deleting the edge $\{u_1, u_2\}$ in every cycle. In this way, we obtain a spanning tree for the input graph that has all full-degree vertices already contained in T, and, additionally, the full-degree vertex v. This contradicts F being of maximum size. □

4.2 Regions and Region Decomposition

With the distance parameter $c_V = 2$ and $c_E = 2$, the framework from [21] suggests a decomposition of the input plane graph into regions that are closed subsets of the plane. The vertices are then divided into two categories based on whether they lie inside of the regions or not. These two vertex categories are handled separately by data reduction rules. To make this paper self-contained, we include the definition of regions and region decomposition:

Definition 4.1. A *region* $R(v, w)$ between two vertices $v, w \in F$ is a closed subset of the plane with the following properties:

1. The *boundary* of $R(v, w)$ is formed by two length-at-most-5 paths between v and w. (Note that these two paths do not need to be disjoint or simple.)

2. All vertices which lie on the boundary or strictly inside of the region $R(v, w)$ have distance at most two to at least one of the vertices v and w and all edges whose both endpoints lie on the boundary or strictly inside of the region $R(u, v)$ have distance at most two to at least one of u and v.

3. With the exception of v and w, none of the vertices which lie inside of the region $R(v, w)$ are from F.

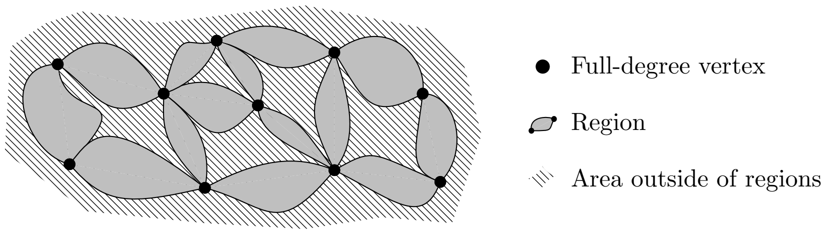

Figure 2: Schematic illustration for using the set of full-degree vertices F to decompose the plane input graph into regions. Not all vertices must lie inside of a region and can also lie in the “outside” area of the decomposition.

The vertices v and w are called the *anchor vertices* of $R(v, w)$. A vertex is said to *lie inside of* $R(v, w)$ if it is either a boundary vertex of $R(v, w)$ or if it lies strictly inside of $R(v, w)$. We use $V(R(v, w))$ to denote the set of vertices that lie inside of a region $R(v, w)$.

Using this definition, the graph can be partitioned by a so-called *region decomposition.*

Definition 4.2. An *F-region decomposition* of G is a set of regions $\mathcal{R}$ such that no two regions intersect (the boundaries of regions may touch each other, however).

For an F-region decomposition $\mathcal{R}$, we let $V(\mathcal{R}) := \bigcup_{R\in\mathcal{R}} V(R)$. An F-region decomposition $\mathcal{R}$ is called *maximal* if there is no region $R \notin \mathcal{R}$ such that $\mathcal{R}' := \mathcal{R} \cup \{R\}$ is an F-region decomposition with $V(\mathcal{R}) \subsetneq V(\mathcal{R}')$.

An example of a maximal F-region decomposition is given in Figure 2. The following upper bound of the number of the regions in a maximal region decomposition follows directly from a more general result in [21]. We defer its proof to the Appendix.

Lemma 4.3. *There is a maximal F-region decomposition $\mathcal{R}$ for the input graph G consisting of at most* $6 \cdot |F| - 12$ *regions.*

We call a maximal F-region decomposition *linear* if it consists of $O(k)$ regions.

4.3 Data Reduction Rules

According to the framework from [21], the next step is to define the so-called *private neighborhood* and *joint private neighborhood* and to derive data reduction rules to keep these two kinds of neighborhoods small. However, for FDST, it turns out that we only need to deal with a special form of joint private neighborhood, which is defined for two vertices u and v and consists of the common neighbors of u and v. The corresponding data reduction rules are as follows:

Reduction rules. Consider a graph $G = (V, E)$ that is part of an instance of FDST. For any two vertices $v \neq w \in V$ that have at least three common neighbors (that is, v and w satisfy $|N(v) \cap N(w)| \geq 3$), perform the following reductions:

1. Consider three vertices $u_1, u_2, u_3 \in N(v) \cap N(w)$:
   - 1.1 If $N(u_1) = \{v, w\}$ and additionally either $N(u_2) = \{v, w\}$ or $N(u_2) = \{u_3, v, w\}$, then remove u_2.
   - 1.2 If $N(u_1) = \{u_2, v, w\}$, $N(u_2) = \{u_1, u_3, v, w\}$, and $N(u_3) = \{u_2, v, w\}$, then remove u_3.
2. Consider four vertices $u_1, u_2, u_3, u_4 \in N(v) \cap N(w)$:
   - 2.1 If $N(u_1) = \{u_2, v, w\}$, $N(u_2) = \{u_1, v, w\}$, $N(u_3) = \{u_4, v, w\}$, and $N(u_4) = \{u_3, v, w\}$, then remove u_3 and u_4.
   - 2.2 If $N(u_2) = \{u_1, u_3, v, w\}$, $N(u_3) = \{u_2, u_4, v, w\}$, and $\{u_1, u_4\} \notin E$, then remove the edge $\{u_2, u_3\}$.
3. Consider five vertices $u_1, u_2, u_3, u_4, u_5 \in N(v) \cap N(w)$:
   If $N(u_2) = \{u_1, u_3, v, w\}$, $N(u_3) = \{u_2, v, w\}$, $N(u_4) = \{u_5, v, w\}$, and $N(u_5) = \{u_4, v, w\}$, then remove u_3.

Definition 4.3. A graph to which the data reduction has been exhaustively applied is called *reduced*.

The five cases of the data reduction are illustrated in Figure 3. Note that the data reduction can be applied to any graph, whether planar or not. The planarity is only required to guarantee that a linear-size kernel is obtained. The correctness and running time requirement of the data reduction can be established as follows:

Lemma 4.4. *The data reduction is correct and can be carried out in $O(n^3)$ time on an n-vertex graph.*

Proof. The main observation underlying the data reduction is that if there is a spanning tree for the input graph where a vertex u has full degree and a vertex u' with $N(u') \subseteq N(u)$ has reduced degree, then there always exists a spanning tree for G where u' has full degree instead of u (the full-/reduced-degree status for all other vertices does not change). This observation can be used to show the correctness of all five cases of the data reduction.

We first show the correctness of Case 1.1 in detail and then briefly indicate how to use the same idea to show the correctness of the other cases. To prove that Case 1.1 is correct, we show that the graph G has a spanning tree T with k full-degree vertices if and only if a graph G' that results from a single application of Case 1.1 on G has a spanning tree T' with k full-degree vertices.

"$\Rightarrow$" Let T be a spanning tree for the (unreduced) input graph G that contains k full-degree vertices. To prove the correctness of Case 1.1, observe

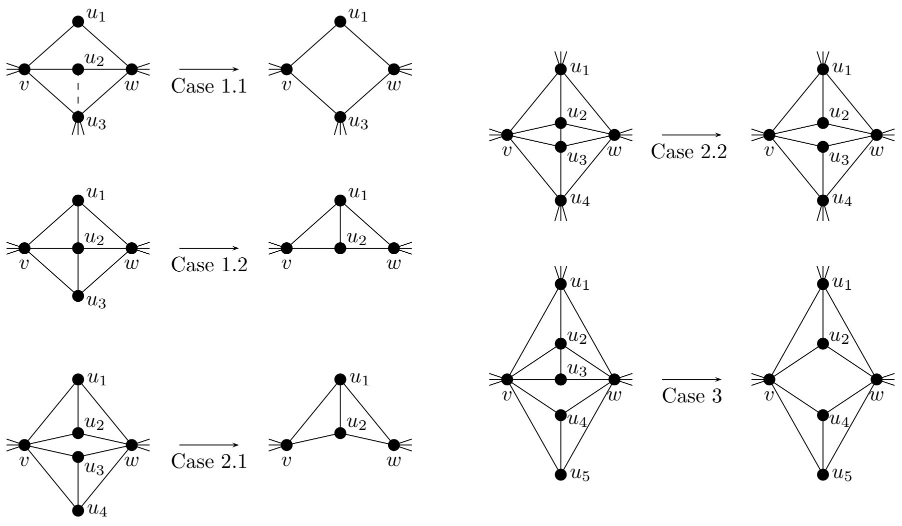

Figure 3: Illustration for the five cases of the data reduction for Maximum Full-Degree Spanning Tree that yields a linear-size problem kernel in planar graphs.

that at most one of the three vertices u_1, u_2, u_3 can have full degree in T: otherwise, there would be a cycle. If u_2 does not preserve its degree in T then—simply by deleting u_2—we can construct a spanning tree T' for the reduced graph G' that has the same number of full-degree vertices as T. If u_2 *does* have full degree, then u_1 must be a reduced-degree vertex or there would be a cycle in T. Therefore, we can construct a spanning tree T' for the reduced graph that preserves the degree of k vertices by letting u_1 have full degree instead of u_2; if there is an edge $\{u_2, u_3\}$ in T, then we additionally attach u_3 to v by an edge in T'.

"$\Leftarrow$" Let T' be a spanning tree for the reduced graph G' with k full-degree vertices. The only problematic case where we cannot simply add u_2 to T' to obtain a spanning tree T for G is when u_3 has full degree in T'. But in that case we can easily modify T' to let u_1 have full degree instead of u_3 by letting u_1 keep all of its incident edges and removing the edge $\{v, u_3\}$ in return.

This concludes our correctness proof for Case 1.1 of the data reduction. The other four cases can be shown by completely analogous means. For example, in Case 3, at most one of the five (or four) common neighbors of vertices v and w can have full degree in a spanning tree of the unreduced (or reduced) graph. With the same argument as for Case 1.1, we can always let u_5 be this full-degree vertex. Herein, a spanning tree of the unreduced graph can be transformed into a spanning tree of the reduced graph with the same number of full-degree vertices, and vice versa.

It remains to show the claimed running time. For this purpose, consider the following algorithm: For every vertex of degree between two and four, we assume it to be the vertex u_2 (that is, temporarily assign it the label "u_2"). There is a constant number of cases to distinguish on how to assign the labels "v," "w," "u_1," and "u_3" or any subset thereof to the neighbors of the designated vertex u_2. Given one such assignment, at most two additional vertices have to be found in order to identify a structure to which one of the data reduction cases applies; for example, in Case 3 of the data reduction, we must still find the vertices u_4 and u_5. To efficiently find the at most two remaining vertices, observe that one of these vertices has degree at most three and must already have all but one of its neighbors labeled by "v," "w," and "u_3." Hence, the remaining two vertices can be identified in $O(n)$ time by checking for each graph vertex of degree at most three whether it already has all but one labeled neighbor; if this is the case, the unlabeled neighbor constitutes the remaining second vertex. For every designated vertex u_2, the reduction rules are performed at most $O(n)$ times because each reduction removes either one of its neighbors or one of its incident edges. The cubic running time follows: We iterate over all graph vertices of degree at most four, which takes $O(n)$ time. There are $O(1)$ possible combinations to assign its neighbors the labels "v," "w," "u_1," and "u_3." As outlined above, the remaining vertices of the data reduction can be identified in $O(n)$ time, and each case is applied at most $O(n)$ times, leading to a total of $O(n^3)$ time. □

4.4 Problem Kernel Size

Now, we turn our attention to the mathematical analysis of the problem kernel size. Based on the linear upper bound on the number of the regions in the region decomposition (Lemma 4.3), it suffices to show that there are only constantly many vertices inside of any region and linearly many vertices outside of all regions.

Upper-bounding the number of vertices inside of regions. To prove that there is only a constant number of vertices inside of each region, graph structures that we call *diamonds*[6] are of great importance (see Figure 4 for two examples of a diamond).

Definition 4.4. Let v and w be two vertices in a plane graph G. A *diamond* $D(v, w)$ is a closed area of the plane that is bounded by two length-2 paths between v and w such that every vertex that lies inside this area is a neighbor of both v and w. If i vertices lie strictly inside a diamond, then it is said to have $(i + 1)$ *facets.*

It is vital that the data reduction eliminates those diamonds that have arbitrarily many facets: at most two vertices of any diamond can retain full degree in any spanning tree and hence the existence of arbitrarily large diamonds would prohibit a provable upper bound on the size of the graph.

[6]Note that standard graph theory uses the term "diamond" to denote a 4-cycle with exactly one chord. We abuse this term here for obvious reasons.

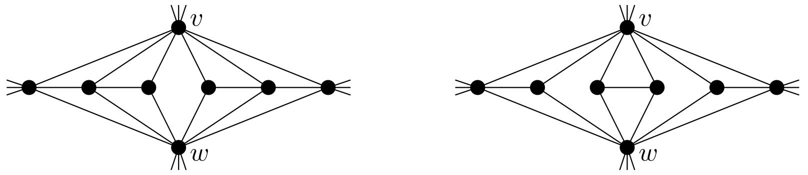

Figure 4: The only two possible five-facet diamonds (the worst-case diamonds, so to say) that a reduced plane graph can induce.

Lemma 4.5. *A reduced plane graph which is a diamond has at most five facets.*

Proof. Assume for the purpose of contradiction that the input diamond $D(v, w)$ has six facets. Let $u_1, \ldots, u_7$ denote the common neighbors of v, w such that u_1 and u_7 are the two vertices on the outer boundary of $D(v, w)$. Due to Case 1.1 of the data reduction, we have $N(u_i) \setminus \{v, w\} \neq \emptyset$ for $2 \leq i \leq 6$. By Case 2.2, there cannot be a length-3 path induced by any four of the vertices $u_1, \ldots, u_7$. Call two vertices u_i, u_{i+1} that are connected by an edge or three vertices u_i, u_{i+1}, u_{i+2} that induce a length-2 path a *block*. Thus, the seven vertices $u_1, \ldots, u_7$ are divided into several blocks, each block containing at most three vertices and having no edge to another block. Since $D(v, w)$ is reduced, only the blocks that contain u_1 or u_7 can have more than two vertices (Case 1.2) and there is at most one block that contains neither u_1 nor u_7 (Case 2.1). But then Case 3 of the data reduction applies, a contradiction to the input graph being reduced. We can thus conclude that a reduced diamond can have at most five facets. □

The only two possible five-facet diamonds that can occur in a reduced plane graph are shown in Figure 4. For the same reason that we must avoid arbitrarily large diamonds by means of the data reduction, it is important that there cannot be arbitrarily many length-2 paths between two vertices of the input graph. Thus, we now generalize Lemma 4.5 in order to obtain an upper bound on the number of length-2 paths between two vertices.

Lemma 4.6. *Let v and w be two vertices in the reduced plane graph G such that an area A of the plane is enclosed by two length-2 paths between v and w. If neither the middle vertices of enclosing paths nor the vertices lying strictly inside of A are contained in F, then the following holds:*

1. *If $v, w \notin F$, then at most eight length-2 paths from v to w lie inside of A.*
2. *If either $v \notin F$ or $w \notin F$, then at most sixteen length-2 paths from v to w lie inside of A.*

Proof. We use Lemma 4.5 to show that the presence of more length-2 paths than claimed contradicts that F has maximum size. For the first case, that is, neither v nor w is contained in F, assume that there are nine length-2 paths between v and w that lie inside of A. Without loss of generality, let these be embedded as shown in Figure 5. Recall that none of the vertices $u_1, \ldots, u_9$ is

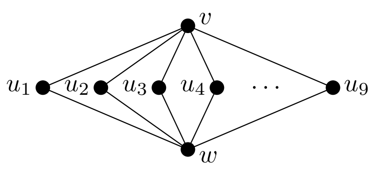

Figure 5: Assumed embedding of the vertices $u_1, \ldots, u_9$ in the proof of Proposition 4.6.

in F by the prerequisites of the lemma. Consider the six-facet diamond induced by $\{v, w, u_2, \ldots, u_8\}$. By Lemma 4.5, there must be a vertex u' that lies inside this induced diamond but is not adjacent to both v and w. Since u' lies inside the induced diamond, it cannot be adjacent to u_1 and u_9. By Lemma 4.1, there must be at least one full-degree vertex in the two-neighborhood of u'. But since u' is not adjacent to both v and w, all length-2 paths from u' to a full-degree vertex use the same edge, which violates Lemma 4.2.

Proving the second case is based on having proved the first one: Assume for the purpose of contradiction that there are seventeen length-2 paths between v and w. Without loss of generality, we let $v \in F$ and $w \notin F$. The outermost paths form the bound of an induced sixteen-facet diamond. The middle path in an embedding further separates this diamond into two induced eight-facet diamonds. Assume that we remove v from F. Then, using the same argument as in the first case, each eight-facet diamond encloses a vertex that can be made full-degree. Thus, removing v from F and adding these two vertices, we have a larger solution than the original F, a contradiction to its maximum size. $\square$

This upper bound on the number of length-2 paths between two vertices in a reduced graph plays a crucial role in upper-bounding the total number of vertices that lie inside and outside of a region. We furthermore require one more lemma to aid us in upper-bounding the number of neighbors that a boundary vertex of a region can have.

Lemma 4.7. *Consider a vertex u in the reduced plane graph G that is adjacent to at least 81 reduced-degree vertices $v_1, \ldots, v_{81} \in V \setminus F$. Then, in the graph $G - u$, there are at least three full-degree vertices that are reachable from vertices in the set $\{v_1, \ldots, v_{81}\}$ via length-at-most-2 paths.*

Proof. Let G_u denote the graph $G - u$. Without loss of generality, we assume that the vertices $v_1, \ldots, v_{81}$ are embedded in a clockwise fashion around the vertex u, that is, v_1 is followed by v_2, v_2 is followed by v_3, and so on.

Our strategy to prove the lemma is to show that we can always find at least three vertices v_h, $v_{h'}$, and $v_{h''}$ among the 81 vertices $v_1, \ldots, v_{81} \in V \setminus F$ that have a mutual distance of at least three to each other in G_u. This suffices to show the lemma because, if there are at most two full-degree vertices a and b then we set $F' := F \setminus \{a, b\} \cup \{v_h, v_{h'}, v_{h''}\}$. By employing the same construction that we used in the proof of Lemma 4.2, we can construct a spanning tree for G with

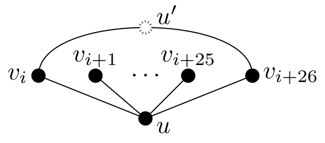

Figure 6: Enclosure of 25 vertices $v_{i+1}, \ldots, v_{i+25}$ in the plane as discussed in the proof of Lemma 4.7.

vertices in F' having full degree. This would contradict that F is of maximum size.

We now prove the following claim: If two vertices v_i and v_{i+26}, where $1 \leq i \leq 55$, have distance at most two in G_u, then there exists a vertex v_h with $i < h < i + 26$ that has distance at least three to all vertices v_j with $j < i$ or $j > i + 26$. To see this claim, assume that there are two vertices v_i and v_{i+26}, $1 \leq i \leq 55$, that have distance at most two to each other in G_u. This implies an enclosure of the 25 vertices $v_{i+1}, \ldots, v_{i+25}$ in the plane as shown in Figure 6. Without loss of generality[7], let us assume that v_i and v_{i+26} have distance *exactly* two to each other, that is, we assume that there exists a path $v_i u' v_{i+26}$ with $u' \neq u$ in G. We consider four subsets of $\{v_{i+1}, \ldots, v_{i+25}\}$ that are defined according to these vertices' adjacency to v_i, v_{i+26}, and u', the first three containing the vertices adjacent to v_i, v_{i+26}, or u', respectively, and the last one containing the vertices adjacent to none of the three. Since none of $v_{i+1} \ldots, v_{i+25}$ are full-degree vertices, each of the first three subsets forms together with u and one vertex from $\{v_i, v_{i+26}, u'\}$ an area enclosed by two length-2 paths. Due to the first part of Lemma 4.6, each of these three subsets contains at most 8 vertices. Thus, at least one vertex is in the fourth subset, that is, it is not adjacent to v_i, v_{i+26}, and u'. By a pigeonhole argument $(24 < 25)$, this means that there is a vertex v_h among the 25 vertices $v_{i+1}, \ldots, v_{i+25}$ that is not adjacent to any of v_i, v_{i+26}, and u', which in turn implies that v_h has distance at least three to all vertices v_j with $j < i$ or $j > i + 26$ in G_u.

With the claim proved, we can now proceed to show the lemma. For this purpose, we start out by considering the two neighbors v_1 and v_{27} of u and distinguish two cases based on whether these two vertices have distance at most two to each other in G_u:

1. As the first case, assume that v_1 and v_{27} have distance at least three to each other in G_u. Consider the vertex v_{53}: If this vertex has distance at least three to both v_1 and v_{27} in G_u, then we are done because we have found three neighbors of u—namely v_1, v_{27}, and v_{53}—that mutually have distance at least three to each other in G_u. Otherwise, if v_{53} has distance at most two to either v_1 or v_{27}, then we know by our above claim that there exists a vertex v_h, $1 < h < 53$, with distance at least three to all vertices v_j, $j > 53$, in G_u. Now consider the vertices v_{54} and v_{80}. If these have distance at least three to each other, then together with v_h they form

[7]The distance-1 case is much easier to show.

a set of three neighbors of u with mutual distance at least three in G_u. Finally, if v_{54} and v_{80} have distance at most two to each other in G_u, then there exists a vertex $v_{h'}$ with $54 < h' < 80$ that has distance at least three in G_u to all vertices v_j with $j < 54$ or $j > 80$. This implies that v_h, $v_{h'}$, and v_{81} have a mutual distance of at least three in the graph G_u.

2. As the second case, assume that v_1 and v_{27} have distance at most two to each other in G_u. Then our claim from above tells us that there exists a vertex v_h, $1 < h < 27$, that has distance at least three in G_u to all vertices v_j with $j > 27$. We now consider the vertices v_{28} and v_{54}: If they have distance at least three to each other in G_u, then together with v_h we have again a set of three vertices with distance at least three to each other in G_u. Otherwise, there exists a vertex $v_{h'}$ with $28 < h' < 54$ that has distance at least three in G_u to all vertices v_j with $j < 28$ or $j > 54$, which means that v_h, $v_{h'}$, and v_{55} form a set of three vertices with mutual distance at least three in G_u.

□

Using the upper bound from Lemma 4.6 on the number of length-2 paths between two vertices and the upper bound on the number of neighbors that we have just established, we can finally obtain a constant upper bound on the number of vertices that lie inside of a region.

Lemma 4.8. *The number of vertices in every region $R(v, w)$ of an F-region decomposition $\mathcal{R}$ is upper-bounded by a constant.*

Proof. To show the lemma, we partition the vertices that lie strictly inside of a region $R(v, w)$ into two disjoint subsets V_A and V_B and separately upper-bound their cardinality. (Observe that the number of boundary vertices is automatically upper-bounded by 10 due to Definition 4.1.)

- The subset V_A contains all vertices that are adjacent to a boundary vertex.
- The subset V_B contains all vertices that are not adjacent to a boundary vertex.

To upper-bound the size of V_A, we partition this set into two subsets V_1^A and $V_{\geq 2}^A$ depending on the number of adjacent boundary vertices: V_1^A contains all vertices that are adjacent to exactly one vertex of the boundary and $V_{\geq 2}^A$ contains all vertices that are adjacent to at least two vertices of the boundary of $R(v, w)$.

To upper-bound the number of vertices in V_1^A, consider a boundary vertex v_b of $R(v, w)$. Every vertex in V_1^A that is adjacent to v_b in G has distance at least three in the graph $G - v_b$ to all full-degree vertices except for, possibly, the two anchor vertices v and w. By Lemma 4.7, this implies that a boundary vertex must not be adjacent to more than 80 vertices from V_1^A. Thus, we have $|V_1^A| \leq 790 = O(1)$ because a region has at most ten boundary vertices.

To upper-bound $|V^A_{\geq 2}|$, consider the graph consisting of the boundary vertices of the region $R(v, w)$. Since every vertex in $V^A_{\geq 2}$ connects at least two vertices of the boundary by a length-2 path, we insert for each vertex in $V^A_{\geq 2}$ exactly one edge between two of its neighbors that lie on the boundary $R(v, w)$ and merge multiple edges in the process. The resulting graph clearly must be planar.[8] By the well-known Euler formula a planar graph with n vertices contains at most $3n - 6$ edges, that is, at most 24 edges in our case because the region has at most $n = 10$ boundary vertices. By Lemma 4.6, each edge can only stand for a constant number of length-2 paths, and hence we have shown that $|V^A_{\geq 2}|$ is upper-bounded by $24 \cdot O(1) = O(1)$.

Note that each vertex in V_B must be adjacent to some vertex in V_A because, otherwise, it would have distance at least three to all boundary vertices and violate Lemma 4.1. To upper-bound the size of V_B, observe that each vertex in this set must be adjacent to at least two vertices of V_A due to Lemma 4.2. This means that every vertex in V_B connects at least two vertices from V_A by a length-2 path. Replacing, for each vertex in V_B, exactly one of such paths by an edge and merging any multiple edges that emerge, we obtain a planar graph[9] containing at most $3|V_A| - 6$ edges. By Lemma 4.6, each such edge can stand for at most 8 length-2 paths, which in turn bounds $|V_B|$ by $(3 \cdot |V_A| - 6) \cdot 8 = O(1)$.

Overall, we have shown that the vertices that lie strictly inside of $R(v, w)$ can be partitioned into two sets V_A and V_B where $|V_A| = O(1)$ and $|V_B| = O(1)$. Thus, as claimed, only $O(1)$ vertices lie inside of any region $R(v, w)$. $\square$

Having proven the linear upper bound on the number of regions (Lemma 4.3 and the constant upper bound on the number of vertices each of these contains (Lemma 4.8), we obtain an $O(k)$ upper-bound on the number of vertices that lie inside of regions.

Proposition 4.1. Given a linear F-region decomposition $\mathcal{R}$ for the reduced plane graph G, the number of vertices that lie inside of regions is $O(k)$.

Proof. Due to Lemma 4.3, a linear F-region decomposition contains $O(|F|) = O(k)$ regions, each of which has $O(1)$ vertices lying inside of it by Lemma 4.8. $\square$

Upper-bounding the number of vertices outside of regions. Our proof strategy for the upper bound on the number of vertices that lie outside of the regions of a linear F-region decomposition is—similar to the proof of the upper bound on the number of vertices inside of regions—to partition the vertices that lie outside of regions into two sets based on their adjacency to the full-degree vertices in F, and separately upper-bound the sizes of each of the two sets. As the first of these two sets, we consider those vertices outside of regions that are adjacent to some full-degree vertex.

[8]More precisely, it must even be *outerplanar*, that is, it has a crossing-free embedding in the plane such that all vertices are on the same face, but this detail does not need to concern us here.

[9]Analogously to above, this graph is actually outerplanar.

Lemma 4.9. *Given a linear F-region decomposition $\mathcal{R}$, the number of vertices that lie outside of regions and are adjacent to a full-degree vertex is $O(k)$.*

Proof. Let V_A denote the set of vertices that lie outside of regions and are adjacent to a full-degree vertex, that is, the lemma claims $|V_A| = O(k)$. Note that a vertex $u \in V_A$ can be adjacent to at most one full-degree vertex v: If it is adjacent to two full-degree vertices, these would form a region together with u that could be added to $\mathcal{R}$, thus contradicting the maximality of the region decomposition. To upper-bound the size of V_A, we partition this set into two subsets, namely a subset V_A^b that contains all vertices from V_A that are adjacent to a boundary vertex other than v (which, as just observed, must not be contained in F) and a subset $V_A^{\not b}$ that contains the remaining vertices.

To upper-bound the size of V_A^b, consider a vertex $u \in V_A^b$ that is adjacent to a full-degree vertex v. By definition of the set V_A^b, the vertex u is adjacent to some boundary vertex $w \notin F$. The main observation now is that this boundary vertex w must be adjacent to v: otherwise, the vertices u, v, w and a part of the boundary on which w lies form a region which could be added to $\mathcal{R}$, contradicting the maximality of the region decomposition. It is now easy to see that $|V_A^b| = O(k)$: Since the region decomposition $\mathcal{R}$ is linear, the number of boundary vertices that are adjacent to a full-degree vertex is $O(k)$. Furthermore, by Lemma 4.6, at most 16 vertices that are adjacent to a full-degree vertex can be connected to the same boundary vertex. Hence, we have $|V_A^b| = 16 \cdot O(k) = O(k)$.

It remains to upper-bound $|V_A^{\not b}|$. Consider a vertex $u \in V_A^{\not b}$ that is adjacent to a full-degree vertex v. Recall that by now we already know u to be adjacent to exactly one full-degree vertex and to no boundary vertex except v. This means that, with the exception of v, all neighbors of u lie outside of a region. Furthermore, none of these neighbors can be adjacent to a full-degree vertex because this would lead to a region that could be added to $\mathcal{R}$, contradicting its maximality. We have thus established that every vertex $u \in V_A^{\not b} \cap N(v)$ has distance at least three to all vertices in $F \setminus \{v\}$. This allows us to upper-bound $|V_A^{\not b} \cap N(v)|$ by 80 (Lemma 4.7), which directly implies that $|V_A^{\not b}| \leq 80 \cdot k = O(k)$.[10]

Overall, we have thus shown that $|V_A| = O(k)$ as claimed. ☐

We now turn our attention to upper-bounding the remaining vertices that lie outside of regions, that is, those vertices that are not adjacent to a full-degree vertex.

Lemma 4.10. *Given a linear F-region decomposition $\mathcal{R}$, the number of vertices that lie outside of regions and are not adjacent to a full-degree vertex is $O(k)$.*

Proof. Let V_B denote the vertices that lie outside of regions and are not adjacent to a full-degree vertex. Furthermore, let V_b denote the set of boundary vertices in the given region decomposition and—as in the previous proof—let V_A denote

[10]Note that this bound could easily be improved by strengthening Lemma 4.7 for the case where no full-degree vertices are reachable from the reduced-degree neighbors of a vertex v via a length-at-most-three path in the graph $G - v$.

the set of all vertices that lie outside of regions and are adjacent to one or more full-degree vertices.

The key idea of the proof is that, in order to show $|V_B| = O(k)$, it suffices to prove that every vertex in V_B is adjacent to at least two vertices of $V_A \cup V_b$. To show this sufficiency, define a graph $G' = (V', E')$ with vertex set $V' := V_A \cup V_b$ and the edge set E' constructed as follows: For each vertex in V_B that has at least two adjacent vertices in $V_A \cup V_b$, add an edge that connects any two of these vertices, merging any multiple edges in the process. Clearly, the graph G' is planar because the input graph G is planar and we have simply replaced some of its degree-at-least-2 vertices by edges. As in previous proofs, we can use the Euler formula to obtain

$$|E'| \leq 3|V'| - 6 < 3|V_A \cup V_b| < 3 \cdot O(k) = O(k)\,.$$

By Lemma 4.6, every edge in E' can stand for at most eight vertices in V_B. Hence, if every vertex in V_B is adjacent to at least two vertices from $V_A \cup V_b$, this also means that $|V_B| = 8 \cdot O(k) = O(k)$.

It remains to show that every vertex in V_B is indeed adjacent to at least two vertices of the set $V_A \cup V_b$. Using Lemma 4.2, we know that each vertex in V_B has at least two edge-disjoint length-2 paths to full-degree vertices. In other words, each vertex $u \in V_B$ is adjacent to two vertices v and w that themselves are adjacent to some full-degree vertex. If one of these two vertices lies outside of a region, it belongs to the set V_A; if it lies inside of a region, then—since u does not lie inside of a region—it is a boundary vertex and thus belongs to the set V_b. Hence, every vertex in V_B is adjacent to at least two vertices from the set $V_A \cup V_b$, which—as argued above—proves the lemma. □

Proposition 4.2. Given a linear F-region decomposition $\mathcal{R}$, the number of vertices that do not lie inside of a region is $O(k)$.

Proof. Lemma 4.9 gives an $O(k)$ upper bound on the number of vertices that lie outside of regions and are adjacent to at least one full-degree vertex; Lemma 4.10 gives an $O(k)$ upper bound on the number of vertices that lie outside of regions and are not adjacent to any full-degree vertex. □

With Propositions 4.1 and 4.2 established, we can prove the main result of this section:

Theorem 4.1. Maximum Full-Degree Spanning Tree *in planar graphs admits an $O(k)$-vertex problem kernel, which is computable in $O(n^3)$ time.*

Proof. We have shown in Lemma 4.3 that the input graph can be decomposed into $O(k)$ so-called regions such that both the number of vertices that lie inside of regions is upper-bounded by $O(k)$ (Proposition 4.1) and the number of vertices that lie outside of regions is upper-bounded by $O(k)$ (Proposition 4.2), leading to the claimed overall $O(k)$ upper bound. The running time follows from Lemma 4.4. □

5 A Tree Decomposition-Based Algorithm

In this section, we give a tree decomposition-based algorithm that solves both MINIMUM-VERTEX FEEDBACK EDGE SET and MAXIMUM FULL-DEGREE SPANNING TREE in $O((2\omega)^{3\omega} \cdot \omega \cdot n)$ time on an n-vertex graph with a given tree decomposition of width ω. The tree decomposition and treewidth of a graph are defined as follows (refer to [5, 6, 7, 24] for surveys):

Definition 5.1. A *tree decomposition* for a graph $G = (V, E)$ is a pair (T, X) where T is a tree and X consists of subsets of V called *bags*. Each node i in T is associated with exactly one bag $X_i \in X$ such that the following three conditions are satisfied:

1. Every vertex is contained in at least one bag.
2. For every edge, there exists a bag containing both of its endpoints.
3. For every vertex $v \in V$, the nodes that are associated with bags that contain v form a connected subtree in T.

The *width* of a tree decomposition is the size of the largest bag minus one. The *treewidth* ω of a graph is the minimum width over all possible tree decompositions.

The running time of our algorithm only depends on the treewidth ω of the input graph; in particular, it solves both FDST and its dual problem VFES in the same running time. While Broersma et al. [8] have already shown that both VFES and FDST are solvable in linear time for graphs with bounded treewidth, they did so by expressing the problem in monadic second-order logic. The linear-time solvability follows then from a result of Arnborg et al. [3]. However, this approach neither provides an exact dependency of the running time on the treewidth—as a rule, the involved constant factors are huge—nor does it provide much insight into how the bounded treewidth is exploited. By way of contrast, we give a concrete dynamic programming algorithm based on a tree decomposition of the input graph.

To simplify the presentation of our algorithm, we assume that the given tree decompositions are *nice*:

Definition 5.2. A tree decomposition (T, X) is called *nice* if the tree T is rooted and the following conditions are satisfied:

1. Every node of T has at most two children.
2. If a node i has two children j and k, the bags of all three nodes are the same, that is, $X_i = X_j = X_k$. In this case, i is called a JOIN NODE.
3. If a node i has one child j, then one of the following holds:
   - $|X_i| = |X_j| + 1$ and $X_j \subsetneq X_i$. In this case, i is called an INTRODUCE NODE.

- $|X_i| = |X_j| - 1$ and $X_i \subsetneq X_j$. In this case, i is called a FORGET NODE.

Assuming that we are working with nice tree decompositions is not a restriction because every tree decomposition can be efficiently transformed into a nice tree decomposition [24, Lemma 13.1.3].

We present our tree decomposition-based algorithm from the "perspective" of FDST, that is, we want to maximize the number of full-degree vertices. Since the running time only depends on the treewidth of the input graph, note that this algorithm also solves the dual problem VFES in the same time. In what follows, we use T_i to denote the subtree of the decomposition tree T that is rooted at node i. The graph G_i denotes the subgraph of G that is induced by the vertices of the bags associated with the nodes of T_i, that is, $G_i = G[\bigcup_{j \in T_i} X_j]$. With "partitioning a graph" we subsequently mean a partition of the vertex set and a "component" of a partition is the subgraph induced by a vertex subset in the partition.

In order to compute a spanning tree with maximum number of full-degree vertices, we have to solve two problems: making as many as possible vertices full-degree and avoiding cycles induced by "full-degree edges," that is, the edges incident to full-degree vertices. Our solution strategy is, at every stage of the dynamic programming (at node i with bag X_i), to consider all *valid annotations* of the graph G_i. A valid annotation of G_i consists of an assignment of vertices in G_i to the two states, full-degree and reduced-degree, and a partition of G_i into components, each of which has one of the following two properties:

P1. If a component contains only one vertex v, then v is either reduced-degree or isolated in G_i.

P2. If a component contains more than one vertex, then the edges in G_i incident to the full-degree vertices in this component induce a spanning tree of this component.

Observe that, given a valid annotation of a connected graph G, one can easily construct a spanning forest of G from the spanning trees of the components in the partition of the annotation. The full-/reduced-degree states of the vertices in this spanning forest correspond to the vertex states defined by the assignment of the annotation. Conversely, each spanning tree of G corresponds clearly to a valid annotation of G. Based on this observation, the dynamic programming algorithm computes bottom-up, from the leaves to the root of the decomposition tree T, all valid annotations of the input graph G and outputs one with maximum number of full-degree vertices.

Obviously, for a leaf node i of T, all valid annotations can be easily enumerated in $O((2|X_i|)^{|X_i|} \cdot |X_i|)$ time, since there are $O(2^{|X_i|})$ possible assignments for vertices in X_i and $O(|X_i|^{|X_i|})$ possible partitions of G_i. The validity of an annotation can be tested in $O(|X_i|)$ time. Concerning a FORGET NODE i with a child node j, since G_i has the same vertex set as G_j, every valid annotation of G_j is also a valid annotation of G_i. The computation at a FORGET NODE is then trivial.

Since the graph G_i of an introduce node i has one more vertex x than the graph G_j associated with its child node j, it is possible that a component defined by a partition of a valid annotation of G_j is extended by x or several components are joined together by x. Therefore, for every valid annotation of G_i, we have to examine for all valid annotations of G_j whether they can be transformed into this valid annotation of G_i by setting x full-degree or not. Note that after setting the full-/reduced-degree state of x we can easily decide whether x should be added to a component or whether some components should be merged. Since x has only neighbors in X_j, this decision can be made in $O(|X_i|)$ time for any given valid annotation of G_j. Altogether, we have a running time of $O((2|X_i|)^{2|X_i|} \cdot |X_i|)$ for one introduce node.

The computation at a join node i with two children j and l works in a similar way as at an introduce node. Here, we have to consider the case that several components of a valid annotation of G_j can be merged together by several components of a valid annotation of G_l. Therefore, for one valid annotation of G_i, all possible combinations of valid annotations from G_j and G_l have to be examined, which results in a running time of $O((2|X_i|)^{3|X_i|} \cdot |X_i|)$.

At the root of T the valid annotation of G with maximum number of full-degree vertices is then returned as the output. The technical but more or less standard details of the dynamic programming algorithm are deferred to the Appendix. We arrive at the following theorem.

Theorem 5.1. *For an n-vertex graph G with a given width-ω tree decomposition (T, X), both* Minimum-Vertex Feedback Edge Set *and* Maximum Full-Degree Spanning Tree *can be solved in $O((2\omega)^{3\omega} \cdot \omega \cdot n)$ time.*

Proof. The most time-consuming computation takes place at a join node i with a running time of $O((2|X_i|)^{3|X_i|} \cdot |X_i|)$. Since $|X_i| \leq \omega$, we have the claimed overall running time. □

Using the linear problem kernels for VFES and FDST that were established in Theorems 3.1 and 4.1, Theorem 5.1 implies subexponential-time algorithms for these problems when restricted to planar graphs:

Corollary 5.1. *In n-vertex planar graphs,* Minimum-Vertex Feedback Edge Set *is solvable in $O(2^{O(\sqrt{k}\log k)} + k^5 + n)$ time and* Maximum Full-Degree Spanning Tree *in $O(2^{O(\sqrt{k}\log k)} + k^5 + n^3)$ time, where k is the number of degree-reduced vertices or full-degree vertices, respectively.*

Proof. Both problems have linear-size kernels in planar graphs, that is, the reduced graphs have at most $O(k)$ vertices after performing the respective kernelization. The claim follows from Theorem 5.1 and the facts that planar $O(k)$-vertex graphs have treewidth $\omega = O(\sqrt{k})$ and that a corresponding tree decomposition of width $3\omega/2$ can be computed in $O(k^5)$ time [29]. □

While Theorem 5.1 means that VFES can be solved in subexponential time in planar graphs, the constants hidden in the O-notation are very large and hence, for reasonable sizes of parameter k, the algorithm from Theorem 3.2 is much more efficient than the one obtained in Theorem 5.1.

There are two further observations concerning subexponential-time fixed-parameter algorithms for VFES and FDST:

1. Both problems are $\Theta(r^2)$-minor-bidimensional, that is, the solution sizes on $r \times r$-grid graphs are $\Theta(r^2)$ and the parameter "solution size" does not increase when taking minors (see Demaine and Hajiaghayi [12] for a recent survey on the already rich literature on this young field). Given this, one can conclude, together with Theorem 5.1, that VFES and FDST can be solved in subexponential time on superclasses of planar graphs based on so-called parameter-treewidth bounds used in the same style as employed in the proof of Corollary 5.1. We refer to the survey [12] and the literature cited there for any details.

2. It seems plausible that by using so-called Catalan structures in dynamic programming our approach can be specialized for planar graphs such that one might get rid of the $\log k$-factor in the exponent of Corollary 5.1. The details, however, are tricky and we only refer to Dorn et al. [13] for more on that.

6 Conclusion

Our main focus in this work was on problem kernels of linear size. As Fellows [16] puts in his recent keynote speech, kernelization is a "humble strategy for coping with hard problems, almost universally applied." Thus, our results contribute to a practically important as well as theoretically deep field of algorithmic research (also see the survey [20]). In particular, it might be worth noting that the data reduction rules for Maximum Full-Degree Spanning Tree work on arbitrary graphs, not only for planar graphs. Experiments with data reduction rules developed for Dominating Set on planar graphs applied to non-planar, sparse graphs exhibited encouraging results on some real-world networks [1, 2]. Perhaps similar results can be achieved here. Thus, pursuing empirical studies (similar to Bhatia et al. [4]) with real-world data in order to explore the practical usefulness of our algorithms and problem kernelizations should be a worthwhile topic for future research. In this way, one might provide a further study on the practical usefulness of fixed-parameter algorithm design techniques (cf. [22]).

Our main technical contribution is the proof of the linear-size problem kernel for FDST in planar graphs. It is easily conceivable that there is room for significantly improving the involved worst-case constant factors by refined mathematical analysis—the situation is comparable (but perhaps even more technical) with analogous results for Dominating Set in planar graphs [2, 9]. Having obtained linear problem kernel sizes for a problem and its dual, the way for applying the lower bound technique for kernel size due to Chen et al. [9] now seems open.

Acknowledgement: We are grateful to two anonymous *Networks* referees for their constructive feedback.

References

[1] J. Alber, N. Betzler, and R. Niedermeier. Experiments on data reduction for optimal domination in networks. *Annals of Operations Research*, 146(1):105–117, 2006. 23

[2] J. Alber, M. R. Fellows, and R. Niedermeier. Polynomial-time data reduction for Dominating Set. *Journal of the ACM*, 51:363–384, 2004. 3, 23, 27

[3] S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12:308–340, 1991. 20

[4] R. Bhatia, S. Khuller, R. Pless, and Y. Sussmann. The full degree spanning tree problem. *Networks*, 36:203–209, 2000. 2, 6, 23

[5] H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209:1–45, 1998. 20

[6] H. L. Bodlaender. Treewidth: Characterizations, applications, and computations. In *Proc. 32nd WG*, volume 4271 of *LNCS*, pages 1–14. Springer, 2006. 20

[7] H. L. Bodlaender and A. M. C. A. Koster. Combinatorial optimization on graphs of bounded treewidth. *The Computer Journal*, 51(3):255–269, 2008. 20

[8] H. J. Broersma, A. Huck, T. Kloks, O. Koppius, D. Kratsch, H. Müller, and H. Tuinstra. Degree-preserving trees. *Networks*, 35:26–39, 2000. 2, 20

[9] J. Chen, H. Fernau, I. A. Kanj, and G. Xia. Parametric duality and kernelization: Lower bounds and upper bounds on kernel size. *SIAM Journal on Computing*, 37(4):1077–1106, 2007. 3, 23

[10] J. Chen, I. A. Kanj, and W. Jia. Vertex Cover: Further observations and further improvements. *Journal of Algorithms*, 41:280–301, 2001. 3

[11] F. K. H. A. Dehne, M. R. Fellows, H. Fernau, E. Prieto, and F. A. Rosamond. Nonblocker: Parameterized algorithmics for Minimum Dominating Set. In *Proc. of 32nd SOFSEM*, volume 3831 of *LNCS*, pages 237–245. Springer, 2006. 3

[12] E. D. Demaine and M. Hajiaghayi. The bidimensionality theory and its algorithmic applications. *The Computer Journal*, 51(3):292–302, 2008. 23

[13] F. Dorn, E. Penninkx, H. L. Bodlaender, and F. V. Fomin. Efficient exact algorithms on planar graphs: Exploiting sphere cut branch decompositions. In *Proc. 13th ESA*, volume 3669 of *LNCS*, pages 95–106. Springer, 2005. Long version to appear in *Algorithmica.* 23

[14] R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Springer-Verlag, Berlin, 1999. 2, 3

[15] V. Estivill-Castro, M. R. Fellows, M. A. Langston, and F. A. Rosamond. FPT is P-time extremal structure I. In *Proc. 1st ACiD*, pages 1–41, 2005. 6

[16] M. R. Fellows. The lost continent of polynomial time: preprocessing and kernelization. In *Proc. 3rd IWPEC*, volume 4169 of *LNCS*, pages 276–277. Springer, 2006. 3, 23

[17] M. R. Fellows, C. McCartin, F. A. Rosamond, and U. Stege. Coordinatized kernels and catalytic reductions: An improved FPT algorithm for Max Leaf Spanning Tree and other problems. In *Proc. 20th FSTTCS*, volume 1974 of *LNCS*, pages 240–251. Springer, 2000. 6

[18] J. Flum and M. Grohe. *Parameterized Complexity Theory.* Springer-Verlag, 2006. 2, 3

[19] S. Gaspers, S. Saurabh, and A. A. Stepanov. A moderately exponential time algorithm for full degree spanning tree. In *Proc. 5th TAMC*, volume 4978 of *LNCS*, pages 479–489. Springer, 2008. 2

[20] J. Guo and R. Niedermeier. Invitation to data reduction and problem kernelization. *ACM SIGACT News*, 38(1):31–45, 2007. 3, 23

[21] J. Guo and R. Niedermeier. Linear problem kernels for NP-hard problems on planar graphs. In *Proc. 34th ICALP*, volume 4596 of *LNCS*, pages 375–386. Springer, 2007. 3, 6, 7, 8, 9

[22] F. Hüffner, R. Niedermeier, and S. Wernicke. Techniques for practical fixed-parameter algorithms. *The Computer Journal*, 51(1):7–25, 2008. 23

[23] S. Khuller, R. Bhatia, and R. Pless. On local search and placement of meters in networks. *SIAM Journal on Computing*, 32(2):470–487, 2003. 2, 4

[24] T. Kloks. *Treewidth: Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science.* Springer, 1994. 20, 21

[25] M. Lewinter. Interpolation theorem for the number of degree-preserving vertices of spanning trees. *IEEE Transactions on Circuits Systems I: Fundamental Theory and Applications*, 34(2):205–205, 1987. 2

[26] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms.* Oxford University Press, 2006. 2, 3

[27] L. E. Ormsbee. Implicit network calibration. *Journal of Water Resources Planning Management*, 115:243–257, 1989. 2

[28] L. E. Ormsbee and D. J. Wood. Explicit pipe network calibration. *Journal of Water Resources Planning Management*, 112:116–182, 1986. 2

[29] P. D. Seymour and R. Thomas. Call routing and the ratcatcher. *Combinatorica*, 14(2):217–241, 1994. 22

[30] S. Wernicke. *Combinatorial Algorithms to Cope With the Complexity of Biological Networks*. PhD thesis, Institut für Informatik, Friedrich-Schiller-Universität Jena, Germany, 2006. 2

Algorithm: Maximal F-region decomposition.
Input: A graph $G = (V, E)$ and a maximum-size set F of full-degree vertices.
Output: A linear F-region decomposition $\mathcal{R}$.

```
01  $\mathcal{R} \leftarrow \emptyset$, $V_{used} \leftarrow \emptyset$
02  for each vertex $u \in V$ do
03      if $u \notin V_{used}$ and there exists a region $R(v, w)$ with $u \in V(R(v, w))$
          such that $\mathcal{R} \cup \{R\}$ is an $F$-region decomposition then
04          $S \leftarrow$ set of all regions $R(v, w)$ with $u \in V(R(v, w))$
                for which $\mathcal{R} \cup \{R\}$ is an $F$-region decomposition
05          $R_{new}(v, w) \leftarrow$ region from $S$ that is space-maximal
                       (that is, no region in $S$ is a proper superset of it)
06          $\mathcal{R} \leftarrow \mathcal{R} \cup \{R_{new}(v, w)\}$, $V_{used} \leftarrow V_{used} \cup V(R_{new}(v, w))$
07      fi
08  od
09  return $\mathcal{R}$
```

Figure 7: A greedy algorithm that constructs a linear F-region decomposition $\mathcal{R}$ for a plane graph G and a maximum-size set F of full-degree vertices.

Appendix: Proof of Lemma 4.3

Proof. We use the greedy algorithm that is shown in Figure 7 to constructively prove the existence of a linear F-region decomposition $\mathcal{R}$ as claimed. As a remark, this algorithm is quite similar to the one that Alber et al. [2] used for their proof of a linear-size kernel for Dominating Set in planar graphs; the only difference is that our regions are bounded by length-at-most-5 paths instead of length-at-most-3 paths.

Clearly, the given algorithm outputs an F-region decomposition for G. To see the maximality of this decomposition, observe that for every vertex u that is not in a region (that is, it has not been put into V_{used}) we check whether there is a region containing u that can be added to the existing region decomposition.

It remains to show the claimed upper bound on the number of regions in $\mathcal{R}$. For this purpose, consider a graph $G_{\mathcal{R}}$ that has the vertex set F and the edge set

$$E_{\mathcal{R}} = \bigcup_{R(v,w) \in \mathcal{R}} \{\{v, w\}\},$$

that is, it contains an edge $\{v, w\}$ for every region $R(v, w) \in \mathcal{R}$. It is clear that $G_{\mathcal{R}}$ is planar because the input graph G is planar and regions do not intersect. We now claim that $G_{\mathcal{R}}$ has the following *thinness property*: If there are four edges e_1, e_2, e_3, e_4 between two vertices v, w, then there exist two further vertices u_1 and u_2 in F, each one sitting in one of the four areas enclosed by e_1, e_2, e_3, e_4. Once we have established the thinness property, the claimed size-bound follows by a result of Alber et al. [2] who show that a plane graph $G =$

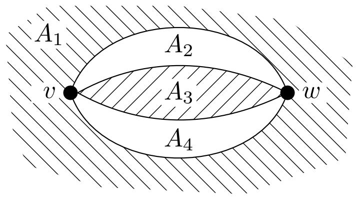

Figure 8: Illustration for the proof of Lemma 4.3: Four edges in the graph $G_{\mathcal{R}}$—each connecting two vertices v and w—divide the plane into four areas $A_1, \ldots, A_4$.

(V, E) with multiple edges satisfies $|E| \leq 3|V| - 6$ for $|V| \geq 3$ if each of the two areas enclosed by two edges e_1 and e_2 with the same endpoints contains at least one vertex; we just replace the two-edge enclosure by a four-edge enclosure of four areas.[11]

To show the thinness property of $G_{\mathcal{R}}$, suppose that there are four edges in $E_{\mathcal{R}}$ between two vertices v and w; these divide the plane into four areas $A_1, \ldots, A_4$. Assume for the purpose of contradiction that at most one of the areas $A_1, \ldots, A_4$ contains a vertex $u \in F$. Without loss of generality, let this be the area A_1 as in Figure 8. Consider the area A_3. By construction of $G_{\mathcal{R}}$, the two edges that enclose A_3 correspond to two regions $R_1(v, w)$ and $R_2(v, w)$ in the input graph G. But then G must have at least one vertex u' lying inside of A_3 because, otherwise, the regions $R_1(v, w)$ and $R_2(v, w)$ could be joined into a single, larger region, which contradicts the space-maximality that we demand in line *05* of the construction algorithm. The space-maximality also tells us that u' must have distance at least three to both v and w. This, however, implies that u' has distance at least three to *every* vertex in F because the areas A_2 and A_4 contain no vertex from F. This contradicts Lemma 4.1 and we have thus shown that $G_{\mathcal{R}}$ has the claimed thinness property. ☐

Appendix: Details of the dynamic programming algorithm in Section 5

To describe the algorithm, we define $2 \cdot \omega$ possible states $C(v) := (C_1(v), C_2(v))$ for each vertex v in a bag X_i where C_1 and C_2 are two functions $C_1 : X_i \to \{1, \ldots, \omega\}$ and $C_2 : X_i \to \{0, 1\}$. The first function $C_1(v)$ for $v \in X_i$ returns the number of the component containing v. (Since $|X_i| \leq \omega$ there can be at most ω components containing vertices from X_i.) The second function $C_2(v)$ encodes the information whether v is a full-degree vertex ($C_2(v) = 1$) or not ($C_2(v) = 0$).

The various possible combinations of vertex states that encode the components of $G[X_i]$ correspond to the rows of a table F_i for every node i of the tree

[11]For $|V| = 2$, it is easy to see that there can be at most one single region and hence the lemma holds.

decomposition. For a combination $C := (C(x_{i_1}), \ldots, C(x_{i_t}))$ of vertex states for the vertices in $X_i = \{x_{i_1}, x_{i_2}, \ldots, x_{i_t}\}$, $t \leq \omega$, the entry $F_i(C)$ stores the maximum number of full-degree vertices of a spanning tree for G_i under the condition that the vertices in X_i have the states as encoded in C.

In the following, we describe the details of the algorithm with respect to each node type. As the dynamic programming works bottom-up, we start with the leaf nodes.

Leaf nodes: Let i denote a leaf of T with $X_i = \{x_{i_1}, \ldots, x_{i_t}\}$. We call a combination of vertex states $C := (C(x_{i_1}), \ldots, C(x_{i_t}))$ *valid*, if the following conditions are satisfied:

- If there are two vertices $u, v \in X_i$ with $C_1(u) = C_1(v)$, then there has to be a path between u and v in $G[X_i]$ whose edges are all full-degree edges.
- For each edge $e = \{u, v\}$ in $G[X_i]$ with $C_1(u) \neq C_1(v)$, it holds that $C_2(u) = C_2(v) = 0$.
- For each $c \in \{1, \ldots, \omega\}$, there is no cycle in $G[\{v \in X_i \mid C_1(v) = c\}]$ consisting of full-degree edges.

The table entries F_i are then computed as follows.

$$F_i(C) = \begin{cases} |\{v \in X_i \mid C_2(v) = 1\}| & \text{if } C \text{ is valid} \\ -\infty & \text{otherwise.} \end{cases}$$

For the subsequent dynamic programming it is important that the following invariant holds: If $F_i(C) \neq -\infty$, then each component of $G[X_i]$ as encoded by C_1 of C has one of the properties P1 and P2 (see Section 5). Clearly, these two properties are already fulfilled by leaf nodes.

Forget nodes: For a forget node i with child node j, $X_i = \{x_{i_1}, x_{i_2}, \ldots, x_{i_t}\}$ and $X_j = \{x_{i_1}, x_{i_2}, \ldots, x_{i_t}, x\}$, the components in G_i are the same as in G_j. Thus, the table entries F_i can be easily computed:

$$F_i(C) = \max_{C(x)} \left\{ F_j \left((C(x_{i_1}), \ldots, C(x_{i_t}), C(x)) \right) \right\}.$$

Introduce nodes: In order to compute $F_i(C)$ for an introduce node i with node j as child, that is $X_i = \{x_{i_1}, x_{i_2}, \ldots, x_{i_t}, x\}$ and $X_j = \{x_{i_1}, x_{i_2}, \ldots, x_{i_t}\}$, we distinguish two cases, namely $C_2(x) = 0$ and $C_2(x) = 1$. For each of these cases, we define first a *compatibility set* $\mathrm{comp}(C)$ of C and $F_i(C)$ is then computed based on $\mathrm{comp}(C)$.

In the case that $C_2(x) = 0$, the compatibility set $\mathrm{comp}(C)$ of C contains the combinations of vertex states $C' = (C'(x_{i_1}), \ldots, C'(x_{i_t}))$ that satisfy the following conditions:

- $\forall u \in X_j : C_2(u) = C'_2(u)$,
- $\forall u \in X_j : C_1(u) = C'_1(u) \vee C_1(u) = C_1(x)$,
- $\neg \exists u, v \in X_i \cap N(x) : C_1(u) = C_1(v) \wedge C'_2(u) = C'_2(v) = 1$, and

- $\forall u \in X_j : C_1(u) = C_1(x) \Leftrightarrow (\exists v \in X_j \cap N(x) : C'_1(u) = C'_1(v) \wedge C'_2(v) = 1)$.

Then, $F_i(C)$ is computed as follows.

$$F_i(C) = \begin{cases} \max\{F_j(C') \mid C' \in \text{comp}(C)\} & \text{if comp}(C) \neq \emptyset \\ -\infty & \text{otherwise.} \end{cases}$$

For the case that $C_2(x) = 1$, the compatibility set comp(C) of C contains the combinations of vertex states $C' = (C'(x_{i_1}), \dots, C'(x_{i_t}))$ that satisfy the following conditions:

- $\forall u \in X_j : C_2(u) = C'_2(u)$,
- $\forall u \in X_j : C_1(u) = C'_1(u) \vee C_1(u) = C_1(x)$,
- $\neg\exists u, v \in X_i \cap N(x) : C_1(u) = C_1(v)$, and
- $\forall u \in X_j : C_1(u) = C_1(x) \Leftrightarrow (\exists v \in X_j \cap N(x) : C'_1(u) = C'_1(v))$.

Then, $F_i(C)$ is computed as follows.

$$F_i(C) = \begin{cases} 1 + \max\{F_j(C') \mid C' \in \text{comp}(C)\} & \text{if comp}(C) \neq \emptyset \\ -\infty & \text{otherwise.} \end{cases}$$

The correctness of these cases follows from the observation that several components of G_j encoded by combinations of states of vertices in X_j can be merged into one connected component. Therefore, we consider only the combinations of states of vertices in X_j which encode components that we can merge, that is, to merge these components does not result in a cycle consisting exclusively of edges incident to vertices v with $C_2(v) = 1$. These combinations are enclosed in comp(C).

Join nodes: For a join node i with nodes j and l as children, assume that $X_i = X_j = X_l = \{x_{i_1}, x_{i_2}, \dots, x_{i_t}\}$. In analogy to introduce nodes, we compute a *compatibility set* comp(C). By setting $S(X_i, C, c) := \{v \in X_i \mid C_1(v) = c\}$ for $c \in \{1, \dots, \omega\}$, the compatibility set comp(C) of C contains pairs of combinations of vertex states (C', C'') such that

- for all $v \in X_i$, $C_2(v) = C'_2(v) = C''_2(v)$, and,
- for each $c \in \{1, \dots, \omega\}$, there are two subsets Y and Z of $\{1, \dots, \omega\}$ such that
  - $S(X_i, C, c) = (\bigcup_{c' \in Y} S(X_i, C', c')) \cup (\bigcup_{c'' \in Z} S(X_i, C'', c''))$,
  - for every two c_1, c_2 from Y (or Z), there is a c from Z (or Y) such that $S(X_i, C', c_1) \cap S(X_i, C'', c) \neq \emptyset$ and $S(X_i, C', c_2) \cap S(X_i, C'', c) \neq \emptyset$, and
  - for every two vertices u, v with $u, v \in S(X_i, C', c') \cap S(X_i, C'', c'')$ for $c' \in Y$ and $c'' \in Z$, there is a path between u and v consisting solely of edges in $G[X_i]$ which are all incident to vertices w with $C_2(w) = 1$.

The computation of $F_i(C)$ is similar to the one at introduce nodes, namely,

$$F_i(C) := \max_{(C,C'')\in\mathrm{comp}(C)} \{F_j(C') + F_l(C'') - |\{v \in X_i \mid C_2(v) = 1\}|\}.$$