

Linear-time Algorithms for Eliminating Claws in Graphs^{*}

Flavia Bonomo-Braberman¹, Julliano R. Nascimento², Fabiano S. Oliveira³,
Uéverton S. Souza⁴, and Jayme L. Szwarcfiter^{3,5}

¹ Universidad de Buenos Aires. FCEyN. DC. / CONICET-UBA. ICC. Argentina.
fbonomo@dc.uba.ar

² INF, Universidade Federal de Goiás, GO, Brazil. julliano@inf.ufg.br

³ IME, Universidade do Estado do Rio de Janeiro, RJ, Brazil.

fabiano.oliveira@ime.uerj.br

⁴ IC, Universidade Federal Fluminense, RJ, Brazil. ueverton@ic.uff.br

⁵ IM, COPPE, and NCE, Universidade Federal do Rio de Janeiro, RJ, Brazil.
jayme@nce.ufrj.br

Abstract. Since many NP-complete graph problems have been shown polynomial-time solvable when restricted to claw-free graphs, we study the problem of determining the distance of a given graph to a claw-free graph, considering vertex elimination as measure. CLAW-FREE VERTEX DELETION (CFVD) consists of determining the minimum number of vertices to be removed from a graph such that the resulting graph is claw-free. Although CFVD is NP-complete in general and recognizing claw-free graphs is still a challenge, where the current best algorithm for a graph G has the same running time of the best algorithm for matrix multiplication, we present linear-time algorithms for CFVD on weighted block graphs and weighted graphs with bounded treewidth. Furthermore, we show that this problem can be solved in linear time by a simpler algorithm on forests, and we determine the exact values for full k -ary trees. On the other hand, we show that CLAW-FREE VERTEX DELETION is NP-complete even when the input graph is a split graph. We also show that the problem is hard to approximate within any constant factor better than 2, assuming the Unique Games Conjecture.

Keywords: Claw-free graph · Vertex deletion · Weighted vertex deletion.

1 Introduction

In 1968, Beineke [1] introduced claw-free graphs as a generalization of line graphs. Besides that generalization, the interest in studying the class of claw-free graphs also emerged due to the results showing that some NP-complete problems are polynomial time solvable in that class of graphs. For example, the maximum

^{*} The authors would like to thank CAPES, FAPERJ, CNPq, ANPCyT, and UBACyT for the partial support.

independent set problem is polynomially solvable for claw-free graphs, even on its weighted version [11].

A considerable amount of literature has been published on claw-free graphs. For instance, Chudnovsky and Seymour provide a series of seven papers describing a general structure theorem for that class of graphs, which are sketched in [5]. Some results on domination, Hamiltonian properties, and matchings are found in [16], [19], and [29], respectively. In the context of parameterized complexity, Cygan et al. [10] show that finding a minimum dominating set in a claw-free graph is fixed-parameter tractable. For more on claw-free graphs, we refer to a survey by Faudree, Flandrin and Ryjáček [12] and references therein.

The aim of our work is to obtain a claw-free graph by a minimum number of vertex deletions. Given a graph G and a property Π , Lewis and Yannakakis [25] define a family of vertex deletion problems (Π -VERTEX DELETION) whose goal is finding the minimum number of vertices which must be deleted from G so that the resulting graph satisfies Π . Throughout this paper we consider the property Π as belonging to the class of claw-free graphs. For a set $S \subseteq V(G)$, we say that S is a *claw-deletion set* of G if $G \setminus S$ is a claw-free graph.

We say that a class of graphs \mathcal{C} is *hereditary* if, for every graph $G \in \mathcal{C}$, every induced subgraph of G belongs to \mathcal{C} . If either the number of graphs in \mathcal{C} or the number of graphs not in \mathcal{C} is finite, then \mathcal{C} is *trivial*. A celebrated result of Lewis and Yannakakis [25] shows that for any hereditary and nontrivial graph class \mathcal{C} , Π -VERTEX DELETION is NP-hard for Π being the property of belonging to \mathcal{C} . Therefore, Π -VERTEX DELETION is NP-hard when Π is the property of belonging to the class \mathcal{C} of claw-free graphs. Cao et al. [4] obtain several results when Π is the property of belonging to some particular subclasses of chordal graphs. They show that transforming a split graph into a unit interval graph with the minimum number of vertex deletions can be solved in polynomial time. In contrast, they show that deciding whether a split graph can be transformed into an interval graph with at most k vertex deletions is NP-complete. Motivated by the works of Lewis and Yannakakis [25] and Cao et al. [4], since claw-free graphs is a natural superclass of unit interval graphs, we study vertex deletion problems associated with eliminating claws. The problems are formally stated below.

Problem 1. CLAW-FREE VERTEX DELETION (CFVD)

Instance: A graph G , and $k \in \mathbb{Z}^+$.

Question: Does there exist a claw-deletion set S of G with $|S| \leq k$?

Problem 2. WEIGHTED CLAW-FREE VERTEX DELETION (WCFVD)

Instance: A graph G , a weight function $w : V(G) \rightarrow \mathbb{Z}^+$, and $k \in \mathbb{Z}^+$.

Question: Does there exist a claw-deletion set S of G with $\sum_{v \in S} w(v) \leq k$?

By Roberts' characterization of unit interval graphs [27], CLAW-FREE VERTEX DELETION on interval graphs is equivalent to the vertex deletion problem where the input is restricted to the class of interval graphs and the target class is the class of unit interval graphs, a long standing open problem (see e.g. [4]). Then, the results by Cao et al. [4] imply that CLAW-FREE VERTEX DELETION is polynomial-time solvable when the input graph is in the class of interval \cap split

graphs. Moreover, their algorithm could be also generalized to the weighted version. In this paper, we show that CLAW-FREE VERTEX DELETION is NP-complete when the input graph is in the class of split graphs.

The results by Lund and Yannakakis [26] imply that CLAW-FREE VERTEX DELETION is APX-hard and admits a 4-approximating greedy algorithm. Even for the weighted case, a pricing primal-dual 4-approximating algorithm is known for the more general problem of 4-HITTING SET [17]. The CFVD problem is NP-complete on bipartite graphs [33], and a 3-approximating algorithm is presented by Kumar et al. in [23] for weighted bipartite graphs. We prove that the unweighted problem is hard to approximate within any constant factor better than 2, assuming the Unique Games Conjecture, even for split graphs.

Regarding to parameterized complexity, CLAW-FREE VERTEX DELETION is a particular case of H -FREE VERTEX DELETION, which can be solved in $|V(H)|^k n^{\mathcal{O}(1)}$ time using the bounded search tree technique. In addition, it can also be observed that CFVD is a particular case of 4-HITTING SET thus, by Sunflower lemma, it admits a kernel of size $\mathcal{O}(k^4)$, and the complexity can be slightly improved [13]. With respect to width parameterizations, it is well-known that every optimization problem expressible in LinEMSOL_1 can be solved in linear time on graphs with bounded cliquewidth [6]. Since claws are induced subgraphs with constant size, it is easy to see that finding the minimum weighted S such that $G \setminus S$ is claw-free is LinEMSOL_1 -expressible. Therefore, WCFVD can be solved in linear time on graphs with bounded cliquewidth, which includes trees, block graphs and bounded treewidth graphs. However, the linear-time algorithms based on the MSOL model-checking framework [7] typically do not provide useful algorithms in practice since the dependence on the cliquewidth involves huge multiplicative constants, even when the clique-width is bounded by two (see [14]). In this work, we provide explicit discrete algorithms to effectively solve WCFVD in linear time in practice on block graphs and bounded treewidth graphs. Even though forests are particular cases of bounded treewidth graphs and block graphs, we describe a specialized simpler linear-time algorithm for CFVD on forests. This allows us to determine the exact values of CFVD for a full k -ary tree T with n vertices. If $k = 2$, we show that a minimum claw-deletion set of T has cardinality $(n + 1 - 2^{(\log_2(n+1) \bmod 3)})/7$, and $(nk - n + 1 - k^{(\log_k(nk-n+1) \bmod 2)})/(k^2 - 1)$, otherwise.

This paper is organized as follows. Section 2 is dedicated to show the hardness and inapproximability results. Sections 3, 4, and 5 present results on forests, block graphs, and bounded treewidth graphs, respectively. Due to space constraints, proofs of statements marked with ‘♣’ are deferred to the appendix, as well as some additional results and well known definitions.

Preliminaries. We consider simple and undirected graphs, and we use standard terminology and notation.

Let T be a tree rooted at $r \in V(T)$ and $v \in V(T)$. We denote by T_v the subtree of T rooted at v , and by $C_T(v)$ the set of children of v in T . For $v \neq r$, denote by $p_T(v)$ the parent of v in T , and by T_v^+ the subgraph of T induced by

$V(T_v) \cup \{p_T(v)\}$. Let $T_r^+ = T$ and $p_T(r) = \emptyset$. When T is clear from the context, we simply write $p(v)$ and $C(v)$.

The *block-cutpoint-graph* of a graph G is the bipartite graph whose vertex set consists of the set of cutpoints of G and the set of blocks of G . A cutpoint is adjacent to a block whenever the cutpoint belongs to the block in G . The block-cutpoint-graph of a connected graph is a tree and can be computed in $\mathcal{O}(|V(G)| + |E(G)|)$ time [30].

Let G and H be two graphs. We say that G is *H-free* if G does not contain a graph isomorphic to H as an induced subgraph. A *claw* is the complete bipartite graph $K_{1,3}$. The class of *linear forests* is equivalent to that of claw-free forests. A vertex v in a claw C is a *center* if $d_C(v) = 3$. The cardinality $\text{cdn}(G)$ of a minimum claw-deletion set in G is the *claw-deletion number* of G . For our proofs, it is enough to consider connected graphs, since a minimum (weight) claw-deletion set of a graph is the union of minimum (weight) claw-deletion sets of its connected components. Williams et al. [32] show that induced claws in an n -vertex graph G can be detected in $\mathcal{O}(n^\omega)$ time, where ω is the matrix multiplication exponent. As far as we know, the best upper bound is $\omega < 2.3728639$ [24].

2 Complexity and Approximability Results

The result of Lewis and Yannakakis [25] implies that CLAW-FREE VERTEX DELETION is NP-complete. In this section, we show that the same problem is NP-complete even when restricted to split graphs, a well known subclass of chordal graphs. Before the proof, let us recall that the VERTEX COVER (VC) problem consists of, given a graph G and a positive integer k as input, deciding whether there exists $X \subseteq V(G)$, with $|X| \leq k$, such that every edge of G is incident to a vertex in X .

Theorem 1. CLAW-FREE VERTEX DELETION *on split graphs is NP-complete.*

Proof. CLAW-FREE VERTEX DELETION is clearly in NP since claw-free graphs can be recognized in polynomial time [32]. To show NP-hardness, we employ a reduction from VERTEX COVER on general graphs [15].

Let (G, k) be an instance of vertex cover, where $V(G) = \{v_1, \dots, v_n\}$, and $E(G) = \{e_1, \dots, e_m\}$. Construct a split graph $G' = (C \cup I, E')$ as follows. The independent set is $I = \{v'_1, \dots, v'_n\}$. The clique C is partitioned into sets C_i , $1 \leq i \leq m + 1$, each on $2n$ vertices. Given an enumeration e_1, \dots, e_m of $E(G)$, if $e_i = v_j v_\ell$, make v'_j and v'_ℓ adjacent to every vertex in C_i .

We prove that G has a vertex cover of size at most k if and only if G' has a claw-deletion set of size at most k . We present Claim 2 first.

Claim 2 *Every claw in G' contains exactly two vertices from I .*

Proof. Let C' be a claw in G' . Since $C' \cap C$ is a clique, $|C' \cap C| \leq 2$, thus $|C' \cap I| \geq 2$ and the center of the claw must be in C . On the other hand, by construction, $d_I(u) = 2$ for every $u \in \bigcup_{i=1}^m C_i$. This implies $|C' \cap I| \leq 2$. \diamond

Suppose that X is a vertex cover of size at most k in G . Then, every edge of G is incident to a vertex in X . Let $e_i \in E(G)$ and $X' = \{v' : v \in X\}$. By construction, every vertex in C_i is adjacent to a vertex in X' , therefore $|N_{G' \setminus X'}(C_i) \cap I| \leq 1$. It follows by Claim 2 that $G' \setminus X'$ is claw-free.

Now, suppose that S' is a claw-deletion set of G' of size at most k . Recall that $|C_i| = 2n$, for every $1 \leq i \leq m+1$. Since $|S'| \leq k$, it follows that there exist $w_i \in C_i \setminus S'$, for every $1 \leq i \leq m+1$. Let $1 \leq i \leq m$ and $N_I(w_i) = \{u', v'\}$. Note that $\{u', v', w_i, w_{m+1}\}$ induces a claw in G' . Since S' is a claw-deletion set of G' , we have that $S' \cap \{u', v'\} \neq \emptyset$. Let $S = \{v : v' \in S' \cap I\}$. By construction, every $uv \in E(G)$ is incident to a vertex in S , thus S is a vertex cover of G . \square

Theorem 3 provides a lower bound for the approximation factor of CFVD. For terminology not defined here, we refer to Crescenzi [8].

Theorem 3. CLAW-FREE VERTEX DELETION *cannot be approximated with $2 - \varepsilon$ ratio for any $\varepsilon > 0$, even on split graphs, unless Unique Games Conjecture fails.*

Proof. The *Unique Games Conjecture* was introduced by Khot [20] in 2002. Some hardness results have been proved assuming that conjecture, for instance, see [21]. Given that VERTEX COVER is hard to approximate to within $2 - \varepsilon$ ratio for any $\varepsilon > 0$ assuming the Unique Games Conjecture [20], we perform an approximation-preserving reduction from VERTEX COVER. Let G be an instance of VERTEX COVER. Let $f(G) = G'$ where G' is the instance of CLAW-FREE VERTEX DELETION constructed from G according to the reduction of Theorem 1. From Theorem 1 we know that G has a vertex cover of size at most k if and only if G' has a claw-deletion set of size at most k . Recall that $k \leq n = |V(G)|$. Then, for every instance G of VERTEX COVER it holds that $\text{opt}_{\text{CFVD}}(G') = \text{opt}_{\text{VC}}(G)$. Now, suppose that S' is a $(2 - \varepsilon)$ -approximate solution of G' for CFVD. Recall that $|C_i| = 2n$, for every $1 \leq i \leq m+1$. Since $\text{opt}_{\text{CFVD}}(G') = \text{opt}_{\text{VC}}(G) \leq n$, it follows that $|S'| < 2n$, thus, there exists $x \in C_{m+1} \setminus S'$, and $w \in C_i \setminus S'$, for every $1 \leq i \leq m$. Again, let $N_I(w) = \{u', v'\}$. Note that $\{u', v', w, x\}$ induces a claw in G' . Since S' is a claw-deletion set of G' , we have that $S' \cap \{u', v'\} \neq \emptyset$. Let $S = \{v : v' \in S' \cap I\}$. By construction, every $uv \in E(G)$ is incident to a vertex in S , and therefore S is a vertex cover of G . Since $|S| \leq |S'|$ and S' is a $(2 - \varepsilon)$ -approximate solution of G' , then $|S| \leq |S'| \leq (2 - \varepsilon) \cdot \text{opt}_{\text{CFVD}}(G') = (2 - \varepsilon) \cdot \text{opt}_{\text{VC}}(G)$. Therefore, if CFVD admits a $(2 - \varepsilon)$ -approximate algorithm then VERTEX COVER also admits a $(2 - \varepsilon)$ -approximate algorithm, which implies that the Unique Games Conjecture fails [20]. \square

3 Forests

We propose Algorithm 1 to compute a minimum claw-deletion set S of a rooted tree T . The correctness of such algorithm follows in Theorem 8.

Algorithm 1: CLAW-DELETION-SET(T, v, p)

Input: A rooted tree T , a vertex v of T , and the parent p of v in T .
Output: A minimum claw-deletion set S of T_v^+ , such that: if $\text{cdn}(T_v^+) = 1 + \text{cdn}(T_v)$ then $p \in S$; if $\text{cdn}(T_v^+) = \text{cdn}(T_v)$ and $\text{cdn}(T_v) = 1 + \text{cdn}(T_v \setminus \{v\})$ then $v \in S$.

```

1 if  $C(v) = \emptyset$  then
2   | return  $\emptyset$ 
3 else
4   |  $S := \emptyset$ 
5   | foreach  $u \in C(v)$  do
6     |  $S := S \cup \text{CLAW-DELETION-SET}(T, u, v)$ 
7   |  $c := |C(v) \setminus S|$ 
8   | if  $c \geq 3$  then
9     |  $S := S \cup \{v\}$ 
10  | else if  $c = 2$  and  $p \neq \emptyset$  and  $v \notin S$  then
11    |  $S := S \cup \{p\}$ 
12  | return  $S$ 

```

Theorem 4. (\clubsuit) *Algorithm 1 is correct. Thus, given a forest F , and a positive integer k , the problem of deciding whether F can be transformed into a linear forest with at most k vertex deletions can be solved in linear time.*

Moreover, based on the algorithm, we have the following results.

Corollary 1. (\clubsuit) *Let T be a full binary tree with n vertices, and $t = \log_2(n + 1) \bmod 3$. Then $\text{cdn}(T) = (n + 1 - 2^t)/7$.*

Corollary 2. (\clubsuit) *Let T be a full k -ary tree with n vertices, for $k \geq 3$, and $t = \log_k(nk - n + 1) \bmod 2$. Then $\text{cdn}(T) = (nk - n + 1 - k^t)/(k^2 - 1)$.*

4 Block Graphs

We describe a dynamic programming algorithm to compute the minimum weight of a claw-deletion set in a weighted connected block graph G . The algorithm to be presented can be easily modified to compute also a set realizing the minimum.

If the block graph G has no cutpoint, the problem is trivial as G is already claw-free. Otherwise, let T be the block-cutpoint-tree of the block graph G . Consider T rooted at some cutpoint r of G , and let $v \in V(T)$. Let G_v the subgraph of G induced by the blocks in T_v . For $v \neq r$, let G_v^+ be the subgraph of G induced by the blocks in T_v^+ . If b is a block, let $G_b^- = G_b \setminus \{p_T(b)\}$ (notice that $p_T(b)$ is a cutpoint of G , and it is always defined because r is not a block), and let $s(b)$ be the sum of weights of the vertices of b that are not cutpoints of G ($s(b) = 0$ if there is no such vertex).

We consider three functions to be computed for a vertex v of T that is a cutpoint of G :

- $f_1(v)$: the minimum weight of a claw-deletion set of G_v containing v .
- $f_2(v)$: the minimum weight of a claw-deletion set of G_v not containing v .
- For $v \neq r$, $f_3(v)$: the minimum weight of a claw-deletion set of G_v^+ containing neither v nor all the vertices of $p_T(v) \setminus \{v\}$ (notice that $p_T(v)$ is a block).

The parameter that solves the whole problem is $f(r) = \min\{f_1(r), f_2(r)\}$.

We define also three functions to be computed for a vertex b of T that is a block of G :

- $f_1(b)$: the minimum weight of a claw-deletion set of G_b^- containing $b \setminus \{p_T(b)\}$.
- $f_2(b)$: the minimum weight of a claw-deletion set of G_b^- .
- $f_3(b)$: the minimum weight of a claw-deletion set of G_b not containing $p_T(b)$.

We compute the functions in a bottom-up order as follows, where v (resp. b) denotes a vertex of T that is a cutpoint (resp. block) of G . Notice that the leaves of T are blocks of G .

If $C(b) = \emptyset$, then $f_1(b) = s(b)$, $f_2(b) = 0$, and $f_3(b) = 0$. Otherwise,

- $f_1(v) = w(v) + \sum_{b \in C(v)} f_2(b)$; $f_1(b) = s(b) + \sum_{v \in C(b)} f_1(v)$;
- if $|C(v)| \leq 2$, then $f_2(v) = \sum_{b \in C(v)} f_3(b)$; if $|C(v)| \geq 3$, then $f_2(v) = \min_{b_1, b_2 \in C(v)} (\sum_{b \in \{b_1, b_2\}} f_3(b) + \sum_{b \in C(v) \setminus \{b_1, b_2\}} f_1(b))$;
- $f_2(b) = \min\{\sum_{v \in C(v)} \min\{f_1(v), f_3(v)\}, \min_{v_1 \in C(v)} (s(b) + f_2(v_1) + \sum_{v \in C(v) \setminus \{v_1\}} f_1(v))\}$;
- $f_3(b) = \sum_{v \in C(b)} \min\{f_1(v), f_3(v)\}$;
- if $C(v) = \{b\}$, then $f_3(v) = f_3(b)$;
- if $|C(v)| \geq 2$, then $f_3(v) = \min_{b_1 \in C(v)} (f_3(b_1) + \sum_{b \in C(v) \setminus \{b_1\}} f_1(b))$.

The explanation of the correctness of these formulas follows in Theorem 5.

Theorem 5. (\clubsuit) *Let G be a weighted connected block graph which is not complete. Let T be the block-cutpoint-tree of G , rooted at a cutpoint r . The previous function $f(r)$ computes correctly the minimum weight of a claw-deletion set of G .*

We obtain this result as a corollary.

Corollary 3. (\clubsuit) *Let G be a weighted block graph with n vertices and m edges. The minimum weight of a claw-deletion set of G can be determined in $\mathcal{O}(n+m)$ time.*

5 Graphs of Bounded Treewidth

Next, we present an algorithm able of solving WEIGHTED CLAW-FREE VERTEX DELETION in linear time on graphs with bounded treewidth, which also implies that we can recognize claw-free graphs in linear time when the input graph has treewidth bounded by a constant. For definitions of tree decompositions and treewidth, we refer the reader to [9,22,28].

Graphs of treewidth at most k are called *partial k -trees*. Some graph classes with bounded treewidth include: forests (treewidth 1); pseudoforests, cacti, outerplanar graphs, and series-parallel graphs (treewidth at most 2); Halin graphs and Apollonian networks (treewidth at most 3) [2]. In addition, control flow graphs arising in the compilation of structured programs also have bounded treewidth (at most 6) [31].

Based on the following results we can assume that we are given a nice tree decomposition of the input graph G .

Theorem 6. [3] *There exists an algorithm that, given a n -vertex graph G and an integer k , runs in time $2^{\mathcal{O}(k)} \cdot n$ and either outputs that the treewidth of G is larger than k , or constructs a tree decomposition of G of width at most $5k + 4$.*

Lemma 1. [22] *Given a tree decomposition $(T, \{X_t\}_{t \in V(T)})$ of G of width at most k , one can compute in time $\mathcal{O}(k^2 \cdot \max\{|V(T)|, |V(G)|\})$ a nice tree decomposition of G of width at most k that has at most $\mathcal{O}(k \cdot |V(G)|)$ nodes.*

Now we are ready to use a nice tree decomposition in order to obtain a linear-time algorithm for WEIGHTED CLAW-FREE VERTEX DELETION on graphs with bounded treewidth.

Theorem 7. WEIGHTED CLAW-FREE VERTEX DELETION *can be solved in linear time on graphs with bounded treewidth. More precisely, there is a $2^{\mathcal{O}(k^2)} \cdot n$ -time algorithm to solve WEIGHTED CLAW-FREE VERTEX DELETION on n -vertex graphs G with treewidth at most k .*

Proof. Let G be a weighted n -vertex graph with $tw(G) \leq k$. Given a nice tree decomposition $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ of G , we describe a procedure that computes the minimum weight of a claw-deletion set of G ($\text{cdn}_w(G)$) using dynamic programming. For a node t of T , let $V_t = \bigcup_{t' \in T_t} X_{t'}$. First, we will describe what should be stored in order to index the table. Given a claw-deletion set \hat{S} of G , for any bag X_t there is a partition of X_t into S_t, A_t, B_t and C_t where

- S_t is the set of vertices of X_t that are going to be removed ($S_t = \hat{S} \cap X_t$);
- $A_t = \{v \in X_t \setminus \hat{S} : |N_{V_t \setminus X_t}(v) \setminus \hat{S}| = 0\}$ is the set of non-removed vertices of X_t that are going to have no neighbor in $V_t \setminus X_t$ after the removal of \hat{S} ;
- $B_t = \{v \in X_t \setminus \hat{S} : N_{V_t \setminus X_t}(v) \setminus \hat{S} \text{ induces a non-empty clique}\}$ is the set of non-removed vertices of X_t that, after the removal of \hat{S} , are going to have neighbors in $V_t \setminus X_t$, but no pair of non-adjacent neighbors;
- $C_t = \{v \in X_t \setminus \hat{S} : \text{there exist } u, u' \in N_{V_t \setminus X_t}(v) \setminus \hat{S} \text{ with } uu' \notin E(G)\}$ is the set of non-removed vertices of X_t that, after the removal of \hat{S} , are going to have a pair of non-adjacent neighbors in $V_t \setminus X_t$.

In addition, the claw-deletion set \hat{S} also provides the set $Z_t = \{(x, y) \in (X_t \setminus \hat{S}) \times (X_t \setminus \hat{S}) : \exists w \in V_t \setminus (X_t \cup \hat{S}) \text{ with } xy, wy \in E(G) \text{ and } wx \notin E(G)\}$ which consists of ordered pairs of vertices x, y of X_t that, after the removal of \hat{S} , are going to induce a $P_3 = x, y, w$ with some $w \in V_t \setminus X_t$.

Therefore, the recurrence relation of our dynamic programming has the signature $\text{cdn}_w[t, S, A, B, C, Z]$, representing the minimum weight of a vertex set whose removal from $G[V_t]$ leaves a claw-free graph, such that S, A, B, C form a partition of X_t as previously described, and Z is as previously described too. The generated table has size $2^{\mathcal{O}(k^2)} \cdot n$.

Function cdn_w is computed for every node $t \in V(T)$, for every partition $S \cup A \cup B \cup C$ of X_t , and for every $Z \subseteq X_t \times X_t$. The algorithm performs the computations in a bottom-up manner. Let T rooted at $r \in V(T)$. Notice that $V_r = V(G)$, then $\text{cdn}_w[r, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset]$ is the weight of a minimum weight claw-deletion set of $G_r = G$, which solves the whole problem.

We present additional terminology. Let t be a node in T with children t' and t'' , and $X \subseteq X_t$. To specify the sets S, A, B, C and Z on t' and t'' , we employ the notation S', A', B', C', Z' and S'', A'', B'', C'', Z'' , respectively.

Now, we describe the recurrence formulas for the function cdn_w defined, based on the types of nodes in T .

– **Leaf node.** If t is a leaf node in T , then $\text{cdn}_w[t, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset] = 0$. (1)

– **Introduce node.** Let t be an introduce node with child t' such that $X_t = X_{t'} \cup \{v\}$ for some vertex $v \notin X_{t'}$. Let $S \cup A \cup B \cup C$ be a partition of X_t , and $Z \subseteq X_t \times X_t$. The recurrence is given by the following formulas.

• If $v \in S$, then

$$\text{cdn}_w[t, S, A, B, C, Z] = \text{cdn}_w[t', S \setminus \{v\}, A, B, C, Z] + w(v). \quad (2.1)$$

• If $v \in A$, then $\text{cdn}_w[t, S, A, B, C, Z] = \text{cdn}_w[t', S, A \setminus \{v\}, B, C, Z']$, (2.2) if $N_{X_t \setminus S}(v)$ does not induce a \overline{K}_3 , for every $(x, y) \in Z, vx \in E(G)$ or $vy \notin E(G)$, $N_{X_t}(v) \cap C = \emptyset$, there is Z' such that $Z = Z' \cup \{(v, y) : y \in B \cup C \text{ and } vy \in E(G)\}$.

Otherwise, $\text{cdn}_w[t, S, A, B, C, Z] = \infty$.

• If $v \in B \cup C$, then $\text{cdn}_w[t, S, A, B, C, Z] = \infty$. (2.3)

– **Forget node.** Consider t a forget node with child t' such that $X_t = X_{t'} \setminus \{v\}$ for some vertex $v \in X_{t'}$. Let $S \cup A \cup B \cup C$ be a partition of X_t , and $Z \subseteq X_t \times X_t$.

$$\text{If } N_A(v) \neq \emptyset, \text{ then } \text{cdn}_w[t, S, A, B, C, Z] = \text{cdn}_w[t', S \cup \{v\}, A, B, C, Z]. \quad (3.1)$$

Otherwise, $\text{cdn}_w[t, S, A, B, C, Z] =$

$$\min \{ \text{cdn}_w[t', S \cup \{v\}, A, B, C, Z], \text{cdn}_w[t', S, A', B', C', Z'] \}, \quad (3.2)$$

among every (S, A', B', C', Z') such that:

$$Z = (Z' \setminus \{(x, y) : x = v \text{ or } y = v\}) \cup \{(x, y) \in X_t \times X_t : xy, vy \in E(G) \text{ and } vx \notin E(G)\},$$

$$A = A' \setminus N_G[v], B = ((B' \setminus \{b \in B' : (v, b) \in Z'\}) \cup (A' \cap N_G(v))) \setminus \{v\},$$

$$C = (C' \cup \{b \in B' : (v, b) \in Z'\}) \setminus \{v\}.$$

– **Join node.** Consider t a join node with children t', t'' such that $X_t = X_{t'} = X_{t''}$. Let $S \cup A \cup B \cup C$ be a partition of X_t , and $Z \subseteq X_t \times X_t$. The recursive formula is given by

$$\text{cdn}_w[t, S, A, B, C, Z] =$$

$$\min \{ \text{cdn}_w[t', S', A', B', C', Z'] + \text{cdn}_w[t'', S'', A'', B'', C'', Z''] \} - w(S), \quad (4)$$

among every (S', A', B', C', Z') and $(S'', A'', B'', C'', Z'')$ such that: $S =$

$$S' = S''; A = A' \cap A''; B = (A' \cap B'') \cup (A'' \cap B'); C = C' \cup C'' \cup (B' \cap B''); \\ Z = Z' \cup Z''.$$

We explain the correctness of these formulas. The base case is when t is a leaf node. In this case $X_t = \emptyset$, then all the sets S, A, B, C, Z are empty. The set $X_t = \emptyset$ also implies that $G[V_t]$ is the empty graph, which is claw-free. Hence, $\text{cdn}_w(G[V_t]) = 0$ and Formula (1) holds.

Let t be an introduce node with child t' , and v the vertex introduced at t . First, suppose that $v \in S$. We assume by inductive hypothesis that $G[V_{t'} \setminus \hat{S}]$ is claw-free. Since $v \in S \subseteq \hat{S}$, we obtain that $G[V_t \setminus (\hat{S} \cup \{v\})]$ is claw-free. Then, the weight of a minimum weight claw-deletion set of $G[V_t]$ is increased by $w(v)$ from the one of $G[V_{t'}]$, stored at $\text{cdn}_w[t', S', A', B', C', Z']$. Since $v \in S$, then $v \notin S'$ and the sets A', B', C', Z' in node t' are the same A, B, C, Z of t . Consequently Formula (2.1) holds.

Now, suppose that $v \in A \cup B \cup C$. By definition of tree decomposition, $v \notin N_{V_t \setminus X_t}(X_t)$. Then, if $v \in B \cup C$, the partition $S \cup A \cup B \cup C$ is not defined as required, and this justifies Formula (2.3). Thus, let $v \in A$. We have three cases in which $G[V_t \setminus \hat{S}]$ contains an induced claw: (i) $N_{X_t}(v)$ induces a \overline{K}_3 , or (ii) there exists $(x, y) \in Z$, such that $vx \notin E(G)$ and $vy \in E(G)$, or (iii) there exists $c \in C$ such that $cv \in E(G)$. A set Z according to definition of cdn_w is obtained by Z' together with the pairs (x, y) such that $x = v$, $xy \in E(G)$ and y has at least one neighbor in $V_t \setminus (X_t \cup \hat{S})$. (Note that $v = y$ is never achieved, since v is an introduce node and $v \notin N_{V_t \setminus X_t}(X_t)$). Then, $Z = Z' \cup \{(v, y) : y \in B \cup C \text{ and } vy \in E(G)\}$. Hence, Formula (2.2) is justified by the negation of each of cases (i), (ii), (iii).

Next, let t be a forget node with child t' . Let v be the vertex forgotten at t . We consider $N_A(v) \neq \emptyset$ or not. Notice that if $N_G(v) \cap A \neq \emptyset$ and $v \notin \hat{S}$, then we have a contradiction to the definition of A , because some $a \in A$ is going to have a neighbor in $V_t \setminus (X_t \cup \hat{S})$. Therefore, if $N_A(v) \neq \emptyset$, v indeed must belong to \hat{S} , then Formula (3.1) holds.

Otherwise, consider that $N_A(v) = \emptyset$. In this case, either $v \in \hat{S}$ or $v \notin \hat{S}$. Then, we choose the minimum between these two possibilities. If $v \in \hat{S}$ we obtain the value stored at $\text{cdn}_w[t', S \cup \{v\}, A, B, C, Z]$. Otherwise, let $v \notin \hat{S}$. It follows that, for some $a \in A$, if $va \in E(G)$, then a must now belong to B . Consequently, A must be $A' \setminus N_G[v]$. Let $\mathcal{B} = \{b \in B' : (v, b) \in Z'\}$. Since $v \notin \hat{S}$, for every $x \in \mathcal{B}$, x must belong to C . Thus, the set B is given by $B' \setminus \mathcal{B}$ together with the vertices from A' that now belong to B . Recall that $v \notin X_t$, then $v \notin B$. Hence, $B = ((B' \setminus \mathcal{B}) \cup (A' \cap N_G(v))) \setminus \{v\}$. Finally, $C = (C' \cup \mathcal{B}) \setminus \{v\}$. Hence, Formula (3.2) holds.

To conclude, let t be a join node with children t' and t'' . Note that the graphs induced by $V_{t'}$ and by $V_{t''}$ can be distinct. Then, we must sum the values of cdn_w in t' and in t'' to obtain cdn_w in t , and choose the minimum of all of these possible sums. Finally, we subtract $w(S)$ from the previous result, since $w(S)$ is counted twice.

By definition of join node, $X_t = X_{t'} = X_{t''}$, then $S = S' = S''$. Let $x \in X_t$. We have that $x \in A$ if and only if $|N_{V_{t'} \setminus X_{t'}}(x) \setminus \hat{S}| = |N_{V_{t''} \setminus X_{t''}}(x) \setminus \hat{S}| = 0$. Then, $A = A' \cap A''$.

Notice that $x \in B$ if and only if $(|N_{V_{t'} \setminus X_{t'}}(v) \setminus \hat{S}| = 0 \text{ and } |N_{V_{t''} \setminus X_{t''}}(v) \setminus \hat{S}| > 1)$ or $(|N_{V_{t'} \setminus X_{t'}}(v) \setminus \hat{S}| = 0 \text{ and } |N_{V_{t'} \setminus X_{t'}}(v) \setminus \hat{S}| > 1)$. Consequently $x \in B$ if and only if $x \in (A' \cap B'') \cup (A'' \cap B')$. This implies that $B = (A' \cap B'') \cup (A'' \cap B')$.

Now, $x \in C$ if and only if $x \in C'$ or $x \in C''$ or $(x \in B'$ and $x \in B'')$. (Note that by the definition of tree decomposition, the forgotten nodes in $G_{t'}$ and $G_{t''}$ are distinct and therefore the condition $x \in B'$ and $x \in B''$ is safe). Consequently, $C = C' \cup C'' \cup (B' \cap B'')$.

Finally, let $x, y \in X_t$. By definition of Z' , if $(x, y) \in Z'$, then there exists $w \in V_{t'} \setminus (X_{t'} \cup \hat{S})$ with $xy, wy \in E(G)$ and $wx \notin E(G)$. This implies that $w \in V_t \setminus (X_t \cup \hat{S})$ and $xy, wy \in E(G)$ and $wx \notin E(G)$. Hence, $(x, y) \in Z$. By a similar argument, we conclude that if $(x, y) \in Z''$, then $(x, y) \in Z$. This gives $Z = Z' \cup Z''$, and completes Formula (4).

Since the time to compute each entry of the table is upper bounded by $2^{\mathcal{O}(k^2)}$ (see Appendix) and the table has size $2^{\mathcal{O}(k^2)} \cdot n$, the algorithm can be performed in $2^{\mathcal{O}(k^2)} \cdot n$ time. This implies linear-time solvability for graphs with bounded treewidth. \square

References

1. Beineke, L.: Derived graphs of digraphs. In: Sachs, H., Voss, H.J., Walter, H.J. (eds.) *Beiträge zur Graphentheorie*, pp. 17–33. Teubner, Leipzig (1968)
2. Bodlaender, H.: A partial k -arboretum of graphs with bounded treewidth. *Theoretical Computer Science* **209**(1–2), 1–45 (1998)
3. Bodlaender, H., Drange, P., Dregi, M., Fomin, F., Lokshtanov, D., Pilipczuk, M.: A $O(c^k n)$ 5-approximation algorithm for treewidth. *SIAM Journal on Computing* **45**(2), 317–378 (2016)
4. Cao, Y., Ke, Y., Otachi, Y., You, J.: Vertex deletion problems on chordal graphs. *Theoretical Computer Science* **745**, 75–86 (2018)
5. Chudnovsky, M., Seymour, P.: The structure of claw-free graphs. In: Webb, B. (ed.) *Surveys in Combinatorics*. London Mathematical Society Lecture Note Series, vol. 327, pp. 153–171. Cambridge University Press (2005)
6. Courcelle, B., Makowsky, J., Rotics, U.: Linear time solvable optimization problems on graphs of bounded clique-width. *Theory of Computing Systems* **33**(2), 125–150 (2000)
7. Courcelle, B.: The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation* **85**(1), 12 – 75 (1990)
8. Crescenzi, P.: A short guide to approximation preserving reductions. In: *Proceedings of the 12th Annual IEEE Conference on Computational Complexity*. pp. 262–273 (1997)
9. Cygan, M., Nederlof, J., Pilipczuk, M., Pilipczuk, M., van Rooij, J., Wojtaszczyk, J.: Solving connectivity problems parameterized by treewidth in single exponential time. In: *Proceedings of the 52nd IEEE Annual Symposium on Foundations of Computer Science*. pp. 150–159 (2011)
10. Cygan, M., Philip, G., Pilipczuk, M., Pilipczuk, M., Wojtaszczyk, J.: Dominating set is fixed parameter tractable in claw-free graphs. *Theoretical Computer Science* **412**(50), 6982–7000 (2011)
11. Faenza, Y., Oriolo, G., Stauffer, G.: Solving the weighted stable set problem in claw-free graphs via decomposition. *Journal of the ACM* **61**(4), 20:1–20:41 (2014)

12. Faudree, R., Flandrin, E., Ryjáček, Z.: Claw-free graphs – A survey. *Discrete Mathematics* **164**(1), 87–147 (1997)
13. Fernau, H.: Parameterized algorithms for d -Hitting Set: The weighted case. *Theoretical Computer Science* **411**(16–18), 1698–1713 (2010)
14. Flum, J., Grohe, M.: Parameterized complexity theory. *Texts Theoret. Comput. Sci. EATCS Ser* (2006)
15. Garey, M., Johnson, D.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman and Company, San Francisco (1979)
16. Hedetniemi, S., Laskar, R.: Recent results and open problems in domination theory. In: *Applications of Discrete Mathematics*, pp. 205–218. SIAM Philadelphia (1988)
17. Hochbaum, D.: Approximation algorithms for the set covering and vertex cover problems. *SIAM Journal on Computing* **11**, 555–556 (1982)
18. Itai, A., Rodeh, M.: Finding a minimum circuit in a graph. *SIAM Journal on Computing* **7**(4), 413–423 (1978)
19. J.-L., E.F., Fouquet, Li, H.: On Hamiltonian claw-free graphs. *Discrete Mathematics* **111**(1–3), 221–229 (1993)
20. Khot, S.: On the power of unique 2-prover 1-round games. In: *Proceedings of the Thirty-Fourth Annual ACM Symposium on Theory of Computing*. pp. 767–775 (2002)
21. Khot, S., Vishnoi, N.: The unique games conjecture, integrality gap for cut problems and embeddability of negative-type metrics into ℓ_1 . *Journal of the ACM* **62**(1), 8 (2015)
22. Kloks, T.: *Treewidth – Computations and Approximations*, *Lecture Notes in Computer Science*, vol. 842. Springer-Verlag (1994)
23. Kumar, M., Mishra, S., Devi, N., Saurabh, S.: Approximation algorithms for node deletion problems on bipartite graphs with finite forbidden subgraph characterization. *Theoretical Computer Science* **526**, 90–96 (2014)
24. Le Gall, F.: Powers of tensors and fast matrix multiplication. In: *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*. pp. 296–303 (2014)
25. Lewis, J., Yannakakis, M.: The node-deletion problem for hereditary properties is NP-complete. *Journal of Computer and System Sciences* **20**(2), 219–230 (1980)
26. Lund, C., Yannakakis, M.: The approximation of maximum subgraph problems. In: Lingas, A., Karlsson, R., Carlsson, S. (eds.) *Proceedings of the Automata, Languages and Programming 1993*. *Lecture Notes in Computer Science*, vol. 700, pp. 90–96 (1993)
27. Roberts, F.: Indifference graphs. In: Harary, F. (ed.) *Proof Techniques in Graph Theory*, pp. 139–146. Academic Press (1969)
28. Robertson, N., Seymour, P.: Graph minors. III. Planar tree-width. *Journal of Combinatorial Theory. Series B* **36**(1), 49–64 (1984)
29. Sumner, D.: Graphs with 1-factors. *Proceedings of the American Mathematical Society* **42**(1), 8–12 (1974)
30. Tarjan, R.: Depth-first search and linear graph algorithms. *SIAM Journal on Computing* **1**(2), 146–160 (1972)
31. Thorup, M.: All structured programs have small tree width and good register allocation. *Information and Computation* **142**(2), 159–181 (1998)
32. Williams, V.V., Wang, J.R., Williams, R., Yu, H.: Finding four-node subgraphs in triangle time. In: *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*. pp. 1671–1680. SIAM (2015)
33. Yannakakis, M.: Node deletion problems on bipartite graphs. *SIAM Journal on Computing* **10**(2), 310–327 (1981)

Appendix

Some Definitions. Let G be a graph. Given a vertex $v \in V(G)$, its *open neighborhood* consists of all adjacent vertices to v and is denoted by $N_G(v)$, whereas its *closed neighborhood* is the set $N_G[v] = N_G(v) \cup \{v\}$. For a set $U \subseteq V(G)$, let $N_G(U) = \bigcup_{v \in U} N_G(v) \setminus U$, and $N_G[U] = N_G(U) \cup U$. When the graph G is clear from the context, we denote $N_G(v) \cap U$ by $N_U(v)$.

The *degree* of a vertex $v \in V(G)$ on a set $U \subseteq V(G)$, is $d_U(v) = |N_G(v) \cap U|$. If $U = V(G)$, we simply write $d_G(v)$. We say that $v \in V(G)$ is an *isolated* (resp. a *leaf*) vertex if $d_G(v) = 0$ (resp. $d_G(v) = 1$). A set $U \subseteq V(G)$ is called a *clique* if the vertices in U are pairwise adjacent.

For $U \subseteq V(G)$, the subgraph of G *induced* by U , denoted by $G[U]$, is the graph whose vertex set is U and whose edge set consists of all the edges in $E(G)$ that have both endpoints in U . If H is a subgraph of G , we write $H \subseteq G$. For $U \subseteq V(G)$, we denote by $G \setminus U$ the graph $G[V(G) \setminus U]$.

A graph is *connected* if every pair of vertices is joined by a path. A maximal connected subgraph of G is called a *connected component* of G . A graph G is called *k-connected* if $G \setminus X$ is connected for every set $X \subseteq V(G)$ with $|X| \leq k$. A *block* of a graph G is a maximal 2-connected subgraph of G . A vertex v of a graph G is a *cutpoint* if $G \setminus \{v\}$ has more connected components than G .

A *block graph* is a graph in which every block is a clique. A *forest* is an acyclic graph or, equivalently, a graph in which every block is an edge. A *linear forest* is the disjoint union of induced paths. A *tree* is a connected forest.

A *k-ary tree* is a rooted tree T in which every node of T has at most k children. In particular, for $k = 2$, and $k = 3$ we have the *binary*, and the *ternary* tree, respectively. A *strict k-ary tree* is a rooted tree T in which every node of T has either zero or k children. The *depth* of a vertex $v \in V(T)$ is the length of a path from v to r in T . A *full k-ary tree* is a strict k -ary tree in which all leaves have the same depth.

A graph G is a *split graph* if $V(G)$ admits a partition $V(G) = C \cup I$ into a clique C and an independent set I . A graph is *chordal* if every cycle of length greater than three has a *chord*, i.e., an edge between two non-consecutive vertices of the cycle. Forests, block graphs, and split graphs are all subclasses of chordal graphs.

Definition 1. [28] A tree decomposition of a graph G is a pair $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$, where T is a tree whose every node t is assigned a vertex subset $X_t \subseteq V(G)$ called bag, such that the following three conditions hold:

- $\bigcup_{t \in V(T)} X_t = V(G)$.
- For every $uv \in E(G)$, there exists a node t of T such that bag X_t contains both u and v .
- For every $u \in V(G)$, the set $T_u = \{t \in V(T) : u \in X_t\}$ induces a connected subgraph of T .

The *width* of a tree decomposition is $\max_{t \in V(T)} (|X_t| - 1)$. The *treewidth* $tw(G)$ of a graph G is the minimum possible width of a tree decomposition of G .

Definition 2. [22] A nice tree decomposition is a tree decomposition with one special node r called root with $X_r = \emptyset$, and each node is one of the following types:

- **Leaf node:** a leaf ℓ of T with $X_\ell = \emptyset$.
- **Introduce node:** a node t with exactly one child t' such that $X_t = X_{t'} \cup \{v\}$ for some vertex $v \notin X_{t'}$; we say that v is introduced at t .
- **Forget node:** a node t with exactly one child t' such that $X_t = X_{t'} \setminus \{v\}$ for some vertex $v \in X_{t'}$; we say that v is forgotten at t .
- **Join node:** a node t with two children t', t'' such that $X_t = X_{t'} = X_{t''}$.

Proof of Theorem 4. We will first prove that Algorithm 1 is correct for trees and that it runs in linear time. Then we will generalize the result to forests.

Theorem 8. Let T be a rooted tree of order n . A minimum claw-deletion set of T can be found by Algorithm 1 in $\mathcal{O}(n)$ time.

Proof. Let T be a rooted tree. We will prove by induction that Algorithm 1 is correct. The basis is the case when $C(v) = \emptyset$. Since T_v^+ consists either of a single edge or of a single vertex (when $p = \emptyset$), clearly the empty set is a minimum claw-deletion set of T_v^+ . Moreover, $\text{cdn}(T_v^+) = \text{cdn}(T_v) = \text{cdn}(T_v \setminus \{v\})$. Hence, function CLAW-DELETION-SET is correct when $C(v) = \emptyset$.

Suppose that $C(v) \neq \emptyset$ and let $C(v) = \{u_1, \dots, u_k\}$, for $k \geq 1$. For the inductive hypothesis, we assume that $S_i = \text{CLAW-DELETION-SET}(T, u_i, v)$ is a minimum claw-deletion set of $T_{u_i}^+$, for every $1 \leq i \leq k$, such that: if $\text{cdn}(T_{u_i}^+) = 1 + \text{cdn}(T_{u_i})$ then $v \in S_i$; if $\text{cdn}(T_{u_i}^+) = \text{cdn}(T_{u_i})$ and $\text{cdn}(T_{u_i}) = 1 + \text{cdn}(T_{u_i} \setminus \{u_i\})$ then $u_i \in S_i$.

Let $S = S_1 \cup \dots \cup S_k$.

If $v \in S$, then the connected components of $T_v^+ \setminus S$ are $\{p\}$ (when $p \neq \emptyset$) and the connected components of $T_{u_i}^+ \setminus S$, for $1 \leq i \leq k$, which, by inductive hypothesis, are induced paths. So S is a claw-deletion set of T_v^+ . Also, by minimality, $S_i \setminus \{v\}$ is a minimum claw-deletion set of T_{u_i} , for every $1 \leq i \leq k$. Let $1 \leq j \leq k$ such that $v \in S_j$. Then $\text{cdn}(T_v^+) \geq \text{cdn}(T_v) \geq \text{cdn}(T_{u_j}^+) + \sum_{1 \leq i \leq k; i \neq j} \text{cdn}(T_{u_i}) = |S|$. Thus, S is a minimum claw-deletion set of T_v^+ and $\text{cdn}(T_v^+) = \text{cdn}(T_v)$. This also implies that S satisfies the further conditions required to the output.

From now on, suppose that $v \notin S$. Then, by inductive hypothesis, $\text{cdn}(T_{u_i}^+) = \text{cdn}(T_{u_i})$ and, moreover, S_i is also a minimum claw-deletion set of T_{u_i} , for every $1 \leq i \leq k$. Let $c = |C(v) \setminus S|$. For each $u_i \in C(v) \setminus S$, it also holds $\text{cdn}(T_{u_i}) = \text{cdn}(T_{u_i} \setminus \{u_i\})$ and S_i is a minimum claw-deletion set of $T_{u_i} \setminus \{u_i\}$.

Suppose first that $c \leq 1$, i.e., $C(v) \setminus S \subseteq \{u_j\}$ for some $1 \leq j \leq k$. Then, the connected components of $T_v^+ \setminus S$ are the connected components of $T_{u_i} \setminus S$, for $1 \leq i \leq k$, $i \neq j$, plus the connected components of $T_{u_j}^+ \setminus S$ which, by inductive hypothesis, are induced paths, with the addition of vertex p (when $p \neq \emptyset$) to the path containing v . It is easy to see that the resulting component is still an induced path. So S is a claw-deletion set of T_v^+ . Since $\text{cdn}(T_v^+) \geq \text{cdn}(T_v) \geq \sum_{1 \leq i \leq k} \text{cdn}(T_{u_i}) = |S|$, S is a minimum claw-deletion set of T_v^+ and

$\text{cdn}(T_v^+) = \text{cdn}(T_v) = \text{cdn}(T_v \setminus \{v\})$. This also implies that S satisfies the further conditions required to the output.

Suppose now that $c \geq 3$. Using the inductive hypothesis and similarly to the case where $v \in S$, it is not difficult to see that $S \cup \{v\}$ is a claw-deletion set of T_v^+ . Moreover, $\text{cdn}(T_v^+) \geq \text{cdn}(T_v) \geq \text{cdn}(T[\{v\} \cup C(v) \setminus S]) + \sum_{u_i \in C(v) \setminus S} \text{cdn}(T_{u_i} \setminus \{u_i\}) + \sum_{u_i \in C(v) \cap S} \text{cdn}(T_{u_i}) = 1 + |S|$. This shows that $S \cup \{v\}$ is a minimum claw-deletion set of T_v^+ and $\text{cdn}(T_v^+) = \text{cdn}(T_v)$. So $S \cup \{v\}$ satisfies the required conditions.

Finally, suppose that $c = 2$, i.e., $C(v) \setminus S = \{u_j, u_{j'}\}$ for some $1 \leq j < j' \leq k$. The connected components of $T_v \setminus S$ are the connected components of $T_{u_i} \setminus S$, for $1 \leq i \leq k$, $i \neq j, j'$ plus the connected components of $T_{u_j}^+ \setminus S$ and of $T_{u_{j'}}^+ \setminus S$ not containing v which, by inductive hypothesis, are induced paths, plus a path having $u_j v u_{j'}$ as a subpath. So S is a claw-deletion set of T_v and $S \cup \{p\}$ is a claw-deletion set of T_v^+ when $p \neq \emptyset$. Notice that, when $p \neq \emptyset$, $T[\{p, v, u_j, u_{j'}\}]$ is a claw, so $\text{cdn}(T[\{p, v, u_j, u_{j'}\}]) = 1$. When $p = \emptyset$, $\text{cdn}(T[\{p, v, u_j, u_{j'}\}]) = 0$. Then $\text{cdn}(T_v^+) \geq \text{cdn}(T[\{p, v, u_j, u_{j'}\}]) + \text{cdn}(T_{u_j} \setminus \{u_j\}) + \text{cdn}(T_{u_{j'}} \setminus \{u_{j'}\}) + \sum_{1 \leq i \leq k; i \neq j, j'} \text{cdn}(T_{u_i}) = 1 + |S|$ when $p \neq \emptyset$, and $|S|$ otherwise. This shows that $S \cup \{p\}$ (resp. S) is a minimum claw-deletion set of T_v^+ when $p \neq \emptyset$ (resp. when $p = \emptyset$). In the first case, by minimality, S is also a minimum claw-deletion set of T_v , so $\text{cdn}(T_v^+) = 1 + \text{cdn}(T_v)$, and $S \cup \{p\}$ satisfies the required conditions. In the second case, $\text{cdn}(T_v) = \text{cdn}(T_v \setminus \{v\})$, so S satisfies the required conditions.

Therefore, CLAW-DELETION-SET returns correctly a minimum claw-deletion set of T_v^+ satisfying that if $\text{cdn}(T_v^+) = 1 + \text{cdn}(T_v)$ then $p \in S$, and if $\text{cdn}(T_v^+) = \text{cdn}(T_v)$ and $\text{cdn}(T_v) = 1 + \text{cdn}(T_v \setminus \{v\})$ then $v \in S$.

Next, we perform the runtime analysis of Algorithm 1.

First, we have that checking each conditional statement of Algorithm 1 requires $\mathcal{O}(1)$ time if the tree is represented by lists of children. Initializing $S = \emptyset$ at the very beginning of the algorithm can be done in $\mathcal{O}(n)$ time by representing S by an array. In that case, adding a vertex to S can be done in constant time. The assignment and union operations of Line 6 of the algorithm are not necessary if all the recursive calls work on the same array representing the set S . Line 7 computes the number of children of a vertex v which are not in S . Having the list of children and S represented by an array, this step takes $\mathcal{O}(d_T(v))$ time. Since function CLAW-DELETION-SET is executed exactly one time for every vertex $v \in V(T)$, we conclude that Algorithm 1 runs in $\mathcal{O}(n + m) = \mathcal{O}(n)$ time. \square

From Theorem 8, we obtain the following Corollary 4, and together imply Theorem 4.

Corollary 4. *Given a forest F , and a positive integer k , the problem of deciding whether F can be transformed into a linear forest with at most k vertex deletions can be solved in linear time.*

Exact Values for Full k -ary Trees. We determine the claw-deletion number of a k -ary tree T with height h , as a function of k and h . The cases $k = 2$ and $k \geq 3$ follow in Theorems 9 and 10, respectively.

Theorem 9. *Let T be a full binary tree of height h , and $t = (h + 1) \bmod 3$. Then $\text{cdn}(T) = (2^{h+1} - 2^t)/7$.*

Proof. Algorithm 1 chooses a claw-deletion S of T comprised by all the vertices in depth $h - 2$. Subsequently, the same procedure chooses all the vertices in depth $h - 5$, and so on, until the depth $t = (h + 1) \bmod 3$. For every $1 \leq i \leq k$, the amount of vertices in depth i is 2^i . Then

$$\text{cdn}(T) = |S| = 2^{h-2} + 2^{h-5} + \dots + 2^t.$$

That leads to a geometric progression with ratio $r = 2^{-3}$, and $(h - t + 1)/3$ terms, which results in $\text{cdn}(T) = (2^{h+1} - 2^t)/7$. \square

The result of Theorem 9 can be rewritten as a function of the order of T .

Proof of Corollary 1. *Let T be a full binary tree with n vertices, and $t = \log_2(n + 1) \bmod 3$. Then $\text{cdn}(T) = \frac{n + 1 - 2^t}{7}$.*

Proof. We know that a full binary tree with n vertices has height $h = \log_2(n + 1) - 1$. By Theorem 9 with $t = \log_2(n + 1) \bmod 3$, we obtain

$$\text{cdn}(T) = \frac{2^{h+1} - 2^t}{7} = \frac{2^{\log_2(n+1)} - 2^t}{7} = \frac{n + 1 - 2^t}{7}.$$

\square

Next, we proceed to full k -ary trees with $k \geq 3$ in Theorem 10.

Theorem 10. *Let T be a full k -ary tree of height h , for $k \geq 3$, and $t = (h - 1) \bmod 2$. Then $\text{cdn}(T) = (k^{h+1} - k^t)/(k^2 - 1)$.*

Proof. Algorithm 1 chooses a claw-deletion S of T comprised by all the vertices in depth $h - 1$, all the vertices in depth $h - 3$, and so on, until the depth $t = (h - 1) \bmod 2$. Then,

$$\text{cdn}(T) = |S| = k^{h-1} + k^{h-3} + \dots + k^t.$$

That leads to a geometric progression with ratio $r = k^{-2}$, and $(h - t + 1)/2$ terms, which follows that $\text{cdn}(T) = (k^{h+1} - k^t)/(k^2 - 1)$. \square

Theorem 10 rewritten as a function of the order of T follows below.

Proof of Corollary 2. *Let T be a full k -ary tree with n vertices, for $k \geq 3$, and $t = \log_k(nk - n + 1) \bmod 2$. Then $\text{cdn}(T) = \frac{nk - n + 1 - k^t}{k^2 - 1}$.*

Proof. We know that a full k -ary tree with n vertices has height $h = \log_k(nk - n + 1) - 1$. By Theorem 10 with $t = \log_k(nk - n + 1) \bmod 2$, we obtain

$$\text{cdn}(T) = \frac{k^{h+1} - k^t}{k^2 - 1} = \frac{k^{\log_k(nk-n+1)} - k^t}{k^2 - 1} = \frac{nk - n + 1 - k^t}{k^2 - 1}.$$

\square

We establish the proportion of vertices in $V(T)$ that belongs to a claw-deletion set of T .

Corollary 5. *Let T be a full k -ary tree of order n and height h . Let $t = (h + 1) \bmod 3$ and $t' = (h - 1) \bmod 2$. It holds that*

$$\frac{\text{cdn}(T)}{n} = \begin{cases} \frac{2^{h+1}-2^t}{7(2^{h+1}-1)}, & \text{if } k = 2; \\ \frac{k^{h+1}-k^{t'}}{(k+1)(k^{h+1}-1)}, & \text{if } k \geq 3. \end{cases}$$

In addition, $t = t' = 0$ implies

$$\frac{\text{cdn}(T)}{n} = \begin{cases} 1/7, & \text{if } k = 2; \\ 1/(k+1), & \text{if } k \geq 3. \end{cases}$$

Among full k -ary trees, $k = 3$ maximizes the proportion of vertices in a claw-deletion set.

Proof of Theorem 5. *Let G be a weighted connected block graph which is not complete. Let T be the block-cutpoint-tree of G , rooted at a cutpoint r . The previous function $f(r)$ computes correctly the minimum weight of a claw-deletion set of G .*

Proof. We will prove by induction (bottom-up), that f_1, f_2, f_3 on $V(T)$ correctly compute the weight stated in their definition. In that case, being r a cutpoint of G and $G_r = G$, it is clear that $f(r)$ computes the minimum weight of a claw-deletion set of G .

Let b be a leaf of T . Then b is a block of G , and G_b and G_b^- are complete, so any set is a claw-deletion set of G_b and G_b^- . Moreover, every vertex of $b \setminus \{p_T(b)\}$ is simplicial in G , so the weight of $b \setminus \{p_T(b)\}$ is $s(b)$. Thus, $f_1(b) = s(b)$, $f_2(b) = f_3(b) = 0$ is correct.

Now, let v be a cutpoint of G and, by inductive hypothesis, assume that for the children of v in T the values of f_1, f_2 , and f_3 are correct according to their definition.

Consider first $f_1(v)$, i.e., the minimum weight of a claw-deletion set of G_v containing v . The connected components of $G_v \setminus \{v\}$ are $\{G_b^-\}_{b \in C(v)}$. So, it is enough to compute the minimum weight of a claw-deletion set of each of them, and add to their sum the weight of v , so $f_1(v) = w(v) + \sum_{b \in C(v)} f_2(b)$.

Consider next $f_2(v)$, i.e., the minimum weight of a claw-deletion set of G_v not containing v . In this case, we have to avoid claws having v as a center and the tree leaves in three distinct blocks of $C(v)$, so all but at most two of the blocks have to be completely contained in the set, except for vertex v . For the remaining (at most two) blocks b , we need to compute the minimum weight of a claw-deletion set of G_b not containing v (which is $p_T(b)$). This justifies the formula $f_2(v) = \sum_{b \in C(v)} f_3(b)$ for $|C(v)| \leq 2$, and $f_2(v) = \min_{b_1, b_2 \in C(v)} (\sum_{b \in \{b_1, b_2\}} f_3(b) + \sum_{b \in C(v) \setminus \{b_1, b_2\}} f_1(b))$, otherwise.

Finally, consider $f_3(v)$, for $v \neq r$, i.e., the minimum weight of a claw-deletion set of G_v^+ containing neither v nor all the vertices of $p_T(v) \setminus \{v\}$ (recall that

$p_T(v)$ is a block). In this case, we have to avoid claws having v as a center, one leaf in $p_T(v) \setminus \{v\}$, and two other leaves in two distinct blocks of $C(v)$. So all but at most one of the blocks have to be completely contained in the set, except for vertex v . For the remaining block b , we need to compute the minimum weight of a claw-deletion set of G_b not containing v (which is $p_T(b)$). This justifies the formula $f_3(v) = f_3(b)$ when $C(v) = \{b\}$, and $f_3(v) = \min_{b_1 \in C(v)} (f_3(b_1) + \sum_{b \in C(v) \setminus \{b_1\}} f_1(b))$, otherwise.

To conclude the proof, let b be a node which is a block of G and, by inductive hypothesis, assume that for the children of b the values of f_1 , f_2 , and f_3 are correct according to their definition.

Consider first $f_1(b)$, i.e., the minimum weight of a claw-deletion set of G_b^- containing all the vertices of $b \setminus \{p_T(b)\}$. All the claws containing vertices of b are hit by the set by definition, so it is enough to compute for every v in $C(b)$ the minimum weight of a claw-deletion set of G_v containing v , and adding to it the weight of all the simplicial vertices of b , that is, $s(b)$. Then the formula $f_1(b) = s(b) + \sum_{v \in C(b)} f_1(v)$ is correct.

Consider next $f_3(b)$, the minimum weight of a claw-deletion set of G_b not containing $p_T(b)$. For each v in $C(b)$, either v belongs to the set, or v does not belong to the set and there is another vertex of b that does not belong to the set. So, we have to recursively compute $f_1(v)$ or $f_3(v)$, respectively, and choose the minimum. In this case the simplicial vertices do not belong to the minimum weight set, since the weights are positive. This justifies the formula $f_3(b) = \sum_{v \in C(b)} \min\{f_1(v), f_3(v)\}$.

Finally, consider $f_2(b)$, i.e., the minimum weight of a claw-deletion set of G_b^- . For each v in $C(b)$, there are three possibilities: either v belongs to the set, or v does not belong to the set and there is another vertex of $b \setminus p_T(b)$ that does not belong to the set, or v does not belong to the set but any other vertex of $b \setminus p_T(b)$ belongs to the set. We have to consider the third possibility for every v in $C(b)$, adding $f_2(v)$ to the sum of $f_1(v')$ for every other v' in $C(b)$, and in that case adding also $s(b)$. For the first two possibilities, the situation is similar to the one in the computation of $f_3(b)$. This justifies the formula $f_2(b) = \min\{\sum_{v \in C(v)} \min\{f_1(v), f_3(v)\}, \min_{v_1 \in C(v)} (s(b) + f_2(v_1) + \sum_{v \in C(v) \setminus \{v_1\}} f_1(v))\}$. \square

In Theorem 11 we analyze the time to compute $f(r)$.

Theorem 11. *Let G be a weighted connected block graph with n vertices. Given a block-cutpoint-tree of G , the minimum weight of a claw-deletion set of G can be determined in $\mathcal{O}(n)$ time.*

Proof. If the graph G is complete, the weight is zero. Otherwise, we root the given block-cutpoint-tree T of G at a cutpoint r of G . Notice that $|V(T)|$ and $|E(T)|$ are $\mathcal{O}(n)$.

Then we compute bottom-up the functions f_1 , f_2 , f_3 . The computation for a leaf b (recall that leaves of T are blocks of G) takes $\mathcal{O}(|b| - 1)$ time. Notice that $|C(b)|$ is also $\mathcal{O}(|b| - 1)$ for a block b of G which is not a leaf. Thus, the

computation of $f_1(b)$ and $f_3(b)$ is also $\mathcal{O}(|b| - 1)$. We can compute (as a fourth function) the difference $f_2(v) - f_1(v)$ for every cutpoint v of G . So, for the computation of $\min_{v_1 \in C(v)} (s(b) + f_2(v_1) + \sum_{v \in C(v) \setminus \{v_1\}} f_1(v))$ we simply choose as v_1 the vertex v minimizing $f_2(v) - f_1(v)$. Therefore the computation of $f_2(b)$ can be also done in $\mathcal{O}(|b| - 1)$ time.

For the vertices v which are cutpoints of G , we can compute (as a fourth function) the difference $f_3(b) - f_1(b)$ for every block b of G . In this way, we can compute each of $f_1(v)$, $f_2(v)$, and $f_3(v)$ in $\mathcal{O}(|C(v)|)$ time.

The whole complexity of the algorithm is then $\mathcal{O}(|V(T)| + |V(G)|) = \mathcal{O}(n)$. \square

Recall that a block-cutpoint-tree of a connected graph G with n vertices and m edges can be computed in $\mathcal{O}(n+m)$ time, as well as the connected components of a graph. This implies

Corollary 3. *Let G be a weighted block graph with n vertices and m edges. The minimum weight of a claw-deletion set of G can be determined in $\mathcal{O}(n+m)$ time.*

Proof of Running Time of Theorem 7. WEIGHTED CLAW-FREE VERTEX DELETION can be solved in linear time on graphs with bounded treewidth. More precisely, there is a $2^{\mathcal{O}(k^2)} \cdot n$ -time algorithm to solve WEIGHTED CLAW-FREE VERTEX DELETION on n -vertex graphs G with treewidth at most k .

Proof. We analyze the time to compute $\text{cdn}_w[r, \emptyset, \emptyset, \emptyset, \emptyset]$. Since $\text{tw}(G) \leq k$, then $|X_t| = \mathcal{O}(k)$, for every node $t \in V(T)$. For every leaf node t , function runs in constant time.

Let t be an introduce node. Functions of Formulas (2.1) and (2.3) run in constant time. Function (2.2) requires $\mathcal{O}(k^{2 \cdot 3728639})$ time [18] for checking if $N_{X_t \setminus S}(v)$ does not induce a \overline{K}_3 , $\mathcal{O}(|X_t \times X_t|) = \mathcal{O}(k^2)$ for checking for every $(x, y) \in Z$, if $vx \in E(G)$ or $vy \notin E(G)$, $\mathcal{O}(|C|) = \mathcal{O}(k)$ for checking if $N_{X_t}(v) \cap C = \emptyset$, and $\mathcal{O}(|X_t \times X_t|) = \mathcal{O}(k^2)$ for the final condition. Such steps are executed for every partition $S \cup A \cup B \cup C$ of X_t , which has $4^{\mathcal{O}(k)}$ possibilities, and for every $Z \subseteq X_t \times X_t$, which leads to $2^{\mathcal{O}(k^2)}$ choices of Z . Since the first is dominated by the latter, we obtain that computing cdn_w for an introduce node requires $2^{\mathcal{O}(k^2)}$ time.

Let t be a forget node. Formula (3.1) runs in $\mathcal{O}(1)$. The minimum value asked for Formula (3.2) is obtained by checking every (S, A', B', C', Z') , which is bounded by the size of the power set of $X_t \times X_t$, $2^{\mathcal{O}(k^2)}$. Since all steps are executed for every partition $S \cup A \cup B \cup C$ of X_t and for every $Z \subseteq X_t \times X_t$, the time required for a forget node t is $2^{\mathcal{O}(k^2)}$.

Finally, let t be a join node. Let $S \cup A \cup B \cup C$ be a partition of X_t , and $Z \subseteq X_t \times X_t$. The value asked for Formula (4) is obtained by the minimum sum of cdn_w in t' and in t'' , among all possibilities of (A', B', C', Z') and (A'', B'', C'', Z'') , where the pair must satisfy (4). This leads to a running time of $2^{\mathcal{O}(k^2)} \cdot 2^{\mathcal{O}(k^2)} \cdot \mathcal{O}(k)$. Those steps are executed for every partition $S \cup A \cup B \cup C$

of X_t and for every $Z \subseteq X_t \times X_t$. Hence, the total running time for computing cdn_w for a join node t is bounded by $2^{\mathcal{O}(k^2)}$.

Since the time to compute each entry of the table is upper bounded by $2^{\mathcal{O}(k^2)}$ and the table has size $2^{\mathcal{O}(k^2)} \cdot n$, the algorithm can be performed in $2^{\mathcal{O}(k^2)} \cdot n$ time. This implies linear-time solvability for graphs with bounded treewidth. \square