Parameterized Graph Cleaning Problems*

Dániel Marx and Ildikó Schlotter

Department of Computer Science and Information Theory,
Budapest University of Technology and Economics,
H-1521 Budapest, Hungary.
{dmarx,ildi}@cs.bme.hu

Abstract. We investigate the Induced Subgraph Isomorphism problem with non-standard parametrization, where the parameter is the difference $|V(G)| - |V(H)|$ with H and G being the smaller and the larger input graph, respectively. Intuitively, we can interpret this problem as "cleaning" the graph G, regarded as a pattern containing extra vertices indicating errors, in order to obtain the graph H representing the original pattern. We show fixed-parameter tractability of the cases where both H and G are planar and H is 3-connected, or H is a tree and G is arbitrary. We also prove that the problem when H and G are both 3-connected planar graphs is NP-complete without parametrization.

1 Introduction

Problems related to graph isomorphisms play a significant role in algorithmic graph theory. The Induced Subgraph Isomorphism problem is one of the basic problems of this area: given two graphs H and G, find an induced subgraph of G isomorphic to H, if this is possible. In this general form, Induced Subgraph Isomorphism is NP-hard, since it contains several well-known NP-hard problems, such as Clique, Independent Set and Induced Path.

As Induced Subgraph Isomorphism has a wide range of important applications, polynomial time algorithms have been given for numerous special cases, such as the case when both input graphs are trees [23] or 2-connected outerplanar graphs [20]. However, Induced Subgraph Isomorphism remains NP-hard even if H is a forest and G is a tree, or if H is a path and G is a cubic planar graph [15]. In many fields where researchers face hard problems, parameterized complexity theory (see e.g. [11] or [13]) has proved to be successful in the analysis and design of algorithms that have a tractable running time in several applications. In parameterized complexity, a parameter k is associated with the input I of the problem. A parameterized problem is *fixed-parameter tractable* (FPT) if it admits an algorithm with running time $O(f(k)|I|^c)$ where f is an arbitrary function and c is a constant independent of k. Such an algorithm is efficient even for large instances provided that the parameter k is a small value.

* Research funded by the Hungarian National Research Fund (OTKA grant 67651). The first author is supported by Magyary Zoltán Felsőoktatási Közalapítvány.

Note that Induced Subgraph Isomorphism is trivially solvable in time $O(|V(G)|^{|V(H)|}|V(H)|^2)$ on input graphs H and G by trying every possible subgraph of G, or more precisely, by checking for every possible injective mapping from $V(H)$ to $V(G)$ whether it is an isomorphism. As H is typically much smaller than G in applications related to pattern matching, the usual parametrization of Induced Subgraph Isomorphism is to define the parameter k to be $|V(H)|$. FPT algorithms with this parametrization are known if G is planar [12], has bounded degree [5], or if H is a log-bounded fragmentation graph and G has bounded treewidth [16]. We note that this parametrization yields a W[1]-complete problem when G can be arbitrary and $H \in \mathcal{H}$ for some graph class $\mathcal{H}$ of infinite cardinality [7]. W[1]-completeness of these problems shows that they are unlikely to be FPT. For more on the hardness theory of parameterized complexity, consult [11] or [13].

In this paper we consider another parametrization of Induced Subgraph Isomorphism, where the parameter is the difference $|V(G)|-|V(H)|$. Considering the presence of extra vertices as some kind of error or noise, the problem of finding the original graph H in the "dirty" graph G containing errors is clearly meaningful. In other words, the task is to "clean" the graph G containing errors in order to obtain H. For two graph classes $\mathcal{H}$ and $\mathcal{G}$ we define the Cleaning$(\mathcal{H},\mathcal{G})$ problem: given a pair of graphs (H,G) with $H \in \mathcal{H}$ and $G \in \mathcal{G}$, find a set of vertices S in G such that $G-S$ is isomorphic to H. The parameter associated with the input (H,G) is $|V(G)|-|V(H)|$. Clearly, the size of the set S to be found has to be equal to the parameter. For the case when $\mathcal{G}$ or $\mathcal{H}$ is the class of all graphs, we will use the notation Cleaning$(\mathcal{H},-)$ or Cleaning$(-,\mathcal{G})$, respectively.

In the special case when the parameter is 0, the problem is equivalent to the Graph Isomorphism problem, so we cannot hope to give an FPT algorithm for the general problem Cleaning$(-,-)$. Thus, we consider two special cases. We give FPT algorithms for the problems Cleaning(*Tree*,$-$) and Cleaning(*3-Connected-Planar*, *Planar*) where *Tree*, *Planar*, and *3-Connected-Planar* denote the class of trees, planar graphs, and 3-connected planar graphs, respectively. Note that these problems differ from the Feedback Vertex Set and the Minimum Apex problems, where the task is to delete a minimum number of vertices from the input graph to get an *arbitrary* acyclic or planar graph, respectively. Both these problems are FPT [3, 21].

Without parametrization, Cleaning(*Tree*,$-$) is NP-hard because it contains Induced Path, and we show NP-hardness for Cleaning(*3-Connected-Planar, 3-Connected-Planar*) too. A polynomial-time algorithm is known for Cleaning(*Tree, Tree*) [23], and an FPT algorithm is known for Cleaning(*Grid*,$-$) where *Grid* is the class of rectangular grids [9].

In Table 1 we summarize the complexity of the Cleaning$(\mathcal{H},\mathcal{G})$ problem for different graph classes $\mathcal{H}$ and $\mathcal{G}$. Table 1 contains results for three versions of the problem: without parametrization, with the standard parameter $|V(H)|$, and finally with the parameter $|V(G)|-|V(H)|$ yielding the parametrization discussed in this paper.

	Parameter		
Graph classes $(\mathcal{H}, \mathcal{G})$	None	$\lvert V(H)\rvert$	$\lvert V(G)\rvert - \lvert V(H)\rvert$
(*Tree*, *Tree*)	P [23]	FPT (trivial)	FPT (trivial)
(*Tree*, $-$)	NP-complete [15]	W[1]-complete [6]	FPT*
(*3-Connected-Planar*, *Planar*)	NP-complete*	FPT [12]	FPT*
($-$, *Planar*)	NP-complete [15]	FPT [12]	Open
(*Grid*, $-$)	NP-complete [15]	W[1]-complete [7]	FPT [9]

Table 1. Summary of some known results for the Cleaning$(\mathcal{H}, \mathcal{G})$ problem. The results obtained in this paper are marked with an asterisk.

The notation we use is briefly described in Sect. 2. In Sect. 3, we present our hardness result for Cleaning(*3-Connected-Planar, 3-Connected-Planar*) and an FPT algorithm for Cleaning(*3-Connected-Planar, Planar*). We discuss the Cleaning(*Tree*,$-$) problem in Sect. 4. Finally, we present some open questions and possibilities for further research in Sect. 5.

2 Notation

We write $[n]$ for $\{1, \ldots, n\}$. For a graph G, $V(G)$ denotes its vertices and $E(G)$ its edges. The set of the neighbors of $x \in V(G)$ is $N_G(x)$, and for some $X \subseteq V(G)$ we let $N_G(X) = \bigcup_{x \in X} N_G(x)$. The degree of x in G is $d_G(x) = |N_G(x)|$. If $Z \subseteq V(G)$ and G is clear from the context, then we let $N_Z(x) = N_G(x) \cap Z$ and $N_Z(X) = N_G(X) \cap Z$. For some $X \subseteq V(G)$, $G - X$ is obtained from G by deleting X, and $G[X] = G - V(G - X)$. For a subgraph H of G, let $G - H = G - V(H)$.

A *plane graph* is a planar graph together with a planar embedding. For a subgraph H of a plane graph G, an edge $e \in E(H)$ is called an *outer edge* of (H, G) if G has a face F_e incident to e which is not in H. In this case, F_e is an *outer face* of e w.r.t. (H, G). The *border* of H in G is the subgraph formed by the outer edges of (H, G). An *isomorphism* from H into G is a bijection $\varphi : V(H) \cup E(H) \to V(G) \cup E(G)$ preserving incidency. For a subgraph H' of H, $\varphi(H')$ is the subgraph of G that consists of the images of the vertices and edges of H'.

3 The Cleaning(*3-Connected-Planar, Planar*) problem

In this section, we present an algorithm for Cleaning(*3-Connected-Planar, Planar*). Since 3-connected planar graphs can be considered as "rigid" graphs in the sense that they cannot be embedded in the plane in essentially different ways, this problem seems to be easy. However, Theorem 1 shows that it is NP-hard.

Theorem 1. CLEANING(*3-Connected-Planar, 3-Connected-Planar*) *is* NP*-hard.*

Proof. We give a reduction from the NP-complete PLANAR 3-COLORABILITY problem [15]. Let F be the planar input graph given. W.l.o.g. we assume that F is connected. We construct 3-connected planar graphs H and G such that CLEANING(*3-Connected-Planar, 3-Connected-Planar*) with input (H, G) is solvable if and only if F is 3-colorable.

The gadgets we construct are shown in Fig. 1. For every $x \in V(F)$ we set an integer $9|V(F)| \leq b(x) \leq 10|V(F)|$ such that $b(x) \neq b(y)$ for any $x \neq y \in V(F)$. For every vertex $x \in V(F)$ we build a *node-gadget* N_x in G as follows. We introduce a central vertex a^x, together with a cycle consisting of the vertices $b^x_0, b^x_1, \ldots, b^x_{6b(x)-1}$ with each b^x_i being connected to a^x, and finally the vertices $c^x_0, \ldots, c^x_{3b(x)-1}$ with each c^x_i being connected to three consecutive vertices from the cycle $b^x_0 b^x_1 \ldots b^x_{6b(x)-1}$, as illustrated in Fig. 1. Formally, the edge set of the node-gadget N_x is $\{a^x b^x_j, b^x_{j-1} b^x_j \mid j \in [6b(x)]\} \cup \{c^x_j b^x_{2j}, c^x_j b^x_{2j+1}, c^x_j b^x_{2j+2} | 0 \leq j < 3b(x)\}$ where $b^x_{6b(x)} = b^x_0$. The node-gadget N_x can be considered as a plane graph, supposing that the vertices $b^x_0, b^x_1, \ldots, b^x_{6b(x)-1}$ (and thus $c^x_0, c^x_1, \ldots, c^x_{3b(x)-1}$) are embedded in a clockwise order around a^x. We define the j-th *block* B^x_j of N_x to be $(c^x_{3j}, c^x_{3j+1}, c^x_{3j+2})$, for every $0 \leq j < b(x)$. The *type* of c^x_j can be 0, 1, or 2, according to the value of j modulo 3. We set $C_x = \{c^x_j | 0 \leq j < 3b(x)\}$.

Let us fix an arbitrary ordering of the vertices of F. For each $x < y$ with $xy \in E(F)$ we build a *connection* E_{xy} in G that uses 9 consecutive blocks from each of N_x and N_y, say $B^x_i, \ldots, B^x_{i+8}$ and $B^y_j, \ldots, B^y_{j+8}$. These blocks are the *base blocks* for E_{xy}, and we also define $b(x, y) = (i, j)$. Note that since $b(x) \geq 9|V(F)| > 9d_F(x)$, we can define connections such that no two connections share a common base block. To build E_{xy} with $b(x, y) = (i, j)$, we introduce new vertices $d^{xy}_1, d^{xy}_2, d^{xy}_3$ and edges $\{c^x_{3i+26-6m+\ell} d^{xy}_m, c^y_{3j+6m-\ell} d^{xy}_m | m \in [3], \ell \in [6]\} \cup \{c^x_{3i} c^y_{3j+24}, c^x_{3i+4} c^y_{3j+22}, c^x_{3i+8} c^y_{3j+20}\}$ (see Fig. 1). By choosing the base blocks for each connection in a way that the order of the connections around a node-gadget is the same as the order of the corresponding edges around the corresponding vertex for some fixed planar embedding of F, we can give a planar embedding of G. Moreover, it is easy to see that G is also 3-connected.

To construct H, we make a disjoint copy $\bar{G}$ of G, and delete some edges and vertices from it as follows. For the copy of c^x_j (a^x, C_x, etc.) we write $\bar{c}^x_j$ ($\bar{a}^x$, $\bar{C}^x$, etc. respectively). To get H, we delete from $\bar{G}$ the three edges connecting vertices of $\bar{C}_x$ and $\bar{C}_y$ for every $x < y$ and $xy \in E(F)$, and also the vertices $\bar{c}^x_{3j+1}$ and $\bar{c}^x_{3j+2}$ for every $x \in V(F), 0 \leq j < b(x)$. Clearly, H is planar, and observe that it remains 3-connected.

Now, we prove that if CLEANING(*3-Connected-Planar, 3-Connected-Planar*) has a solution S for the input (H, G), then F is 3-colorable. Let φ be an isomorphism from H to $G - S$. First, observe that since $b(x) \neq b(y)$ if $x \neq y$, and the integers $\{b(x) | x \in V(F)\}$ are large enough, φ must map $\bar{a}^x$ to a^x because of its degree. For each $x \in V(F)$, the vertices in $C_x \setminus S$ must have the same type, so let the color of x be this type. If $xy \in E(F)$, then the color of x and y must differ, otherwise one of the edges $c^x_{3i} c^y_{3j+24}, c^x_{3i+4} c^y_{3j+22}, c^x_{3i+8} c^y_{3j+20}$ would be in

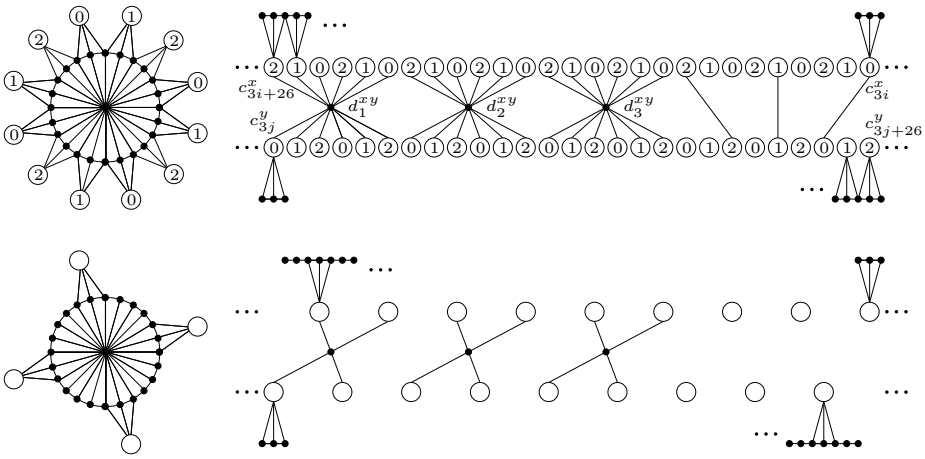

Fig. 1. A node-gadget and a connection in G, and the corresponding subgraphs of H used in the proof of Theorem 1.

$G - S$ where $b(x, y) = (i, j)$, as for every type t, one of these edges connects two vertices of type t. Thus the coloring is proper.

For the other direction, let $t : V(F) \to \{0, 1, 2\}$ be a coloring of F. For each $x \in V(F)$, let S contain those vertices in C_x whose type is not $t(x)$. Let φ map $\bar{a}^x$ and $\bar{d}^{xy}_m$ (for every meaningful x, y, m) to a^x and d^{xy}_m, respectively, and let φ map $\bar{c}^x_j$ to $c^x_{j+t(x)}$. By adjusting φ on the vertices $\bar{b}^x_i$ in the natural way, we can prove that φ is an isomorphism. It is clear that the restriction of φ on $\bar{N}_x$ is an isomorphism. Note that the only vertex of B^x_j present in $G - S$ is $c^x_{3j+t(x)} = \varphi(\bar{c}^x_{3j})$, so independently from $t(x)$ and $t(y)$, the neighborhood of $\bar{d}^{xy}_m$ is also preserved. We only have to check that the edges connecting C_x and C_y are not present in $G - S$. This is implied by the properness of the coloring, as all such edges connect vertices of the same type, but for $xy \in E(F)$ the types of the vertices in $C_x \setminus S$ and $C_y \setminus S$ differ. □

We present an FPT algorithm for Cleaning(*3-Connected-Planar, Planar*) where the parameter is $k = |V(G)| - |V(H)|$ for input (H, G). We assume $n = |V(H)| > k + 2$ and $n \geq 4$ as otherwise we can solve the problem by brute force. We also assume that H and G are simple graphs.

Let S be a solution. First observe that if C is a set of at most 2 vertices such that $G - C$ is not connected, then there is a component K of $G - C$ such that the 3-connected graph $G - S$ is contained in $G[V(K) \cup C]$. Clearly, $|V(K)| \geq n - 2$, so K is unique by $n > k + 2$. Since such a separating set C can be found in linear time [17], K can also be found in linear time. If no component of $G - C$ has size at least $n - 2$, the algorithm outputs 'No', otherwise it proceeds with $G[V(K) \cup C]$ as input.

So we can assume that G is 3-connected. First the algorithm determines a planar embedding of H and G. Every planar embedding determines a circular order of the edges incident to a given vertex. Two embeddings are equivalent, if these orderings are the same for each vertex in both of the embeddings. It is well-known that a 3-connected planar graph has exactly two planar embeddings, and these are reflections of each other (see e.g. [10]). Let us fix an arbitrary embedding θ of H. By the 3-connectivity of G, one of the two possible embeddings of G yields an embedding of $G - S$ that is equivalent to θ. The algorithm checks both possibilities. From now on, we regard H and G as plane graphs, and we are looking for an isomorphism φ from H into $G-S$ which preserves the embedding.

In a general step of the algorithm, we grow a partial mapping, which is a restriction of φ. We assume that φ is already determined on a connected subgraph D of H having at least one edge. The definition of D implies $\varphi(V(D)) \cap S = \emptyset$, so if at some point the algorithm would have to delete vertices from $\varphi(D)$, it outputs 'No'.

The algorithm grows the subgraph D on which φ is determined step by step. At each step, it chooses an outer edge e of (D, H), and either deletes some vertices of $G - \varphi(D)$ that must be in S, or adds to D an outer face F of e w.r.t. (D, H). The algorithm chooses e and F in a way such that after the first step the following property will always hold:

Invariant 1: the outer edges of (D, H) form a cycle.

We refer to this as choosing a suitable face. Formally, a face F is *suitable* for (D, H) if it is an outer face w.r.t. (D, H) and Invariant 1 holds after adding F to D. Lemma 1 argues that a suitable face can always be found. We will see that the algorithm can only add a face F to D if $\varphi(F)$ is a face of G as well (that is, the interior of $\varphi(F)$ does not contain vertices from S). Hence, this method ensures that all vertices of $\varphi(V(H - D))$ and S are embedded on the same side of the border of $\varphi(D)$, allowing us to assume the following:

Invariant 2: the vertices of $V(G) \setminus \varphi(V(D))$ are embedded in the unique unbounded region determined by the border of $\varphi(D)$ in G.

The most important consequence of Invariant 2 is that φ yields a bijection between the outer edges of (D, H) and the outer edges of $(\varphi(D), G)$.

Lemma 1. *If D is a subgraph of a 3-connected graph H such that $|V(D)| < |V(H)|$ and the border of D in H is a cycle C, then there exists a suitable face for (D, H).*

Proof. By $|V(D)| < |V(H)|$, each edge of C has an outer face w.r.t. (D, H). The planarity of H implies that if a, b, c, d are four vertices appearing in this order on C, then there cannot exist two outer faces F_1, F_2 of (D, H) such that F_1 contains a and c, and F_2 contains b and d. Given an outer face F of (D, H), let the *gap* of F be the maximum length of a subpath of C whose endpoints are in $V(F)$ but has no internal vertices in $V(F)$.

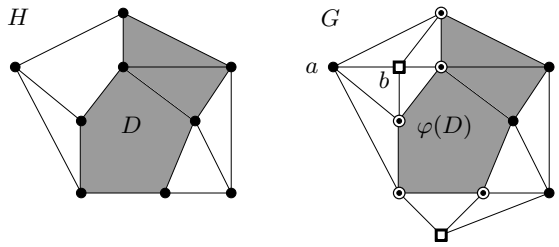

Fig. 2. Common neighbors test. Vertices of M are indicated by double circles, vertices of S by squares. By Lemma 2, we obtain that $b \in S$ but $a \notin S$.

Now, consider an outer face F^* of (D, H) that has minimum gap. If the gap of F^* is at least 2, then there is a subpath Q of C having at least two edges such that $V(F^*) \cap V(C)$ contains exactly the endpoints of Q. Consider any outer face F_Q of (D, H) that is incident to an edge of Q. By the observation of the previous paragraph, such a face cannot be incident to a vertex of C that is not in Q. Thus, F_Q must have smaller gap than F^*, which contradicts to the minimality of F^*. Therefore, F^* must have gap 1. Hence, the vertices of $V(F^*) \cap V(C)$ are consecutive vertices of C, implying that F^* is suitable. □

To find an initial partial mapping, we try to find a pair of edges ab and $a'b'$ in H and G, respectively, such that $\varphi(a) = a'$ and $\varphi(b) = b'$. To do that, the algorithm fixes an arbitrary edge ab in H and guesses $\varphi(a)$ and $\varphi(b)$. This yields $2|E(G)|$ possibilities. After this, the algorithm applies one of the following steps.

3-connectivity test. Although in the beginning G is assumed to be 3-connected, the algorithm may delete vertices from G throughout its running, and thus it can happen that G ceases to be 3-connected. This can be handled as described above, by finding a separating set C of size at most 2, and determining the component K of $G-C$ with at least $|V(H)|-2$ vertices. If no such component exists, or if it does not include $\varphi(D)$, then the algorithm outputs 'No', otherwise it deletes $V(G - C - K)$.

Common neighbors test. Let $M = \{\varphi(v) | v \in V(D), d_H(v) < d_G(\varphi(v))\}$. First, note that every vertex in M must have a neighbor in S, thus if $|M| > 2k$, then some vertex in S is adjacent to at least three vertices in M. By Invariant 2, the vertices of $S \subseteq V(G) \setminus \varphi(V(D))$ are embedded in the unbounded region determined by the border of $\varphi(D)$ in G, the vertices of M lie on this border. The algorithm checks every vertex $q \in V(G) \setminus \varphi(V(D))$ having at least three neighbors on the border of $\varphi(D)$, and determines whether $q \in S$, using Lemma 2 below. If no such vertex of S can be found in spite of $|M| > 2k$, then the algorithm outputs 'No'. Figure 2 shows an example.

Lemma 2. *Let $q \in V(G) \setminus \varphi(V(D))$ be adjacent to different vertices x, y and z on the border of $\varphi(D)$ in G. Then $q \in S$ if and only if there is no vertex $p \in V(H) \setminus V(D)$ which is a common neighbor of $\varphi^{-1}(x), \varphi^{-1}(y)$ and $\varphi^{-1}(z)$.*

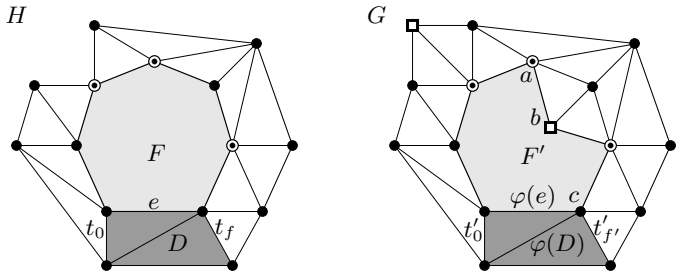

Fig. 3. Examining an outer face. Vertices of $\{t_j | j \in R\}$ and $\{t'_j | j \in R\}$ are indicated by double circles, vertices of S by squares. By Lemma 3, we obtain that $\{a, b, c\} \cap S \neq \emptyset$. Note that in general F might have more than one common edges with D.

Proof. For contradiction, let us assume $q \in S$ and suppose that a vertex p exist as described. As Invariant 1 holds for (D, H) and φ preserves the embedding, the outer edges of $(\varphi(D), G)$ and the edges $\varphi(p)x, \varphi(p)y$ and $\varphi(p)z$ cut the plane into four regions, and the only region among these containing all three of x, y and z is the bounded region determined by the outer edges of $(\varphi(D), G)$. But as no vertex in S can be embedded in this region by Invariant 2, q cannot be adjacent to all of x, y and z, a contradiction. On the other hand, if there is no vertex in $V(H) \setminus V(D)$ adjacent to $\varphi^{-1}(x), \varphi^{-1}(y)$ and $\varphi^{-1}(z)$, then $q \in S$ is trivial. □

Examining an outer face. In this step, the algorithm takes an outer edge $e = xy$ of (D, H) with a suitable outer face F in H, and the corresponding outer face F' of $\varphi(e)$ w.r.t. $(\varphi(D), G)$. Lemma 1 shows that a suitable face can always be found. If the algorithm finds that $V(F') \cap S = \emptyset$ must hold because of a sufficient condition given in Lemma 3 below, then it extends φ by adding F to D. Otherwise, $V(F')$ may contain vertices in S, so the algorithm branches into a bounded number of directions.

In the branch assuming $V(F') \cap S = \emptyset$, the extension of φ is performed. In the branches when $V(F') \cap S \neq \emptyset$ is assumed, the algorithm tries to find and delete the first vertex s on the border of F' in S, and branches according to the choice of s. Lemma 3 bounds the possibilities to choose s.

Intuitively, if there is a vertex of S on the border of F', then its deletion decreases the degree of at least two other vertices in $V(F')$. Also, the 3-connectedness of $G - S$ implies that deleting a vertex of S not in $V(F')$ can decrease the degree of at most two vertices on the border of F'. Lemma 3 states the consequences of these observations in a precise manner. See Fig. 3 for an illustration.

Lemma 3. *Let $e = t_0 t_f$ be an outer edge of (D, H) and F its outer face w.r.t. (D, H) such that its vertices in clockwise ordering are $t_0, t_1, \ldots, t_f$. Similarly, let F' be the outer face of $\varphi(e)$ w.r.t. $(\varphi(D), G)$, where the vertices of F' in clockwise*

ordering are $t'_0 = \varphi(t_0), t'_1, \ldots, t'_{f'-1}$ *and* $t'_{f'} = \varphi(t_f)$. *Let also* $R = \{j \mid 0 \leq j \leq \min(f, f'), d_H(t_j) \neq d_G(t'_j)\}$ *and let the indices in* R *be* $r_1 < \ldots < r_{|R|}$.

(1) If $|R| \leq 1$ *and* $f = f'$, *then* $V(F') \cap S = \emptyset$ *and* $\varphi(t_i) = t'_i$ *for every* $i \in [f]$.
(2) If $V(F') \cap S \neq \emptyset$ *and* t'_{i^*} *is the first vertex on the border of* F' *that is in* S, *then* $i^* - 1 \in \{r_j \mid j \in [\min(|R|, 2k+1)]\}$.

Proof. Let $e_0 = t_0 t_f$ and let $e_i = t_i t_{i-1}$ for every $i \in [f]$, so e_i is followed by e_{i-1} in the clockwise circular order of the edges incident to t_i for every $i \in [f]$. Similarly, we define $e'_i = t'_i t'_{i-1}$ for every $i \in [f']$. Supposing $V(F') \cap S = \emptyset$, we prove by induction that $\varphi(t_i) = t'_i$ for every $i \in \{0, 1, \ldots, f\}$. This is clear for $i = 0$, so assume that it holds for each index smaller than i. Since φ is an isomorphism, we have $\varphi(e_{i-1}) = e'_{i-1}$. Now, $e'_i \in E(G - S)$ implies $\varphi(e_i) = e'_i$ as well, since φ preserves the embedding. This proves the claim.

Now, if $V(F') \cap S \neq \emptyset$ and t'_{i^*} is the first vertex on the border of F' that is in S, then the vertices $t'_0, \ldots, t'_{i^*-1}$ are not in S, so by applying the above argument we get $\varphi(t_\ell) = t'_\ell$ for all $\ell < i^*$. But t'_{i^*-1} has a neighbor in S, hence $d_G(t'_{i^*-1}) > d_{G-S}(t'_{i^*-1}) = d_{G-S}(\varphi(t_{i^*-1})) = d_H(t_{i^*-1})$. This implies $i^*-1 \in R$. Letting j^* be the last vertex on the border of F' that is in S, and using $f = f'$ and the same argument as above, we get $j^* + 1 \in R$. Clearly $i^* - 1 < j^* + 1$, so $V(F') \cap S \neq \emptyset$ would imply $|R| \geq 2$. Hence, the conditions of (1) imply $V(F') \cap S = \emptyset$, proving also $\varphi(t_i) = t'_i$ for every $i \in [f]$.

To prove (2), suppose $V(F') \cap S \neq \emptyset$. As $i^* - 1 \in R$, $i^* - 1 = r_{\ell^*}$ for some ℓ^*. We claim $\ell^* \leq 2k+1$, which clearly implies $i^*-1 \in \{r_j \mid j \in [\min(|R|, 2k+1)]\}$. To see the claim, first observe that the definition of i^* implies $S \cap \{t_i | i < i^*\} = \emptyset$, so if $\ell < \ell^*$ then $\varphi(t_{r_\ell}) = t'_{r_\ell}$. But from the definition of R we know $d_H(t_{r_\ell}) \neq d_G(t'_{r_\ell})$, so we get that t'_{r_ℓ} must be adjacent to a vertex $s \in S$. As $r_\ell < i^* - 1$, this vertex s cannot be in the region of G corresponding to the face F of H. Note that in a 3-connected graph with at least four vertices no three vertices on the border of a single face can also lie on the border of another face, so no three vertices in $V(F')$ can be adjacent to the same $s \in S$. Therefore, each vertex of s can be adjacent to at most two vertices from $\{t_{r_\ell} | \ell \in [\ell^* - 1]\}$, so we obtain $\ell^* - 1 \leq 2|S| = 2k$. □

Now let us describe the key mechanism of our algorithm. The essential work is done by a recursive algorithm that we call *GrowSolution*, described in Fig. 4. The input of *GrowSolution* is a 4-tuple (H, G, D, φ), where H and G are plane graphs, H is 3-connected, D is a subgraph of H which is either an edge (in the first step) or the union of faces whose border in H is a cycle (Invariant 1), and φ is an embedding preserving isomorphism from D to an induced subgraph of G, such that the vertices of $V(G) \setminus \varphi(V(D))$ are embedded in the unique unbounded region determined by the border of $\varphi(D)$ in G (Invariant 2). The algorithm finds out whether there is an $S \subseteq V(G) \setminus \varphi(V(D))$ such that φ can be extended to map H to $G - S$ while remaining an isomorphism that preserves embedding. In each call, *GrowSolution* may stop or branch into a few directions. According to this, we will speak of *terminal* and *branching calls*. In each branch of a branching call, *GrowSolution* either deletes a vertex from G, or extends φ by adding a new face to D. If at the end of a branch a vertex is deleted, then this is a *deletion*

GrowSolution(H, G, D, φ)

1. If $k = |V(G)| - |V(H)| < 0$ then output('No').
2. Perform the 3-connectivity test.
3. If D equals H then output('Yes').
4. Perform the common neighbors test.
5. Examine an outer face. If for the chosen pair (F, F') of faces $|V(F)| = |V(F')|$ and $|R| \leq 1$, then extend φ on F, and go to Step 3. Otherwise branch as follows:
   - for all $j \in [2k+1]$: let $i^* = r_j + 1$ and call *GrowSolution*$(H, G - t'_{i^*}, D, \varphi)$.
   - if $|V(F)| = |V(F')|$ then extend φ on F and call *GrowSolution*(H, G, D, φ).

Fig. 4. The algorithm *GrowSolution*.

branch, otherwise it is an *extension branch*. (Actually, the algorithm may extend φ also in the deletion branches before performing the deletion.) At the end of each branch, *GrowSolution* calls itself recursively with the modified input.

In a single call, the algorithm first checks whether $|V(G)| < |V(H)|$, and if so, then correctly outputs 'No'. Next, it handles the case when G is not 3-connected. If D equals H, then Step 3 outputs 'Yes'. Note that this step allows $V(G) \setminus \varphi(V(H)) = S \neq \emptyset$ as well. To proceed, the algorithm searches for common neighbors, as described above. Recall that the algorithm might output 'No' or delete vertices from G at this step according to Lemma 2. If the algorithm deletes vertices $S' \subseteq S$ in Step 2 or Step 4, then this means that it calls *GrowSolution*$(H, G - S', D, \varphi)$. Now, if the algorithm does not stop or delete vertices, it examines an outer face. If for the chosen pair of faces (F, F') the conditions of (1) in Lemma 3 are fulfilled, then we know $V(F') \cap S = \emptyset$, so the algorithm proceeds by extending φ on F according to the lemma. When *GrowSolution* performs this extension, it also adds F to D, and checks whether φ is still an isomorphism on D, and if not, outputs 'No'. This is correct by Lemma 3. This extension step is iterated until either a vertex is deleted or the algorithm stops in Step 3, 4 or 5, or the conditions of (1) in Lemma 3 do not hold.

In the last case, we don't know whether $V(F') \cap S$ is empty or not, so the algorithm branches into at most $2k+2$ directions. First we assume $V(F') \cap S \neq \emptyset$, in this case statement (2) of Lemma 3 implies that $i^* \in \{r_j + 1 \,|\, j \in [\min(2k+1, |R|)]\}$ where t'_{i^*} is the first vertex on the border of F' being in S. The algorithm branches on these at most $2k+1$ possibilities to delete t'_{i^*}. The last branch is an extension branch corresponding to the case $V(F') \cap S = \emptyset$. Here, *GrowSolution* performs the extension of φ on F as described above. Note that this branch is only necessary if $|V(F)| = |V(F')|$.

Observe that the correctness of the algorithm directly follows from Lemmas 2 and 3. Although *GrowSolution* only answers the decision problem, it is straightforward to modify it in order to output the set S and the mapping φ.

To analyze the running time of the algorithm, we assign a search tree $T(I)$ to a run of *GrowSolution* with a given input I. The nodes of this tree correspond

to the calls of *GrowSolution*. The leaves represent the terminal calls and the internal nodes represent branching calls. The edge(s) leaving a node represent the branch(es) of the corresponding call of *GrowSolution*, so e heads from x to y if y is called in the branch represented by e in the call corresponding to x. The parameter of a node with input $I = (H, G, D, \varphi)$ is $k_I = |V(G)| - |V(H)|$. The parameter clearly decreases in a deletion branch, which cannot happen more than $k + 1$ times. However, in the extension branches this is not true, which seems to make it problematic to bound the size of the search tree. The following lemma shows that this problem does not arise, thanks to Step 4 of the algorithm.

Lemma 4. *The size of $T(I)$ is bounded by a function $f(k)$ where $k = k_I$.*

Proof. Let E^* denote the edges in $T(I)$ that correspond to extension branches. The value of the parameter decreases in each deletion branch, and it can only be negative in a leaf. Thus a path P leading from the root to a leaf in $T(I)$ can include at most $k + 1$ edges which are not in E^*. Let $Q = v_0v_1 \dots v_q$ be a subpath of P containing only edges in E^*.

First, we observe the fact that given a set L of vertices in a simple 3-connected planar graph G and a set $\mathcal{F}$ of faces each having at least 2 vertices from L on their border, we have $|\mathcal{F}| \leq \max\{6|L| - 12, 2\}$. To see this, we define the planar graph G' such that $V(G') = L$ and for each face $F \in \mathcal{F}$ there is an edge in G' connecting two vertices in $V(F) \cap L$. It is easy to see observe that in a 3-connected simple graph each pair of vertices can lie on the border of at most two faces. As G is 3-connected, this implies that every edge in G' has multiplicity at most 2. Now, if $|V(G')| \geq 3$ then the planarity of G' yields $|E(G')| \leq 2(3|L|-6)$. On the other hand, if $|V(G')| \leq 2$ then $|E(G')| \leq 2$ is trivial. For each face in $\mathcal{F}$ we defined an edge in G', so $|\mathcal{F}| \leq |E(G')| \leq \max\{6|L| - 12, 2\}$.

For a node w representing a call with input (H, G, D, φ), we define $M(w)$ to be the set containing those vertices $\varphi(t)$ on the border of $\varphi(D)$ in G such that $d_H(t) < d_G(\varphi(t))$. As $M(v_i)$ can only decrease after the deletion of some vertices, we get $M(v_{i-1}) \subseteq M(v_i)$ for every $i \in [q]$. Observe that in Step 5 of the branch represented by the edge $v_{i-1}v_i$, a face is added to $\varphi(D)$ that has at least two vertices in $M(v_i) \subseteq M(v_q)$. This follows because the conditions of (1) in Lemma 3 cannot hold in this step, and so the set $R \subseteq M(v_i)$ in Step 5 has cardinality at least 2. By Step 4 of the algorithm, $|M(v_q)| \leq 2k$. As shown above, there can be at most $\max\{12k - 12, 2\} \leq 12k - 10$ faces in G that are adjacent to at least 2 vertices in $M(v_q)$, so the number of extensions branches in Q, i.e. the length of Q is at most $12k - 10$. This enables us to bound the length of P, which is at most $k + 1 + (k + 1)(12k - 10) < 13k^2$. As every node in $T(I)$ has at most $2k + 2$ children, there are at most $(2k + 2)^{13k^2-1}$ leaves in $T(I)$, so the number of nodes in $T(I)$ is at most $f(k) = 2(2k + 2)^{13k^2-1}$. □

By careful implementation, it can be ensured that the amount of work done when extending φ on a face F is linear in $|V(F)|$, as we only spend constant time at a given vertex. This implies that the consecutive iteration of Steps 3, 4, and 5 can be performed in a total of linear time in $|V(G)|$. As other steps also can

3-Connected Planar Cleaning (H, G)

1. Perform the 3-connectivity test.
2. Let H_θ denote an embedded version of H, and let G_{θ_1} and G_{θ_2} be the two possible embedded versions of G. For $i = 1, 2$ do:
   3. Let $xy \in E(H)$ be arbitrary. For all (a, b) where $ab \in E(G)$ do:
      4. Let $\varphi_{a,b}$ denote the function mapping x to a and y to b.
         Output('Yes') if *GrowSolution*$(H_\theta, G_{\theta_i}, xy, \varphi_{a,b})$ returns 'Yes'.
5. Output('No').

Fig. 5. The algorithm solving Cleaning(*3-Connected-Planar, Planar*).

be performed in time linear in $|V(G)|$, by Lemma 4 we can conclude that the running time of *GrowSolution* on input (H, G, D, φ) is $O(f(k)|V(G)|)$ for some function f, where $k = |V(G)| - |V(H)|$.

As a result, there is an algorithm that solves Cleaning(*3-Connected-Planar, Planar*) in FPT time. The steps of the decision version of this algorithm are described in Fig. 5. Its correctness easily follows from the discussion above. As it calls *GrowSolution* at most $4|E(G)| = O(n)$ times, we can conclude:

Theorem 2. *The* Cleaning(*3-Connected-Planar, Planar*) *problem on input* (H, G) *can be solved in time* $O(f(k)n^2)$*, where* $n = |V(H)|$ *and* $|V(G)| = n + k$.

4 The Cleaning(*Tree,*−) problem

The aim of this section is to present an FPT algorithm for Cleaning(*Tree,*−). We parameterize this problem by assigning the parameter $k = |V(G)| - |V(T)|$ for an input (T, G).

Note that if (T, G) is solvable, then the treewidth of G is at most $k+1$. However, this does not yield an obvious way to deal with the problem, as problems related to isomorphism typically remain hard for graphs of bounded treewidth. In particular, Induced Subgraph Isomorphism remains NP-hard when restricted to inputs (H, G) where H is a tree and G has treewidth 2 [22]. Although the restriction of Graph Isomorphism to graphs of treewidth at most k can be solved in $O(n^{k+4.5})$ [2], the parameterized complexity of this problem with k being the parameter is still unknown. Note also that since Cleaning(*Tree,*−) contains the Induced Path problem, the standard parametrization where the parameter is $|V(T)|$ yields a W[1]-complete problem [6].

W.l.o.g. we can assume that G is simple, $n = |V(T)| > k$ (otherwise we can solve the problem by a brute force algorithm) and $e = |E(G)| = O(kn)$ (as we can automatically refuse instances where $e > n - 1 + k(n + k - 1)$). Let S be a fixed solution, i.e. let $G - S = T_S$ be a tree isomorphic to T. Throughout the run of the algorithm, we can assume that G is connected, since by $n > k$ it is trivial to find the unique connected component of G containing T_S.

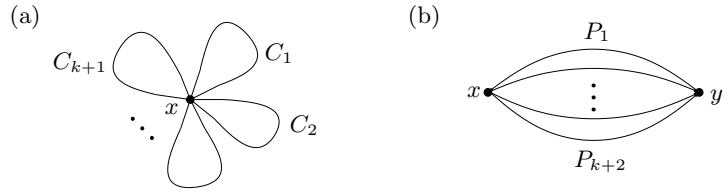

Fig. 6. Figures (a) and (b) illustrating Reductions $\mathcal{A}$ and $\mathcal{B}$, respectively.

4.1 Preprocessing

First, we introduce two kinds of reductions, each deleting some vertices from G which must be included in S. See Fig. 6 for an illustration.

Reduction $\mathcal{A}$: cycles with one common vertex. If for some vertex $x \in V(G)$ there exist cycles $C_1, \ldots, C_{k+1}$ in G such that $V(C_i) \cap V(C_j) = \{x\}$ for each pair $i \neq j$, then it is easy to see that x must be in S. Indeed, if $x \notin S$ then S must contain at least one vertex from each cycle C_i, implying $|S| \geq k+1$. Thus, we get $x \in S$.

It can be checked whether the above condition holds for some $x \in V(G)$ by a technique based on generalized matchings, used by Bodlaender in [4]. Using this method, we can reduce our problem to a b-matching problem in a graph having $O(|V(G)|)$ vertices and $O(|E(G)|)$ edges, with the degree constraints being at most $2k+2$ for each vertex and $O(|V(G)|)$ in total. Applying the algorithm of Gabow [14] for the obtained problem, we can solve it in $O(\sqrt{|V(G)|}|E(G)|) = O(\sqrt{n}e) = O(kn\sqrt{n})$ time. This means that Reduction $\mathcal{A}$ can be performed in $O(kn^{5/2})$ total time for all vertices of G.

Reduction $\mathcal{B}$: disjoint paths between two vertices. Let $x, y \in V(G)$ be vertices such that there exist paths $P_1, \ldots, P_{k+2}$ from x to y which are disjoint apart from their endpoints. Then x or y must be included in any solution S of size at most k, as assuming $x, y \notin S$ implies the existence of a cycle through x and y in $G - S$. Using standard flow techniques we can check in time $O(ke)$ whether (x, y) is such a pair of vertices, so finding such a pair takes time $O(ken^2) = O(k^2n^3)$. Given such a pair of vertices yields two possibilities for a reduction, so the algorithm branches in two directions. Since $|S| = k$, we can apply Reduction $\mathcal{B}$ at most k times, which means a total of at most 2^k branches.

Now denote by K the minimal connected subgraph of G containing every cycle of G. Note that K is unique, and is an induced subgraph of G. Let K_3 denote the vertices of K whose degree in K is at least 3.

Lemma 5. *If Reductions $\mathcal{A}$ and $\mathcal{B}$ cannot be applied, then $d_K(x) \leq k^2 + k$ for every $x \in V(K - S)$ and $|K_3| < g(k) = 2k^3(k+1) + 3k = O(k^4)$.*

Proof. Let us assume that $x \in V(K - S)$ has neighbors $v_1, v_2, \ldots, v_{k^2+k+1}$ in K. Observe that $K - S$ is a tree, and each of its leaves is adjacent to some vertex of S. Using this, we obtain that the edges xv_i (for $i \in [k^2 + k + 1]$) can be extended

to internally disjoint paths in K starting from x, ending in a vertex of S, and having internal vertices in $K-S$. We note that such a path can be of length 1. As $|S| \leq k$, there must exist a vertex $s \in S$ such that at least $\lceil (k^2+k+1)/k \rceil = k+2$ of these paths end in s. Now, these paths form at least $k+2$ internally disjoint paths between x and s, yielding a possibility for Reduction $\mathcal{B}$, a contradiction.

We claim that given a tree T' with maximum degree d and a set $Z \subseteq V(T')$ with cardinality at least $pd+2$, there always exists a set $\mathcal{P}$ of $p+1$ disjoint paths connecting vertices of Z. Let us regard T' as a rooted tree. We add paths to $\mathcal{P}$ in the following manner: we always choose a new path to put in $\mathcal{P}$ such that its distance from the root is the largest possible. When adding the path P to $\mathcal{P}$, we delete those vertices from Z that can no longer be connected to the root without crossing P. This ensures that connecting any two vertices of Z in T' always results in a path that can be added to $\mathcal{P}$. Note that by our choice on P and by the maximum degree d of T' we obtain that $|Z|$ can decrease at most by d in each step, except for the case when P contains the root and thus $|Z|$ might decrease by $d+1$. Therefore, we can indeed put $p+1$ paths into $\mathcal{P}$.

For a vertex $s \in S$, let T_s denote the unique minimal subtree of $K - S$ containing $Z_s = N_{V(K-S)}(s)$. Suppose $|Z_s| \geq k(k^2+k)+2$ for some s. As every vertex in T_s has maximum degree k^2+k by the first claim of the lemma, using the claim proved above we get that there are $k+1$ disjoint paths in T_s connecting vertices of Z_s. These paths together with s form $k+1$ cycles whose only common vertex is s, contradicting our assumption that Reduction $\mathcal{A}$ is not applicable.

Thus, we get $|Z_s| \leq k(k^2+k)+1 = k^2(k+1)+1$ for each $s \in S$. Let L denote the leaves of $K-S$. Every vertex in L has a neighbor in S, so $L \subseteq N_{V(K-S)}(S) = \bigcup_{s \in S} Z_s$, implying $|L| \leq |N_{V(K-S)}(S)| \leq k^3(k+1)+k$. Observe that every vertex in $K_3 \setminus (S \cup N_{V(K-S)}(S))$ has degree at least 3 also in $K - S$. Since the number of vertices in the tree $K - S$ having degree at least 3 is less than the number $|L|$ of leaves, we get $|K_3| < |S| + |N_{V(K-S)}(S)| + |L| \leq |S| + 2|N_{V(K-S)}(S)|$, implying $|K_3| < 2k^3(k+1) + 3k$. □

4.2 Growing a mapping

From now on, we assume that Reductions $\mathcal{A}$ and $\mathcal{B}$ cannot be applied. At this point, the algorithm checks whether the conditions of Lemma 5 are fulfilled, and correctly outputs 'No' if the conditions do not hold. Let ϕ denote the isomorphism from T to T_S that we are looking for. As in Sect. 3, we try to grow a partial mapping from T to T_S, which is always a restriction of ϕ. To begin, the algorithm chooses an arbitrary starting vertex r_0 in T, and branches on the choice of $\phi(r_0)$ in G, which means $|V(G)|$ possibilities.

Throughout its running, the algorithm may modify G by deleting vertices of S from it. We denote by G^i the graph obtained from G after the i-th step, with $G = G^0$. Assume that in the i-th step of the algorithm there is a subtree D^i of T on which ϕ is already known. The algorithm proceeds step by step, choosing a leaf r^i of D^i in the i-th step that has not been examined yet. For the chosen vertex r^i, it determines ϕ on $N_T(r^i)$ by applying a method described below. This means also that it adds $N_T(r^i)$ to D^i to get D^{i+1}, deletes $N_G(\phi(r^i)) \cap S$ from G^i

to get G^{i+1}, and checks whether ϕ is still an isomorphism. When determining ϕ on $N_T(r^i)$, the algorithm may branch into a bounded number of branches, or may proceed with a single branch. Accordingly, we distinguish between *branching* and *simple cases*.

Let us describe the details of a single step of the algorithm. First, it checks whether $|V(G^i)| \geq |V(T)|$ holds, outputting 'No' if the condition fails. Next, it verifies some simple conditions considering the neighbors of r^i and $\phi(r^i) = r'^i$. To do this, it determines the minimal connected subgraph K^i of G^i containing every cycle of G^i. Note that K^i can be constructed from G^i easily in linear time, as the 2-connected components of a graph can be determined in linear time, e.g. by applying depth first search [8].

To proceed, let us introduce some new notation. We divide the vertices of $N_T(r^i)$ into two groups as follows:

- those neighbors of r^i that are in D^i,
- those neighbors of r^i that are not in D^i. Let $t^i_1, \ldots, t^i_{\alpha^i}$ denote these vertices, and let T^i_j be the tree component of $T - r^i$ containing t^i_j.

Similarly, we classify the vertices of $N_G(r'^i)$ into three groups:

- those neighbors of r'^i that are in $\phi(D^i)$,
- those neighbors of r'^i outside $\phi(D^i)$ that are connected to r'^i by edges not in K^i. Let $t'^i_1, \ldots, t'^i_{\beta^i}$ denote these vertices, and T'^i_j denote the component of $G^i - r'^i$ that includes t'^i_j. Observe that either T'^i_j is a tree, or $r'^i \notin V(K^i)$ and T'^i_j contains K^i.
- those neighbors of r'^i outside $\phi(D^i)$ that are connected to r'^i by edges in K^i. Let γ^i be the number of such vertices.

Clearly, $\alpha^i \leq \beta^i + \gamma^i$, and the equality holds if and only if $N_{G^i}(r'^i) \cap S = \emptyset$. Thus, if the algorithm finds that $\alpha^i > \beta^i + \gamma^i$, then it outputs 'No'.

First, let us observe that if the tree T^i_h is isomorphic to T'^i_j for some h and j, then w.l.o.g. we can assume that $\phi(T^i_h) = T'^i_j$. As the trees of a forest can be classified into equivalence classes with respect to isomorphism in time linear in the size of the forest [1, 18], this case can be noticed easily. Given two isomorphic trees, an isomorphism between them can also be found in linear time, so the algorithm can extend ϕ on T^i_h, adding also T^i_h to the subgraph D^i. Hence, we only have to deal with the following case: no tree T^i_h ($h \in [\alpha^i]$) is isomorphic to one of the graphs T'^i_j ($j \in [\beta^i]$). This argument makes our situation significantly easier, since every graph T'^i_j must contain some vertex from S. Therefore $\beta^i \leq |S| = k$. Clearly, if $r'^i \notin V(K^i)$ then $\gamma^i = 0$. On the other hand, if $r'^i \in V(K^i)$ then r'^i can have degree at most $k^2 + k$ in K^0, and thus in K^i, by Lemma 5. Thus, we get $\gamma^i \leq k^2 + k$, implying also $\alpha^i \leq \beta^i + \gamma^i \leq k^2 + 2k$. The algorithm determines α^i, β^i and γ^i in each step, and outputs 'No' if these bounds do not hold for them.

The algorithm faces one of the following two cases at each step.

Simple case: $\beta^i + \gamma^i \leq 1$. In this case $\alpha^i \leq 1$. If $\beta^i + \gamma^i = 0$ then $\alpha^i = 0$, hence the algorithm proceeds with the next step by choosing another leaf of D^i not yet visited. Otherwise, let v be the unique vertex in $N_{G^i}(r'^i) \setminus V(\phi(D^i))$. If

$\alpha^i = 0$ then v must be in S, otherwise $\phi(t_1^i) = v$. According to this, the algorithm deletes v or extends ϕ on t_1^i, adding also t_1^i to D^i.

Branching case: $\beta^i + \gamma^i \geq 2$. In this case, the algorithm branches on every possible choice of determining ϕ on $N_T(r^i)$. Guessing $\phi(v)$ for a vertex $v \in N_{V(T-D^i)}(r^i)$ can result in at most $\beta^i + \gamma^i$ possibilities, so the number of possible branches in a branching step is at most $(\beta^i + \gamma^i)^{\alpha^i} \leq (k^2 + 2k)^{k^2+2k}$. After guessing $\phi(v)$ for each vertex $v \in N_{V(T-D^i)}(r^i)$, the algorithm puts the remaining vertices $N_G(r'^i) \setminus \{\phi(v) | v \in N_T(r^i)\}$ into S, deleting them from G^i.

Lemma 6. *In a single branch of a run of the algorithm described above on a solvable input for the* CLEANING(*Tree*,−) *problem with parameter k, there can be at most $g(k) + 2k - 2$ branching steps.*

Proof. We use the notation applied in the description of the algorithm. The i-th step can only be a branching case if either $\gamma^i \geq 2$, $\beta^i \geq 2$, or $\beta^i = \gamma^i = 1$ holds. For each of these cases, we give an upper bound on the number of steps in a single branch of a run of the algorithm where these cases can happen.

To determine a bound for the case $\gamma^i \geq 2$, let r^* be the first vertex in T examined in a step such that $\phi(r^*)$ is in K^0. Recall that $K^0 - S$ is a tree, so supposing $\phi(r^i) \in V(K^0)$ we get that if $r^i \neq r^*$ then for the edge e incident to r^i in D^i it must hold that $\phi(e)$ is in K^0. Now, observe that if $\gamma^i \geq 2$ holds, then this implies that either $r^i = r^*$ or $\phi(r^i)$ has at least three edges incident to it in K^0. The latter means that $r^i \in K_3$, where K_3 denotes the vertices of K^0 having degree at least three in K^0. Thus, the condition $\gamma^i \geq 2$ can hold in at most $|K_3| + 1 \leq g(k)$ steps, by Lemma 5.

On the other hand, if the algorithm finds that $\beta^i \geq 2$, then recall that both $T_1'^i$ and $T_2'^i$ include at least one vertex from S, and thus $G^i - \phi(D^i)$ has more connected components containing vertices of S than $G^i - \phi(D^i - r^i)$ has. It is easy to see that this can be true for only at most $|S| - 1$ such vertices r^i, so this case can happen at most $k-1$ times in a single branch of a run of the algorithm.

Finally, let S^* denote those vertices of S that are not contained in K^0. Clearly, if $s \in S^*$, then s is not contained in any cycle of G, so $|N_G(s) \cap V(T_S)| \leq 1$. Now, if $\beta^i = \gamma^i = 1$, then $r'^i \in V(K)$ and the edge $r'^i t_1'^i$ must be one of the edges that connect to K^0 a tree in $G - K^0$ containing a vertex in S^*. Observe that there can be at most $|S^*| \leq k - 1$ such edges. Therefore, the claim follows. □

As Lemma 6 only bounds the number of branching steps for solvable inputs, the algorithm ensures the same bound on every input by maintaining a counter for these steps. Thus, it outputs 'No' if it encounters a branching case for the $(g(k) + 2k - 1)$-th time.

At each vertex the algorithm uses time at most linear in $|V(G)|$. The number of steps performed is at most $|V(T)|$. As both the number of branches in a branching case and the number of branching cases in a single branch of a run of the algorithm is bounded by a function of k, the algorithm needs quadratic time after choosing $\phi(r_0)$ for the starting vertex r_0. Trying all possibilities on $\phi(r_0)$ increases this to cubic time. Reductions $\mathcal{A}$ and $\mathcal{B}$ can also be executed in cubic time, as argued before, so we can conclude:

Theorem 3. *The* CLEANING(*Tree*,−) *problem on input* (T, G) *can be solved in time* $O(f(k)n^3)$*, where* $n = |V(T)|$ *and* $|V(G)| = n + k$*.*

5 Conclusions and open problems

We have investigated the complexity of the parameterized CLEANING$(\mathcal{H}, \mathcal{G})$ problem, where given two graphs $H \in \mathcal{H}$ and $G \in \mathcal{G}$ with the parameter $k = |V(G)| - |V(H)|$, the task is to find k vertices in G whose deletion results in a graph isomorphic to H. We proved NP-completeness for the unparameterized version of the CLEANING problem in the case when both graphs are 3-connected planar graphs, and we presented FPT algorithms for the CLEANING(*Tree*,−) and CLEANING(*3-Connected-Planar*, *Planar*) problems.

A natural question is whether the parameterized CLEANING(−, *Planar*) problem, or equivalently the CLEANING(*Planar*, *Planar*) problem, is FPT with parameter $|V(G)|-|V(H)|$. It would be also interesting to determine the complexity of the CLEANING problem for other graph classes for which the GRAPH ISOMORPHISM problem can be solved in polynomial time, such as the class of interval graphs.

Another way of generalization is to extend the set of allowed operations which can be used to clean the larger input graph in order to obtain the smaller one. Such operations could involve the deletion of edges, or contraction of vertices or edges. By allowing the deletion of a certain number of vertices to be applied on the smaller graph also we obtain the MAXIMUM COMMON INDUCED SUBGRAPH problem [19]. To our knowledge, none of these natural questions have been studied using the non-standard parametrization examined in this paper.

Acknowledgment

We would like to thank an anonymous referee for pointing out an error in a previous version of the paper.

References

1. A. V. Aho, J. E. Hopcroft, J. D. Ullman: *The design and analysis of computer algorithms*, Addison-Wesley, 1974.
2. H. L. Bodlaender: Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees. *J. Algorithms*, 11:631–643, 1990.
3. H. L. Bodlaender: On disjoint cycles. *Int. J. Found. Comput. Sci.*, 5:59–68, 1994.
4. H. L. Bodlaender: A cubic kernel for feedback vertex set. *STACS 2007, LNCS 4393*, 320–331, 2007.
5. L. Cai, S. M. Chan, S. O. Chan: Random separation: a new method for solving fixed-cardinality optimization problems. *IWPEC 2006, LNCS 4169*, 239–250, 2006.
6. Y. Chen, J. Flum: On parameterized path and chordless path problems. *22nd Annual IEEE Conference on Computational Complexity*, 250–263, 2007.

7. Y. Chen, M. Thurley, W. Weyer: Understanding the complexity of induced subgraph isomorphisms. *ICALP 2008, LNCS 5125*, 587–596, 2008.
8. T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein: *Introduction to Algorithms (Second Edition)*, MIT Press, 2001.
9. J. Díaz, D. M. Thilikos: Fast FPT-algorithms for cleaning grids. *STACS 2006, LNCS 3884*, 361–371, 2006.
10. R. Diestel: *Graph Theory*, Springer, Berlin, 2000.
11. R. G. Downey, M. R. Fellows: *Parameterized Complexity*, Springer-Verlag, New York, 1999. Springer, 1999.
12. D. Eppstein: Subgraph isomorphism in planar graphs and related problems. *J. Graph Algorithms Appl.* 3(3):1–27, 1999.
13. J. Flum, M. Grohe: *Parameterized Complexity Theory*, Springer-Verlag, Berlin, 2006. Springer, 2006.
14. H. N. Gabow: An Efficient Reduction Technique for Degree-Constrained Subgraph and Bidirected Network Flow Problems. *Proc. 15th Annual ACM Symp. on Theory of Comp.*, 448–456, 1983.
15. M. R. Garey, D. S. Johnson: *Computers and Intractability. A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979.
16. M. T. Hajiaghayi, N. Nishimura: Subgraph isomorphism, log-bounded fragmentation and graphs of (locally) bounded treewidth. *J. Comput. Syst. Sci.* 73(5):755–768, 2007.
17. J. E. Hopcroft, R. E. Tarjan: Dividing a graph into triconnected components. *SIAM J. Computing* 2(3):135–158, 1973.
18. J. E. Hopcroft, R. E. Tarjan: Efficient planarity testing. *J. Assoc. Comput. Mach.* 21:549–568, 1974.
19. X. Huang, J. Lai: Maximum common subgraph: upper bound and lower bound results. *IMSCCS 2006*, 1:40–47
20. A. Lingas: Subgraph isomorphism for biconnected outerplanar graphs in cubic time. *Theoret. Comput. Sci.* 63(3):295–302, 1989.
21. D. Marx, I. Schlotter: Obtaining a planar graph by vertex deletion. *WG 2007, LNCS 4769*, 292–303, 2007.
22. J. Matoušek, R. Thomas: On the complexity of finding iso- and other morphisms for partial k-trees. *Discrete Math.* 108:343–364, 1992.
23. D. Matula: Subtree isomorphism in $O(n^{5/2})$. *Ann. Discrete Math.* 2:91–106, 1978.