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.

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 Independent Set or Longest 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 [16] or 2-connected outerplanar graphs [14]. 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 [10]. In many fields where researchers face hard problems, parameterized complexity theory (see e.g. [7] or [9]) has proved to be successful in the analysis and design of algorithms that have a tractable running time in many applications. In parameterized complexity, a parameter k is introduced besides 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.

Note that Induced Subgraph Isomorphism is trivially solvable in time $O(|V(G)|^{|V(H)|}|E(H)|)$ on input graphs H and G. As H is typically much smaller

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

than G in many applications related to pattern matching, the usual parametrization of INDUCED SUBGRAPH ISOMORPHISM is to define the parameter to be $|V(H)|$. FPT algorithms are known if G is planar [8], has bounded degree [3], or if H is a log-bounded fragmentation graph and G has bounded treewidth [11].

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 $|V(G)| - |V(H)|$. 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 [2, 15].

Without parametrization, CLEANING(*Tree*,$-$) is NP-hard because it contains LONGEST 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*) [16], and an FPT algorithm is known for CLEANING(*Grid*,$-$) where *Grid* is the class of rectangular grids [5].

2 Notation

We write $[n]$ for $\{1, \ldots, n\}$. 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 $G - H = G - V(H)$. By contracting a vertex of degree 2, we mean deleting it and adding an edge between its neighbors.

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). An *isomorphism* from H into G is a bijection $\varphi : V(H) \cup E(H) \rightarrow V(G) \cup E(G)$ preserving incidency. For a subgraph H' of H, $\varphi(H')$ 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 [10]. Let F be the planar input graph given. 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 by taking vertices $a^x, b_1^x, \ldots, b_{6b(x)}^x$ and $c_1^x, \ldots, c_{3b(x)}^x$ and edge set $\{a^x b_j^x, b_j^x b_{j+1}^x \mid j \in [6b(x)]\} \cup \{c_j^x b_{2j-1}^x, c_j^x b_{2j}^x, c_j^x b_{2j+1}^x | j \in [3b(x)]\}$ where $b_{6b(x)+1}^x = b_1^x$. The node-gadget N_x can be considered as a plane graph, supposing that the vertices $b_1^x, b_2^x, \ldots, b_{6b(x)}^x$ (and so $c_1^x, c_2^x, \ldots, c_{3b(x)}^x$) are embedded in a clockwise order around a^x. We define the j-th *block* B_j^x of N_x to be $(c_{3j-2}^x, c_{3j-1}^x, c_{3j}^x)$, for every $j \in [b(x)]$. The *type* of c_j^x can be 0, 1 or 2, according to the value of j modulo 3. We set $C_x = \{c_j^x | j \in [3b(x)]\}$.

For each edge $xy \in E(F)$ we build a *connection* E_{xy} in G that uses 9-9 consecutive blocks from N_x and N_y, say $B_i^x, \ldots, B_{i+8}^x$ and $B_j^y, \ldots, B_{j+8}^y$. 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 block is a base block for different connections. To build E_{xy} with $b(x, y) = (i, j)$, we introduce three new vertices $d_1^{xy}, d_2^{xy}, d_3^{xy}$ and edges $\{c_{3(i+8)-6m+\ell}^x d_m^{xy}, c_{3j-2+6m-\ell}^y d_m^{xy} | m \in [3], \ell \in [6]\} \cup \{c_{3i+6}^x c_{3j+18}^y, c_{3i+2}^x c_{3j+20}^y, c_{3i-2}^x c_{3j+22}^y\}$ (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_j^x (a^x, C_x, etc.) we write $\bar{c}_j^x$ ($\bar{a}^x$, $\bar{C}^x$, 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 $x \neq y$, and vertices $\bar{c}_{3j-2}^x$ and $\bar{c}_{3j-1}^x$ for every $x \in V(F), j \in [b(x)]$. 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 $c_{3i+6}^x c_{3j+18}^y, c_{3i+2}^x c_{3j+20}^y, c_{3i-2}^x c_{3j+22}^y$ would be

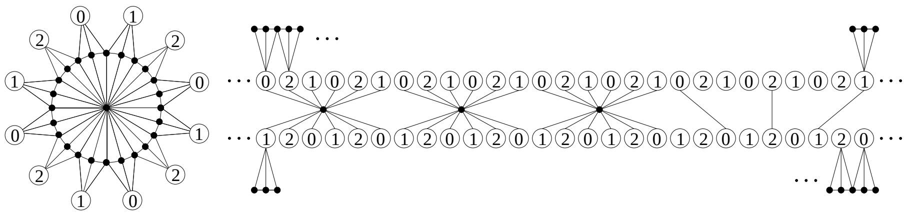
Fig. 1. A node-gadget and a connection for the proof of Theorem 1.

in $G - S$ 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 $t : V(F) \to \{1, 2, 3\}$ be a coloring of F. For each $x \in V(F)$, let S contain those vertices in C_x whose type is not $t(x)$ modulo 3. 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 $c^x_{j+t(x)-3}$ (in a cyclic order). 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 $c^x_{3j+t(x)-3} = \varphi(\bar{c}^x_{3j})$, 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 $n = |V(H)| > k + 2$ 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 $G - S$ is contained in $G[V(K) \cup C]$. Clearly, K has size at least $n - 2$, and it is unique by $n > k + 2$. Since such a separating set of size at most 2 can be found in linear time [12], K can also be found in linear time. If no component of $G - C$ has size at least $n - 2$, then 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. [6]). 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 subgraph D of H having at least one edge, such that the vertices of $H - D$ are embedded in the unbounded face of D. As implied implicitly, $\varphi(V(D)) \cap S = \emptyset$, 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 $G - \varphi(D)$ or adds to D an outer face F of e w.r.t. (D, H). This implies that the outer edges of (D, H) correspond to the outer edges of $(\varphi(D), G)$. Moreover, the algorithm chooses e and F in a way such that after the first step it will always hold that the outer edges of (D, H) form a cycle. We refer to this as choosing an *appropriate* face. This method ensures that every vertex in $V(H) \setminus V(D)$ is embedded in the unique unbounded region determined by the border of D in H. (The *border* of D in H is the subgraph formed by the outer edges of (D, H)). Note that it also follows 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.

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. As we can delete vertices from G, it may 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. As 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 having at least three neighbors on the border of $\varphi(D)$ in G, and determines whether $q \in S$, using Lemma 1. If no such vertex of S can be found in spite of $|M| > 2k$, then the algorithm outputs 'No'.

Lemma 1. *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 $\varphi^{-1}(z)$.*

Proof. For contradiction, let us assume $q \in S$ and let a vertex p exist as described. As D is connected 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 our assumption on D), 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. $\square$

Examining an outer face. In this step, the algorithm takes an outer edge $e = xy$ of (D, H) with an appropriate outer face F in H, and the corresponding outer face F' of $\varphi(e)$ w.r.t. $(\varphi(D), G)$. If the algorithm finds that $V(F') \cap S = \emptyset$ must hold because of a sufficient condition given in Lemma 2, 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 2 bounds the possibilities to choose s.

Lemma 2. *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 $t_0, t_1, \dots, t_f$. Similarly, let F' be an outer face of $\varphi(e)$ w.r.t. $(\varphi(D), G)$, where the vertices of F' in clockwise ordering are $t'_0 = \varphi(t_0), t'_1, \dots, t'_{f'-1}$ and $t'_{f'} = \varphi(t_f)$. Let also $R = \{j \in [\min(f, f')] \mid d_H(t_j) \neq d_G(t'_j)\}$ and let the indices in R be $r_1 < \dots < r_{|R|}$.*

(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 $e_i = t_i t_{i-1}$ for every $i \in [f]$, so e_{i+1} is followed by e_i in the clockwise circular order of the edges incident to t_i. Now, if e'_{i+1} is followed by $\varphi(e_i)$ in the clockwise circular order of the edges incident to $\varphi(v_i)$ and $e'_{i+1} \in E(G - S)$, then $\varphi(e_{i+1}) = e'_{i+1}$ as φ preserves the embedding. Thus if $V(F') \cap S = \emptyset$, then applying this argument iteratively, from $\varphi(t_0 t_f) = t'_0 t'_{f'}$ we can deduce $\varphi(t_i) = t'_i$ for every $i \in [f]$.

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 $t'_0, \dots, t'_{i^*-1}$ 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^* to 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 $i^* - 1 \in R$, if ℓ^* is the last index in R such that for any $\ell \leq r_{\ell^*} + 1$ the vertex t'_ℓ is not in S, we get $i^* - 1 = r_{\ell^*+1}$. We claim $\ell^* \leq 2k$, which clearly implies $i^* - 1 \in \{r_j \mid j \in [\min(|R|, 2k + 1)]\}$. To see the claim, suppose $\ell \leq \ell^*$. Since $d_H(t_{r_\ell}) \neq d_G(t'_{r_\ell})$ but $\varphi(t_{r_\ell}) = t'_{r_\ell}$, we get that t'_ℓ is adjacent to a vertex $s \in S$, and by the definition of ℓ^* we know that $s \notin V(F')$, so s is not in the region of G corresponding to the face F of H. Note that in a 3-connected graph 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 $s \in S$. Using this we obtain $\ell^* \leq 2|S| = 2k$. $\square$

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. 2.

```
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 $\varphi$ 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 $\varphi$ on $F$ and call GrowSolution$(H, G, D, \varphi)$.
```

Fig. 2. The algorithm *GrowSolution*.

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, and φ is an embedding preserving isomorphism from D to an induced subgraph of G, such that the border of $\varphi(D)$ in G is also a cycle. The algorithm finds out whether there is an $S \subseteq V(G)$ 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 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'. Then it searches for common neighbors, as described above. 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 2 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 2. 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 2 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 2 implies that $i^* \in \{r_j + 1 \mid j \in [\min(2k+1, |R|)]\}$ 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 1 and 2. Although *GrowSolution* only answers the decision problem, it is straightforward to modify it in order to output the set S.

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 3. *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 $Q = v_0 v_1 \dots v_q$ 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 $|\mathcal{F}| \leq 6|L| - 12$. 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 $V(F) \cap L$. As G is 3-connected, every edge in G' has multiplicity at most 2, so the planarity of G' yields $|E(G')| \leq 2(3|L| - 6)$. For each face in $\mathcal{F}$ we defined an edge in G', so $|\mathcal{F}| \leq |E(G')| \leq 6|L| - 12$.

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 2 cannot hold in this step, and so the set $R \subseteq M(v_i)$ in Step 5 has cardinality least 2. By Step 4 of the algorithm, $|M(v_q)| \leq 2k$. As shown above, there can be at most $12k - 12$ 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 $12k - 12$. This enables us to bound the length of P, which is at most $k + 1 + (k + 1)(12k - 12) < 13k^2$. As every node in $T(I)$ has at most $2k + 2$ children, the number of nodes in $T(I)$ is at most $f(k) = (2k + 2)^{13k^2}$. □

By careful implementation, it can be assured that the amount of work done when extending φ on a face F is linear in $|V(F)|$, as we only spend constant time

```
3-Connected Planar Cleaning $(H, G)$

 1. Perform the 3-connectivity test.
 2. Let $H_\theta$ denote an embedded version of $H$, and let $G_{\theta_1}$ and $G_{\theta_2}$ 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. 3. The algorithm solving Cleaning(*3-Connected-Planar, Planar*).

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 be performed in time linear in $|V(G)|$, by Lemma 3 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. 3. Its correctness easily follows from the discussion above. As it calls *GrowSolution* at most $4|E(G)|$ 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*,−). Note that since Cleaning(*Tree*,−) contains the Longest Induced Path problem, the standard parametrization where the parameter is $|V(H)|$ yields a W[2]-hard problem [4].

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.

4.1 Preprocessing

First, we introduce two kinds of reductions, each deleting some vertices from G which must be included in S.

Reduction $\mathcal{A}$: cycles with one common vertex. If for some vertex $x \in V(G)$ there exist cycles $C_1, C_2, \ldots, C_{k+1}$ such that $V(C_i) \cap V(C_j) = \{x\}$ if $i \neq j$, then x must be included in any solution. To see this, observe that if x is not in the solution S, then S must contain at least one vertex from each cycle C_i, but this would imply $|S| \geq k+1$. For each x, we can find such cycles by solving a flow problem in an appropriately defined directed graph. Since we need to find flows with value at most $k+1$, this can be done in time $O(ke)$ for a single vertex x. This means that Reduction $\mathcal{A}$ can be performed in time $O(ken) = O(k^2n^2)$.

Reduction $\mathcal{B}$: disjoint paths between two vertices. Let $x, y \in V(G)$ be vertices such that there exist at least $k+2$ paths 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. We can construct K from G 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. Let K_3 denote the vertices of K whose degree in K is at least 3.

Lemma 4. *If Reduction $\mathcal{A}$ 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. Then the edges xv_i (for $i \in [k^2+k+1]$) can be extended to innerly disjoint paths in K starting from x and ending in a vertex of S. 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. These paths form at least $k+2$ innerly 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. This is easy to see if we regard T' as a rooted tree and we always choose a new path to put in $\mathcal{P}$ such that its distance from the root is the largest possible. 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, 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 $|K_3| < |S| + |N_{V(K-S)}(S)| + |L| \leq |S| + 2|L|$, 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. 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.

Assume now that the algorithm has a subtree D of T on which ϕ is already known. The algorithm proceeds step by step, at each step choosing a leaf r of D such that $d_D(r) < d_T(r)$. For the chosen vertex r, it determines ϕ on $N_T(r)$. This means also that it adds $N_T(r)$ to D, deletes $N_G(\phi(r)) \cap S$ from G and checks whether ϕ is still an isomorphism. When determining ϕ on $N_T(r)$, 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. Let $t_1, \dots, t_{n_1}$ denote the neighbors of r in T not in D, and let T_i be the tree component of $T - r$ containing t_i. Similarly, let $t'_1, \dots, t'_{n_2}$ be the neighbors of $r' = \phi(r)$ not in $\phi(D)$ that are connected to r' by edges not in K. Let T'_i denote the component of $G - r'$ that includes t'_i. Observe that either T'_i is a tree, or $r' \notin V(K)$ and T'_i contains K. Finally, let n_3 be the number of vertices in $N_G(r')$ not in $\phi(D)$ that are connected to r' by edges in K. Clearly, $n_1 \leq n_2 + n_3$, and the equality holds if and only if $N_G(r') \cap S = \emptyset$.

First, let us observe that if the tree T_i is isomorphic to T'_j for some i and j, then w.l.o.g. we can assume that $\phi(T_i) = T'_j$. 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, 13], 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, adding also T_i to the subgraph D. Hence, we only have to deal with the following case: no tree T_i ($i \in [n_1]$) is isomorphic to one of the graphs T'_j ($j \in [n_2]$). This argument makes our situation significantly easier, since every graph T'_j must contain some vertex from S. Therefore $n_2 \leq |S| = k$. By Lemma 4, r' can have degree at most $k^2 + k$ in K, so we get $n_3 \leq k^2 + k$, implying also $n_1 \leq n_2 + n_3 \leq k^2 + 2k$. If these bounds do not hold in some step, then the algorithm outputs 'No'.

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

Simple case: $n_2 + n_3 \leq 1$. In this case $n_1 \leq 1$. If $n_2 + n_3 = 0$ then the algorithm proceeds with the next step. Otherwise, let v be the unique vertex in $N_G(r') \setminus V(\phi(D))$. If $n_1 = 0$ then v must be in S, otherwise $\phi(t_1) = v$. According to this, the algorithm deletes v or extends ϕ on t_1, adding also t_1 to D.

Branching case: $n_2 + n_3 \geq 2$. In this case, the algorithm branches on every possible choice of determining ϕ on $N_T(r)$. Guessing $\phi(v)$ for a vertex $v \in N_{V(T-D)}(r)$ can result in at most $n_2 + n_3$ possibilities, so the number of possible branches in a branching step is at most $(n_2 + n_3)^{n_1} \leq (k^2 + 2k)^{k^2+2k}$.

We claim that in a single branch of a run of the algorithm on a solvable input, there can be at most $g(k) + 2k - 2$ branching steps. Observe that $n_3 \geq 2$ implies that r' is either the first vertex in $\phi(D)$ that is in K or $r' \in K_3$, so $n_3 \geq 2$ can happen at most $|K_3|+1 \leq g(k)$ times, by Lemma 4. If $n_2 \geq 2$, then $G-\phi(D)$ has more connected components containing vertices of S than $G-\phi(D-r)$ has. It is easy to see that this can be true for only at most $|S|-1$ such vertex r, so this case can happen at most $k-1$ times. Finally, let S^* denote those vertices of S that are not contained in K. Clearly, if $s \in S^*$, then $|N_{V(T_S)}(s)| \leq 1$. Now, if $n_2 = n_3 = 1$, then $r' \in V(K)$ and the edge $r't'_1$ must be one of the edges that connect to K a tree in $G-K$ containing a vertex in S^*. Observe that there can be at most $|S^*| \leq k-1$ such edges, thus the claim follows. Therefore, the algorithm only executes at most $g(k)+2k-2$ branching steps.

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 branching cases and the number of branches in a branching case 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 possibities on $\phi(r_0)$ enhances this to a 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* $|V(G)| = n+k$.

References

1. A. V. Aho, J. E. Hopcroft, J. D. Ullman: *The design and analysis of computer algorithms*, Addison-Wesley, 1974.
2. H. Bodlaender: On disjoint cycles. *Int. J. Found. Comput. Sci.*, 5:59–68, 1994.
3. 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.
4. Y. Chen, J. Flum: On parameterized path and chordless path problems. *22nd Annual IEEE Conference on Computational Complexity*, 250–263, 2007.
5. J. Díaz, D. M. Thilikos: Fast FPT-algorithms for cleaning grids. *STACS 2006, LNCS 3884*, 361–371, 2006.
6. R. Diestel: *Graph Theory*, Springer, Berlin, 2000.
7. R. G. Downey, M. R. Fellows: *Parameterized Complexity*, Springer, 1999.
8. D. Eppstein: Subgraph isomorphism in planar graphs and related problems. *J. Graph Algorithms Appl.* 3(3):1–27, 1999.
9. J. Flum, M. Grohe: *Parameterized Complexity Theory*, Springer, 2006.
10. M. R. Garey, D. S. Johnson: *Computers and Intractability. A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979.
11. 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.
12. J. E. Hopcroft, R. E. Tarjan: Dividing a graph into triconnected components. *SIAM J. Computing* 2(3):135–158, 1973.
13. J. E. Hopcroft, R. E. Tarjan: Efficient planarity testing. *J. Assoc. Comput. Mach.* 21:549–568, 1974.
14. A. Lingas: Subgraph isomorphism for biconnected outerplanar graphs in cubic time. *Theoret. Comput. Sci.* 63(3):295–302, 1989.
15. D. Marx, I. Schlotter: Obtaining a planar graph by vertex deletion. *WG 2007, LNCS 4769*, 292–303, 2007.
16. D. Matula: Subtree isomorphism in $O(n^{5/2})$. *Ann. Discrete Math.* 2:91–106, 1978.