Subexponential Time Algorithms for Embedding H-Minor Free Graphs

Hans L. Bodlaender[1], **Jesper Nederlof**[*2], **and**
Tom C. van der Zanden[3]

1 **Department of Computer Science, Utrecht University, Utrecht, The Netherlands; and**
Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands
`H.L.Bodlaender@uu.nl`

2 **Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands**
`J.Nederlof@tue.nl`

3 **Department of Computer Science, Utrecht University, Utrecht, The Netherlands**
`T.C.vanderZanden@uu.nl`

Abstract

We establish the complexity of several graph embedding problems: Subgraph Isomorphism, Graph Minor, Induced Subgraph and Induced Minor, when restricted to H-minor free graphs. In each of these problems, we are given a pattern graph P and a host graph G, and want to determine whether P is a subgraph (minor, induced subgraph or induced minor) of G. We show that, for any fixed graph H and $\epsilon > 0$, if P is H-Minor Free and G has treewidth tw, (induced) subgraph can be solved $2^{O(k^\epsilon tw + k/\log k)} n^{O(1)}$ time and (induced) minor can be solved in $2^{O(k^\epsilon tw + tw \log tw + k/\log k)} n^{O(1)}$ time, where $k = |V(P)|$.

We also show that this is optimal, in the sense that the existence of an algorithm for one of these problems running in $2^{o(n/\log n)}$ time would contradict the Exponential Time Hypothesis. This solves an open problem on the complexity of Subgraph Isomorphism for planar graphs.

The key algorithmic insight is that dynamic programming approaches can be sped up by identifying isomorphic connected components in the pattern graph. This technique seems widely applicable, and it appears that there is a relatively unexplored class of problems that share a similar upper and lower bound.

1998 ACM Subject Classification G.2.2 Graph Algorithms

Keywords and phrases subgraph isomorphism, graph minors, subexponential time

Digital Object Identifier 10.4230/LIPIcs.ICALP.2016.9

1 Introduction

We study several problems related to recognizing a pattern graph P as substructure of a host graph G: Subgraph Isomorphism, Induced Subgraph, Graph Minor and Induced Minor. We consider the case in which P excludes a specific minor H, $\epsilon > 0$ is a constant and give algorithms parameterized by the treewidth tw of G and the number of vertices k of

[*] This work was done while the second author was visiting the Simons Institute for the Theory of Computing.

P. Specifically, we show that for any $\epsilon > 0$ and graph H, if P is H-minor free and G has treewidth tw, (induced) subgraph can be solved $2^{O(k^\epsilon tw + k/\log k)} n^{O(1)}$ time and (induced) minor can be solved in $2^{O(k^\epsilon tw + tw \log tw + k/\log k)} n^{O(1)}$ time.

The $k/\log k$ dependence in the exponent is optimal: we present lower bounds based on the Exponential Time Hypothesis, showing that all these problems can not be solved in time $2^{o(n/\log n)}$, even when P is a tree and G is connected and series-parallel (and thus planar). Our lower bound answers a question of Marx [24] negatively: assuming the ETH, there is no $2^{O(\sqrt{k})} n^{O(1)}$ algorithm for subgraph isomorphism on planar graphs. This result is surprising, since for many problems on planar graphs a square root does appear in the exponent.

As an important special case of our result, we show that subgraph isomorphism can be solved in time $2^{O(k^\epsilon \sqrt{n} + k/\log k)}$ on H-Minor Free graphs (which includes planar, bounded-treewidth and bounded-genus graphs). Our result can be combined with a recent result of Fomin et al. [17] to show that subgraph isomorphism can be solved in $2^{O(k/\log k)} n^{O(1)}$ if P is connected and G is apex-minor free, which our lower bound shows is optimal (under ETH).

Subgraph isomorphism has received considerable attention in the literature. Results include polynomially solvable cases, such as recognizing a fixed pattern in planar graphs [13, 15], biconnected outerplanar graphs [22], graphs of log-bounded fragmentation [20] and graphs of bounded genus [10] and certain subclasses of graphs of bounded treewidth [26], exact algorithms [30], lower bounds [19, 11, 28] and results on parameterized complexity [25].

For a pattern graph P of treewidth t, subgraph isomorphism is solvable in $2^{O(k)} n^{O(t)}$ time using the color-coding technique [3]. If the host graph is planar, subgraph isomorphism can be solved in $2^{O(k)} n$ time [13]. In general graphs, subgraph isomorphism can be solved in $2^{O(n \log n)}$ time and, assuming the ETH, this is tight [16].

Graph minor problems are also of interest, especially in the light of Robertson and Seymour's seminal work on graph minor theory (see e.g. [29]) and the recent development of bidimensionality theory [12]. Many graph properties can be tested by checking for the inclusion of some minor. Testing whether a graph G contains a fixed minor P can be done in $O(n^3)$ time [27], this was recently improved to $O(n^2)$ [21]. However, the dependence on n is superexponential in a strong sense. Testing whether a graph P is a minor of a planar graph G can be done in $2^{O(k)} n^{O(1)}$ time [1], which is only single-exponential. Our lower bound shows this running time can not be improved to $2^{o(k/\log k)}$, assuming ETH. Our algorithms are subexponential in k, but (in contrast to [1, 21]) superpolynomial in n. This is to our knowledge the first subexponential minor testing algorithm (for a non-trivial class of graphs).

Our algorithms are based on dynamic programming on tree decompositions. In particular, we use dynamic programming on the host graph and store correspondences to vertices in the pattern graph. The key algorithmic insight is that this correspondence may or may not use certain connected components of the pattern graph (that remain after removing some separator vertices). Instead of storing for each component whether it is used or not, we identify isomorphic connected components and store only the number of times each is used.

In [20], the authors give an algorithm for subgraph isomorphism, which runs in polynomial time for a host graph of bounded treewidth and a pattern graph of log-bounded fragmentation (i.e. removing a separator decomposes the graph into at most logarithmically many connected components). This is achieved using a similar dynamic programming technique, which (in general) uses time exponential in the number of connected components that remain after removing a separator. By assuming the number of connected components (fragmentation) is logarithmic, the authors obtain a polynomial time algorithm. In contrast, we consider a graph class where fragmentation is unbounded, but the number of non-isomorphic connected components is small. This leads to subexponential algorithms.

This paper builds on techniques due to Bodlaender, Nederlof and van Rooij [7, 9]. They give a $2^{O(n/\log n)}$-time algorithm for finding tree decompositions with few bags and a matching lower bound (based on the Exponential Time Hypothesis), and a $2^{O(n/\log n)}$ algorithm for determining whether a given k-colored graph is a subgraph of a properly colored interval graph. These earlier papers, coupled with our results, suggest that this technique may have many more applications, and that there exists a larger class of problems sharing this upper and lower bound.

2 Preliminaries

Graphs. Given a graph G, we let $V(G)$ denote its vertex set and $E(G)$ its edge set. Given $X \subseteq V(G)$, let $G[X]$ denote the subgraph of G induced by X and use shorthand $E(X) = E(G[X])$. Let $Nb(v)$ denote the neighbourhood of v, that is, the vertices adjacent to v. For a set of vertices S, let $Nb(S) = \cup_{v \in S} Nb(v) \setminus S$. Let $CC(G)$ denote the set of the connected components of G. Given $X \subseteq V(G)$, we write as shorthand $CC(X) = CC(G[X])$.

Functions. Given a function $f : A \to B$, we let $f^{-1}(b) = \{a \in A \mid f(a) = b\}$. Depending on the context, we may also let $f^{-1}(b)$ denote (if it exists) the unique $a \in A$ so that $f(a) = b$. We say $g : A \to B$ is a *restriction* of $f : A' \to B'$ if $A \subseteq A'$ and $B \subseteq B'$ and for all $a \in A$, $g(a) = f(a)$. We say g is an *extension* of f if f is a restriction of g.

Isomorphism. We say a graph P is isomorphic to a graph G if there is a bijection $f : V(P) \to V(G)$ so that $(u, v) \in E(P) \iff (f(u), f(v)) \in E(G)$. We say a graph P is a subgraph of G if we can obtain a graph isomorphic to P by deleting edges and or vertices from G, and we say P is an induced subgraph if we can obtain it by deleting only vertices.

Contractions, minors. We say a graph G' is obtained from G by contracting edge (u, v), if G' is obtained from G by replacing vertices u, v with a new vertex w which is made adjacent to all vertices in $Nb(u) \cup Nb(v)$. A graph G' is a minor of G if a graph isomorphic to G' can be obtained from G by contractions and deleting vertices and/or edges. G' is an induced minor if we can obtain it by contractions and deleting vertices (but not edges).

Tree decompositions. A tree decomposition of a graph G is a rooted tree T with for every vertex $i \in V(T)$ a bag $X_i \subseteq V(G)$, such that $\cup_{i \in V(T)} X_i = V(G)$, for all $(u, v) \in E(G)$ there is an $i \in V(T)$ so that $\{u, v\} \subseteq X_i$ and for all $v \in V(G)$, $T[\{i \in V(T) \mid v \in X_i\}]$ is connected. The *width* of a tree decomposition is $\max_{i \in V(T)} |X_i| - 1$ and the *treewidth* of a graph G is the minimum width over all tree decompositions of G. For a node $t \in T$, we let $G[t]$ denote the subgraph of G induced by the vertices contained in the bags of the subtree of T rooted at t.

To simplify our algorithms, we assume that a tree decomposition is given in *nice* form, where each node is of one of four types:

- **Leaf**: A leaf node is a leaf $i \in T$, and $|X_i| = 1$.
- **Introduce**: An introduce node is a node $i \in T$ that has exactly one child $j \in T$, and X_i differs from X_j only by the inclusion of one additional vertex.
- **Forget**: An introduce node is a node $i \in T$ that has exactly one child $j \in T$, and X_i differs from X_j only by the removal of one vertex.
- **Join**: A join node is a node $i \in T$ with exactly two children $j, k \in T$, so that $X_i = X_j = X_k$.

A tree decomposition can be converted to a nice tree decomposition of the same width and of linear size in linear time [5].

3 Algorithmic Results

We begin by describing our algorithm for subgraph isomorphism, which is based on dynamic programming on a tree decomposition T of the host graph G. The algorithm is somewhat similar to that of Hajiaghayi et al. [20] for subgraph isomorphism on log-bounded fragmentation graphs, and we use similar notions of (extensible) partial solutions and characteristic of a partial solution (Section 3.1). Our main contribution is the canonization technique (Section 3.2) and its analysis (Section 3.3), which gives the subexponential running time.

3.1 An algorithm for Subgraph Isomorphism

▶ **Definition 1** ((Extensible) Partial Solution)**.** For a given node $t \in T$ of the tree decomposition of G, a *partial solution (relative to t)* is a triple (G', P', ϕ) where G' is a subgraph of $G[t]$, P' is an induced subgraph of P and $\phi : V(G') \to V(P')$ is an isomorphism from G' to P'.

Say that a partial solution (G', P', ϕ) relative to t is *extensible* if there exists an extension of ϕ, $\psi : V(G'') \to V(P)$ which is an isomorphism from a subgraph G'' of G to P where $V(G'') \cap V(G[t]) = V(G')$.

To facilitate dynamic programming, at node t of the tree decomposition we only consider partial solutions (G', P', ϕ) which *might* be extensible (i.e. we attempt to rule out non-extensible solutions). Note that in a partial solution we have already decided on how the vertices in $G[t]$ are used, and the extension only makes decisions about vertices not in $G[t]$. Instead of dealing with partial solutions directly, our algorithm works with characteristics of partial solutions:

▶ **Definition 2** (Characteristic of a Partial Solution)**.** The characteristic (f, S) of a partial solution (G', P', ϕ) relative to a node $t \in T$ is a function $f : X_t \to V(P) \cup \{\square\}$, together with a subset $S \subseteq V(P) \setminus f(X_t)$, so that:

- for all $v \in V(G') \cap X_t$, $f(v) = \phi(v)$ and $f(v) = \square$ otherwise,
- f is injective, except that it may map multiple elements to $\square$,
- $S = V(P') \setminus \phi(X_t)$.

The following easy observation justifies restricting our attention to characteristics of partial solutions:

▶ **Lemma 3** (Equivalent to Lemma 10, [20])**.** *If two partial solutions have the same characteristic, either both are extensible or neither is extensible.*

▶ **Lemma 4.** *If (f, S) is the characteristic of an extensible partial solution (G', P', ϕ) relative to a node $t \in T$, then S is a union of connected components of $P[V(P) \setminus f(X_t)]$.*

Proof (due to Hajiaghayi et al. [20]). Suppose there exist adjacent vertices $v_1, v_2 \in V(P) \setminus \phi(X_t)$ and $v_1 \in V(P')$, $v_2 \notin V(P')$. Then it is never possible to find a preimage u for v_2 in an extension of (G', P', ϕ): we require that $u \notin V(G[t])$, but all vertices adjacent to $\phi^{-1}(v_1)$ are contained in $V(G[t])$. ◀

The latter fact will be key to achieving the subexponential running time. The requirement that S is a union of connected components also appears in the definition of 'good pair' in Bodlaender et al. [7]. We show how to compute the characteristics of partial solutions in

Procedure 1 Leaf case: computes the partial solution characteristics for a leaf bag $t \in T$, with $X_t = \{v\}$.

```
1: let $R = \emptyset$
2: for each $u \in V(P) \cup \{\square\}$ do
3:     let $f : X_t \to V(P) \cup \{\square\}$ be the function so that $f(v) = u$
4:     let $R = R \cup \{(f, \emptyset)\}$
5: end for
6: filter $R$
7: return $R$
```

Procedure 2 Introduce case: introduces a vertex v into a bag X_t.

```
1: let $R$ be the set of partial solution characteristics for $t$
2: let $R' = \emptyset$
3: for each $(f, S) \in R$ and each $u \in V(P) \setminus (f(X_t) \cup S) \cup \{\square\}$ do
4:     if $u = \square$ or for all $w \in Nb(u) \cap f(X_t), (v, f^{-1}(w)) \in E(G)$ then
5:         let $f' : X_t \cup \{v\} \to V(P) \cup \{\square\}$ be the extension of $f$ so that $f(v) = u$
6:         let $R' = R' \cup \{(f', S)\}$
7:     end if
8: end for
9: filter $R'$
10: return $R'$
```

Procedure 3 Forget case: forgets a vertex v from a bag X_t.

```
1: let $R$ be the set of partial solution characteristics for $t$
2: let $R' = \emptyset$
3: for each $(f, S) \in R$ do
4:     let $f'$ be the restriction of $f$ to $X_t \setminus \{v\}$
5:     if $f(v) = \square$ or $f(v)$ is not adjacent to any vertex of $V(P) \setminus (f(X_t) \cup S)$ then
6:         let $R' = R' \cup \{(f', S \cup \{f(v)\} \setminus \{\square\})\}$
7:     end if
8: end for
9: filter $R'$
10: return $R'$
```

Procedure 4 Join case: combines the partial solution characteristics for two bags $X_s = X_t$.

```
1: let $R$ be the set of partial solution characteristics for $s$
2: let $T$ be the set of partial solution characteristics for $t$
3: let $R' = \emptyset$
4: for each $(f, S) \in R$ and each $(g, Q) \in T$ do
5:     if $f = g$ and $S \cap Q = \emptyset$ then
6:         let $R' = R' \cup \{(f, S \cup Q)\}$
7:     end if
8: end for
9: filter $R'$
10: return $R'$
```

a bottom-up fashion, so that we can tell whether G has a subgraph isomorphic to P by examining the characteristics of the root bag. We proceed by giving pseudocode for the leaf, introduce, forget and join cases and argue for their correctness.

The correctness of Procedure 1 is self-evident, as we simply enumerate all the possibilities for f (which means guessing a vertex in P to map v to). We will give details of the *filter* procedure in the next section, for now it suffices to treat the pseudocode as if this call were not present.

Procedure 2 extends existing partial solutions by choosing a vertex to map v to. To ensure we obtain valid characteristics of partial solutions, we check that for any edge incident to v in $P[S \cup f(X_t)]$ there is a corresponding edge in G. Because S is a union of connected components of $G[V(P) \setminus f(X_t)]$, $f(v)$ can not be adjacent to any vertex in S, and thus it suffices to check adjacency only against vertices in $f(X_t)$. Then S remains a union of connected components since the removal of a vertex can only further disconnect S. Note that u is chosen so that f remains injective.

Procedure 3 discards any solutions that would result in S not remaining the union of connected components that we require after forgetting a vertex (note that this means we keep only partial solutions were we have already chosen preimages for all of the neighbours of the image of the vertex being forgotten).

Finally, consider Procedure 4. Because (as a basic property of nice tree decompositions) $V(H[i]) \cap V(H[j]) = X_i$, we obtain an injective function if and only if $S \cap R = \emptyset$. We can therefore merge two partial solutions if they map the vertices of $X_t = X_s$ in the same way and $S \cap R = \emptyset$. Note that we do indeed create all possible partial solutions in this way: given a partial solution, we can split it into partial solutions for the left and right subtrees since (as there are no edges between the internal vertices of the left and right subtrees) a connected component of S must be covered entirely by either the left or right subtree.

These procedures, applied bottom-up on the tree decomposition, give an algorithm that decides subgraph isomorphism. Note that if one exists, a solution can be reconstructed from the characteristics.

3.2 Reducing the number of partial solutions using isomorphism tests

In this section, we show how adapt the algorithm from the previous section to achieve the claimed running time bound. This involves a careful analysis of the number of characteristics, and using isomorphism tests to reduce this number. Currently, if the connected components of S are small (e.g., $O(1)$ vertices each) then their number is large (e.g., $\Omega(n)$ components) and thus in the worst case we have $2^{\Omega(n)}$ partial solutions. However, if there are many small connected components many will necessarily be isomorphic to each other (since there are only few isomorphism classes of small connected components) and we can thus reduce the number of characteristics by identifying isomorphic connected components:

▶ **Definition 5** (Partial Solution Characteristic Isomorphism). Given a bag $t \in T$, two characteristics of partial solutions $(f, S), (g, R)$ for t are *isomorphic* if:

- $f = g$,
- there is a bijection $h : CC(S) \to CC(R)$,
- for all connected components $c \in CC(S)$, c and $h(c)$ are isomorphic when all vertices $v \in c$ vertices are labelled with $Nb(v) \cap f(X_t)$ (i.e. the set of vertices of $f(X_t)$ to which v is adjacent).

Clearly, the algorithm given in the previous section remains correct even if after each procedure we remove duplicate isomorphic characteristics. To this end, we modify the join

Procedure 5 Connected Component Canonization: Computes a canonical representation for a union of connected components $S \subseteq V(P) \setminus f(X_t)$

```
1: let $S'$ be the union of the large connected components of $S$
2: let $Q = \emptyset$
3: for each small connected component $s$ of $S$ do
4:     compute the canonical representation $r$ of $s$ when each $v \in V(s)$ is labelled with $Nb(v) \cap f(X_t)$
5:     let $Q = Q \cup \{r\}$
6: end for
7: Sort $S'$ and $Q$ lexicographically
8: return $(S', Q)$
```

Procedure 7 Filtering Procedure: Filters a set of partial solution characteristics R to remove duplicates

```
1: compute the canonical representation $C_S$ for every $(f, S) \in R$ using Procedure 5
2: sort $R$ first by $f$, then by $C_S$ in lexicographical order
3: loop over $R$, removing all but one of each group of isomorphic partial solutions
4: return $R$
```

case (Procedure 4): the disjointness check $S \cap Q = \emptyset$ should be replaced with a check that if $P[V(P) \setminus f(X_t)]$ contains $N_P(y)$ connected components of isomorphism class y, and $P[S]$ (resp. $P[Q]$) contains $N_S(y)$ (resp. $N_Q(y)$) connected components of isomorphism class y, then $N_S(y) + N_Q(y) \leq N_P(y)$. Similarly, the statement $S \cup Q$ needs to be changed to, if the union is not disjoint, replace connected components that occur more than once with other connected components of the same isomorphism class (so as to make the union disjoint while preserving the total number of components of the same type).

Call a connected component *small* if it has at most $c \log k$ vertices, and large otherwise. We let $c > 0$ be a constant that depends only on $|V(H)|$ and ϵ. We do not state our choice of c explicitly, but in our analysis we will assume it is "small enough".

For a small connected component s, we label each of its vertices by the subset of vertices of $f(X_t)$ to which it is adjacent. We then compute a canonical representation of this labeled component, for example by considering all permutations of its vertices, and choosing the permutation that results in the lexicographically smallest representation. Note that since we only canonize the small connected components using such a trivial canonization algorithm does not affect the running time of our algorithm, as $(c \log k)!$ is only slightly superpolynomial.

Procedure 5 computes a canonical representation of a partial solution. It requires that we have some predefined ordering of the vertices of G. The canonization procedure 5 allows us to define the *filter* procedure (Procedure 6).

Traditionally, a canonization is a function that maps non-isomorphic graphs to distinct strings, and isomorphic graphs to identical strings. We use this term slightly more loosely, as our canonization procedure 5 may map isomorphic graphs to distinct strings since it only canonizes the small connected components. Thus, Procedure 6 may not remove all duplicate isomorphic partial solutions. However, we will show that it removes enough of them.

3.3 Bounding the number of non-isomorphic partial solutions

In this section, we analyse the number of non-isomorphic partial solutions, and show that the algorithm given in the previous section indeed achieves the stated time bound. In the following, let $\epsilon > 0$ and let G be a graph of treewidth at most tw. Furthermore, suppose that P is H-minor free for some fixed graph H.

Recall that a partial solution for a node $t \in T$ of the tree decomposition consists of $f : X_t \to V(P) \cup \{\square\}$ and a subset $S \subseteq V(P) \setminus f(X_t)$, which is a union of connected components of the subgraph induced by $S \subseteq V(P) \setminus f(X_t)$. The number of choices for f is at most $(k+1)^{|X_t|} = 2^{O(tw \log k)}$. We now proceed to bound the number of cases for S.

We distinguish between connected components of $V(P) \setminus f(X_t)$ of which there are "few", and connected components of $V(P) \setminus f(X_t)$ of which there can be "many", but few non-isomorphic ones. For some constant c, we say a component is *small* if it has at most $c \log k$ vertices, and *large* otherwise. The large connected components are amongst the few, since there are at most $k/(c \log k)$ components with at least $c \log k$ vertices. For each of these components, we store explicitly whether or not it is contained in S. They contribute a factor of $2^{O(k/\log k)}$ to the number of cases. For the small connected components, we will show a partition into the "few" (which we treat similarly to the large connected components), and into the "many, but few non-isomorphic" (for which we store, for each isomorphism class, the number of components from that isomorphism class contained in S).

▶ **Claim.** *For a given node t and function $f : X_t \to V(P)$, there are at most $O(k^{\epsilon/2} tw)$ isomorphism classes of small connected components.*

Proof. For a (small) connected component $x \in CC(V(P) \setminus f(X_t))$, its isomorphism class is determined by the isomorphism class of x itself, and the adjacency of vertices $v \in x$ to vertices in $f(X_t)$. Since $|x| \leq c \log k$ and P is H-minor free, there exists a constant $C_H > 1$ so that there are at most $2^{C_H \cdot c \log k}$ cases for the isomorphism class of x itself (see [4]).

What remains is to bound the number of cases for adjacency of x to X_t. In this specific case, $Nb(x)$ denotes the set of vertices of X_t to which x is incident, that is, $v \in Nb(x)$ if and only if $v \in X_t$ and there exists a vertex $u \in x$ so that $(u, v) \in E(P)$. Using the following lemma, we further divide the small connected components into two cases: the components with a large neighbourhood, and the components with a small neighbourhood.

▶ **Lemma 6** (Gajarskỳ et al., special case of Lemma 3 of [18]). *Let H be a fixed graph. Then there exists a constant d (depending on H), so that if $G = (A, B, E)$ is H-minor free and bipartite, there are at most*

- *$O(|A|)$ vertices in B with degree greater than d,*
- *$O(|A|)$ subsets $A' \subseteq A$ such that $A' = Nb(u)$ for some $u \in B$.*

Taking $A = X_t$, deleting the edges between vertices in X_t and contracting every connected component $x \in CC(V(P) \setminus f(X_t))$ to a single vertex in B, the lemma states that there are at most $O(tw)$ components with $|Nb(x)| > d$ and that the components with $|Nb(x)| \leq d$ have at most $O(tw)$ distinct neighbourhoods in X_t.

For the connected components $x \in CC(V(P) \setminus f(X_t))$ with $|Nb(x)| \leq d$, we have $2^{C_H \cdot c \log k}$ cases for the isomorphism class of x, $O(tw)$ cases for $Nb(x)$ and for every vertex of x, at most 2^d cases for incidence to $Nb(x)$. We thus have at most $2^{C_h \cdot c \log k} \cdot O(tw) \cdot (2^d)^{c \log k}$ isomorphism classes for $x \in CC(S)$ with $Nb(x) \leq d$. For sufficiently small $c > 0$, the asymptotic complexity is $O(k^{\epsilon/2} tw)$. ◀

Since of each component there can be most k occurrences in S, the total number of cases for storing the multiplicity of each class of components is $(k+1)^{O(k^{\epsilon/2}tw)} = 2^{O(k^{\epsilon/2}tw \log k)} = 2^{O(k^{\epsilon}tw)}$.

We now have all the results we need to finish the analysis. Storing the multiplicities of the small connected components gives $2^{O(k^{\epsilon}tw)}$ cases, while storing the subset of large connected components explicitly contributes $2^{O(k/\log k)}$ cases. A partial solution is further characterized by f, for which there are only $2^{O(tw \log k)}$ cases. For a given node t of the tree decomposition, there are thus at most $2^{O(k^{\epsilon}tw + k/\log k)}$ partial solutions.

Finally, we can compute a 5-approximate tree-decomposition of G in time exponential in tw [6], perform dynamic programming as described in Procedures 1-6 to obtain:

▶ **Theorem 7.** *For any graph H and $\epsilon > 0$,* Subgraph Isomorphism *can be solved in time $2^{O(k^{\epsilon}tw + k/\log k)}n^{O(1)}$ if the host graph has treewidth tw and the pattern graph is H-minor free.*

3.4 Adaptation to other problems

In this section, we discuss how our algorithm for Subgraph Isomorphism can be adapted to Induced Subgraph and (Induced) Minor. We begin by describing the graph minor case, then give a brief note on how to adapt both algorithms for the induced case. Some details are omitted from this extended abstract.

Note that P is a minor of G if and only if we have a function $f : V(G) \to V(P) \cup \{\square\}$, such that

- for all $v \in V(P)$, $f^{-1}(v)$ is non-empty, and induces a connected subgraph of G,
- for all $(v, w) \in E(P)$, there are $x \in f^{-1}(v)$ and $y \in f^{-1}(w)$ with $(x, y) \in E(G)$.

Vertices that are deleted are mapped to $\square$, otherwise $f(v)$ gives the vertex that v is contracted to. Call such a function a *solution* for the Graph Minor problem.

If we restrict such solutions to a subgraph $G[t]$, we obtain the notion of partial solution:

▶ **Definition 8** (Partial Solution (Graph Minor)). Given a node $t \in T$ of the tree decomposition of G, a *partial solution for the* Graph Minor *problem relative to a node t* is a function $f : V(G[t]) \to V(P) \cup \{\square\}$, such that

1. For each $v \in V(P)$, at least one of the following three cases holds:
   a. each connected component of $G[t][f^{-1}(v)]$ contains at least one vertex from X_t,
   b. $G[t][f^{-1}(v)]$ has one connected component,
   c. $f^{-1}(v)$ is empty.
2. For all $(v, w) \in E(P)$, at least one of the following cases holds:
   a. Some vertex of $f^{-1}(v)$ is adjacent to some vertex of $f^{-1}(w)$ in $G[t]$,
   b. $f^{-1}(v) \cap X_t \neq \emptyset$ and $f^{-1}(w) \cap X_t \neq \emptyset$,
   c. $f^{-1}(v) \cap X_t \neq \emptyset$ and $f^{-1}(w) = \emptyset$,
   d. $f^{-1}(w) \cap X_t \neq \emptyset$ and $f^{-1}(v) = \emptyset$.
   e. $f^{-1}(w) = \emptyset$ and $f^{-1}(v) = \emptyset$.

Intuitively, in 1) we require that the preimage of v can still be made connected (a), is already connected (b) or has not been assigned yet (c), and in 2) we require that the edge (v, w) is already covered (a), can still be covered (b,c,d,e).

As before, our dynamic programming algorithm uses characteristics of partial solutions:

▶ **Definition 9** (Characteristic of a partial solution (Graph Minor)). Given a node $t \in T$ of the tree decomposition of G and a *partial solution for the* Graph Minor *problem relative to a node t* $f : V(G[t]) \to V(P) \cup \{\square\}$, the *characteristic* of f is a tuple $(f', S, \sim, F)$ such that:

1. f' is the restriction of f to X_t,
2. $S \subseteq V(P)$ with $S = f(V(G[t])) \setminus (f(X_t) \cup \{\square\})$,
3. $\sim$ is an equivalence relation on X_t, with $v \sim w$, if and only if $f(v) = f(w)$ and there exists a path from v to w in $G[t]$ such that for all vertices x on this path $f(x) = f(v)$.
4. $F \subseteq E(P[f(X_t)])$ such that for every $(v, w) \in E(P[f(X_t)])$, it holds that $(v, w) \in F$ if and only if each of the following holds:
   a. $f^{-1}(v) \cap X_t \neq \emptyset$,
   b. $f^{-1}(w) \cap X_t \neq \emptyset$,
   c. There are $x \in f^{-1}(v)$ and $y \in f^{-1}(w)$ with $(x, y) \in E(G[t])$.

As before, it is easy to see the equivalence between the existence of a minor, solution, and partial solution with certain characteristic.

Compared to our approach for subgraph isomorphism, we no longer require f to be injective – $f^{-1}(v)$ corresponds to the vertices that are contracted to form v. We require that $f^{-1}(v)$ eventually becomes connected. This can either be achieved inside the bag, be achieved below the bag (in the tree decomposition), or above the bag. The relation $\sim$ tracks which components are already connected by vertices below the bag. Similarly, edges inside $P[f(X_t)]$ might not have corresponding edges inside $G[X_t]$, but might instead correspond to edges below or above this bag. The set F stores which edges correspond to edges below the current bag.

This lemma is the counterpart of Lemma 3 and shows that we can apply dynamic programming:

▶ **Lemma 10.** *If partial solutions for the* GRAPH MINOR *problem f and g have the same characteristic and are both relative to t, then f can be extended to a solution if and only if g can be extended to a solution.*

The following lemma shows that we can apply our technique of reducing the number of partial solution characteristics by using isomorphisms:

▶ **Lemma 11.** *A partial solution for* GRAPH MINOR *f with characteristic $(f', S, \sim, F)$ can be extended to a solution only if S is a union of connected components of $G[V(P) \setminus f(X_t)]$.*

The analysis of the number of cases of f' and S remains unchanged. There are at most $(tw)^{tw} = 2^{O(tw \log tw)}$ cases for $\sim$, and since P is sparse, at most $2^{O(tw)}$ cases for F.

For the induced cases, only a small modification is needed: it suffices to check in the introduce case that all neighbours (in X_t) of the vertex being introduced are mapped to vertices that are adjacent to the image of the introduced vertex and discard the partial solution otherwise. We thus obtain the following theorem:

▶ **Theorem 12.** *For any graph H and $\epsilon > 0$, if the host graph has treewidth tw and the pattern graph is H-minor free,* SUBGRAPH ISOMORPHISM *and* INDUCED SUBGRAPH *can be solved in time $2^{O(k^\epsilon tw + k/\log k)} n^{O(1)}$ and* GRAPH MINOR *and* INDUCED MINOR *can be solved in time $2^{O(k^\epsilon tw + tw \log tw + k/\log k)} n^{O(1)}$.*

As a direct corollary, we have that SUBGRAPH ISOMORPHISM, GRAPH MINOR, INDUCED SUBGRAPH and INDUCED MINOR can be solved in $2^{O(n^{0.5+\epsilon} + k/\log k)}$ time if the host graph is H-minor free for some fixed graph H, as H-minor free graphs have treewidth $O(\sqrt{n})$ [2]. Important special cases include planar graphs, graphs of bounded genus, and graphs of bounded treewidth.

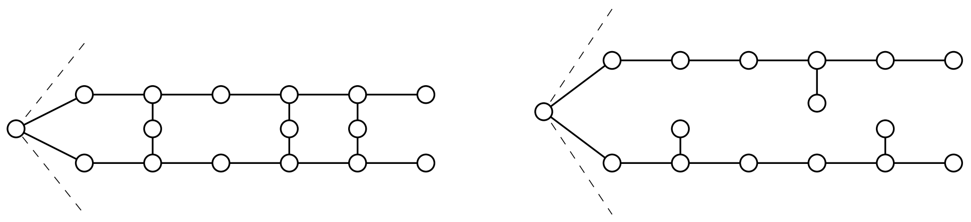

Figure 1 Construction used in the proof of Theorem 14. Host graph (a), representing a string $101001 \in A$ and pattern graph (b), representing strings $000100 \in B$ and $010010 \in C$. The dashed lines represent additional paths attached to vertices u and u'.

4 Hardness Results

Both (Induced) Subgraph and (Induced) Minor are known to be *NP*-complete, even if P is a tree and G is series-parallel (and thus planar), connected and all but one vertex of P and G have degree at most 3, by reduction from Three-Dimensional Matching [26].

We obtain our $2^{o(n/\log n)}$ lower bound by a reduction very similar to the reduction from Three-Dimensional Matching in [26]. We instead reduce from the String 3-Groups problem [7]. In the following, given a string s, we let s_i denote the i^{th} character of s.

String 3-Groups

Instance: Sets $A, B, C \subseteq \{0,1\}^{6\lceil\log n\rceil+1}$, $|A| = |B| = |C| = n$

Question: Do there exist n triples, so that for each chosen triple $(a, b, c) \in A \times B \times C$ and for all i, $a_i + b_i + c_i \leq 1$ and each element of A, B, C occurs in exactly one triple?

▶ **Theorem 13** ([7], [8]). *Assuming the Exponential Time Hypothesis, there is no algorithm solving* String 3-Groups *in* $2^{o(n)}$ *time.*

▶ **Theorem 14.** *Assuming the Exponential Time Hypothesis, there is no algorithm solving* Subgraph Isomorphism *in* $2^{o(n/\log n)}$ *time, even when the pattern graph is a tree and the host graph is connected and series-parallel and in both the host graph and pattern graph, all vertices but one have maximum degree 3.*

Proof. Let A, B, C be an instance of String 3-Groups. We modify the instance by prepending a 0 to each string in A and B, and a 1 to each string in C. Let $m = 6\lceil\log n\rceil + 2$.

To construct the host graph G, we take a vertex u. For each $a \in A$ we take a path $p_1, \ldots, p_m$ and a path $q_1, \ldots, q_m$. We add edges (p_1, u) and (q_1, u). Whenever $a_i = 0$, we create a vertex r_i and edges $(p_i, r_i), (r_i, q_i)$.

To construct the pattern graph P, we take a vertex u'. For each $b \in B$ (resp. each $c \in C$) we take a path $s_1, \ldots, s_m$. We add an edge (s_1, u'). Whenever $b_i = 1$ (resp. $c_i = 1$), we create a vertex t_i and an edge (s_i, t_i).

A solution to the String 3-Groups problem and this instance of Subgraph Isomorphism correspond as follows: u is mapped to u'. If (a, b, c) is a triple in a solution to String 3-Groups, then the path $s_1, \ldots, s_m$ corresponding to b can be mapped to the path $p_1, \ldots, p_m$ corresponding to a, while the path $s_1, \ldots, s_m$ corresponding to c is mapped to the path $q_1, \ldots, q_m$. Clearly such a mapping is possible if and only if for each i, at most one of a_i, b_i or c_i is 1, since the vertex v_i only exists if $a_i = 0$ and can be used at most once (by either the vertex t_i corresponding to b_i or the vertex t_i corresponding to c_i).

In the reverse direction, note that any subgraph isomorphism must map u to u' by virtue of their degrees. Clearly, the paths in H must correspond one-to-one with paths in G. The correspondence of the paths immediately gives us a partition into triples. The construction enforces that in each such triple, in each position at most one of the strings has a 1, as required. Note that since we modified the instance so that each string $c \in C$ starts with a 1, no triple will contain more than one string from c and consequently, each triple contains exactly one string from each of a, b, c.

Since the graph created in the reduction has $O(n \log n)$ vertices, and assuming the Exponential Time Hypothesis there is no $2^{o(n)}$ algorithm for String 3-Groups, there is no $2^{o(n/\log n)}$ time algorithm for Subgraph Isomorphism, even for the graph classes as claimed in the theorem. ◀

The proof of Theorem 14 can be adapted to Induced Subgraph by subdividing each edge (p_i, r_i) once. Furthermore, the proof also works for (Induced) Graph Minor, since performing a contraction in H is never beneficial.

▶ **Theorem 15.** *Assuming the Exponential Time Hypothesis, there is no algorithm solving* Subgraph Isomorphism, Graph Minor, Induced Subgraph *and* Induced Minor *in $2^{o(n/\log n)}$ time, even when the pattern graph is a tree and the host graph is connected and series-parallel and in both the host graph and pattern graph, all vertices but one have maximum degree 3.*

5 Conclusion

We have presented algorithms for (Induced) Subgraph and (Induced) Minor that, by taking advantage of isomorphic structures in the pattern graph, run in subexponential time on H-minor free graphs. These algorithms are essentially optimal since we have shown that the existence of $2^{o(n/\log n)}$ time algorithms would contradict the Exponential Time Hypothesis. We have thus settled the (traditional) complexity of these problems on (general) H-Minor Free graphs.

Our result applies to a wide range of graphs: we require P to be H-Minor Free and G to have truly sublinear treewidth. Some restriction on G is indeed necessary, since if G is an arbitrary graph then Hamiltonian path is a special case (in which P is a path) and a $2^{o(n)}$ algorithm would contradict the ETH [23].

An interesting open question is whether the parameterized complexity can still be improved. Perhaps the dependence of the running time on the treewidth of G can be removed: does there exist an algorithm for subgraph isomorphism on planar graphs running in $2^{O(k/\log k)}n^{O(1)}$? Our result, combined with one due to Fomin et al. [17] implies that the answer to this question is yes if P is connected and G is apex-minor free, but the problem remains open otherwise.

We note that our lower bound proof also works for Immersion [14]. However, our algorithmic technique does not seem to work for immersion. Does the immersion problem also have a $2^{O(n/\log n)}$ algorithm, or is a stronger lower bound possible?

Lemma 6 holds for a more general class of graphs, and we believe it may be possible to extend our result to patterns from a graph class with expansion $O(1)$ or perhaps expansion $O(\sqrt{r})$. We note that for different graph classes, a tradeoff between the size of the small connected components and the factor $k/\log k$ in the exponent is possible: it might be possible to obtain a $2^{O(n/\log\log n)}$-time algorithm for less restrictive graph classes.

Together with the Minimum Size Tree/Path Decomposition problem [7], these problems are amongst the first for which a $2^{\Theta(n/\log n)}$ upper and lower bound is known. Our work shows that the techniques from [7] can be adapted to other problems, and we suspect there may be many more problems for which identifying isomorphic components can speed up dynamic programming algorithms.

References

1 Isolde Adler, Frederic Dorn, Fedor V. Fomin, Ignasi Sau, and Dimitrios M. Thilikos. Fast minor testing in planar graphs. *Algorithmica*, 64(1):69–84, 2012.

2 Noga Alon, Paul Seymour, and Robin Thomas. A separator theorem for nonplanar graphs. *Journal of the American Mathematical Society*, 3(4):801–808, 1990.

3 Noga Alon, Raphy Yuster, and Uri Zwick. Color-coding: A New Method for Finding Simple Paths, Cycles and Other Small Subgraphs Within Large Graphs. In *Proceedings of the Twenty-sixth Annual ACM Symposium on Theory of Computing*, STOC'94, pages 326–335, New York, NY, USA, 1994. ACM. `doi:10.1145/195058.195179`.

4 Omid Amini, Fedor V. Fomin, and Saket Saurabh. Counting subgraphs via homomorphisms. *SIAM Journal on Discrete Mathematics*, 26(2):695–717, 2012.

5 Hans L. Bodlaender. Treewidth: Algorithmic techniques and results. In Igor Prívara and Peter Ružička, editors, *Mathematical Foundations of Computer Science 1997*, volume 1295 of *Lecture Notes in Computer Science*, pages 19–36. Springer Berlin Heidelberg, 1997. `doi:10.1007/BFb0029946`.

6 Hans L. Bodlaender, Pal Gronas Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and Michal Pilipczuk. An $O(c^k n)$ 5-approximation algorithm for treewidth. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, pages 499–508. IEEE, 2013.

7 Hans L. Bodlaender and Jesper Nederlof. Subexponential Time Algorithms for Finding Small Tree and Path Decompositions. In *Algorithms–ESA 2015*, pages 179–190. Springer, 2015.

8 Hans L. Bodlaender and Jesper Nederlof. Subexponential time algorithms for finding small tree and path decompositions. *arXiv preprint arXiv:1601.02415*, 2016.

9 Hans L. Bodlaender and Johan M.M. Van Rooij. Exact algorithms for intervalizing colored graphs. In *Theory and Practice of Algorithms in (Computer) Systems*, pages 45–56. Springer, 2011.

10 Paul Bonsma. Surface split decompositions and subgraph isomorphism in graphs on surfaces. *arXiv preprint arXiv:1109.4554*, 2011.

11 Marek Cygan, Jakub Pachocki, and Arkadiusz Socała. The hardness of Subgraph Isomorphism. *arXiv preprint arXiv:1504.02876*, 2015.

12 Erik D. Demaine and MohammadTaghi Hajiaghayi. Bidimensionality: new connections between FPT algorithms and PTASs. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 590–601. Society for Industrial and Applied Mathematics, 2005.

13 Frederic Dorn. Planar subgraph isomorphism revisited. *arXiv preprint arXiv:0909.4692*, 2009.

14 Rodney G. Downey and Michael R. Fellows. *Parameterized complexity*. Springer Science & Business Media, 2012.

15 David Eppstein. Subgraph Isomorphism in Planar Graphs and Related Problems. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'95, pages 632–640, Philadelphia, PA, USA, 1995. Society for Industrial and Applied Mathematics.

16 Fedor V. Fomin, Alexander Golovnev, Alexander S. Kulikov, and Ivan Mihajlin. Tight Bounds for Subgraph Isomorphism and Graph Homomorphism. *arXiv preprint arXiv:1507.03738*, 2015.

17 Fedor V. Fomin, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. Subexponential parameterized algorithms for planar and apex-minor-free graphs via low treewidth pattern covering. *arXiv preprint arXiv:1604.05999*, 2016.

18 Jakub Gajarskỳ, Petr Hliněnỳ, Jan Obdržálek, Sebastian Ordyniak, Felix Reidl, Peter Rossmanith, Fernando Sanchez Villaamil, and Somnath Sikdar. Kernelization using structural parameters on sparse graph classes. In *Algorithms–ESA 2013*, pages 529–540. Springer, 2013.

19 Arvind Gupta and Naomi Nishimura. The complexity of subgraph isomorphism for classes of partial k-trees. *Theoretical Computer Science*, 164(1):287–298, 1996.

20 MohammadTaghi Hajiaghayi and Naomi Nishimura. Subgraph isomorphism, log-bounded fragmentation, and graphs of (locally) bounded treewidth. *Journal of Computer and System Sciences*, 73(5):755–768, 2007.

21 Ken-ichi Kawarabayashi, Yusuke Kobayashi, and Bruce Reed. The disjoint paths problem in quadratic time. *Journal of Combinatorial Theory, Series B*, 102(2):424–435, 2012. `doi:10.1016/j.jctb.2011.07.004`.

22 Andrzej Lingas. Subgraph isomorphism for biconnected outerplanar graphs in cubic time. *Theoretical Computer Science*, 63(3):295–302, 1989. `doi:10.1016/0304-3975(89)90011-X`.

23 Daniel Lokshtanov, Dániel Marx, Saket Saurabh, et al. Lower bounds based on the exponential time hypothesis. *Bulletin of the EATCS*, (105):41–72, 2011.

24 Dániel Marx. What is next? Future directions in parameterized complexity. In *The Multivariate Algorithmic Revolution and Beyond*, pages 469–496. Springer, 2012.

25 Dániel Marx and Michał Pilipczuk. Everything you always wanted to know about the parameterized complexity of Subgraph Isomorphism (but were afraid to ask). *arXiv preprint arXiv:1307.2187*, 2013.

26 Jiří Matoušek and Robin Thomas. On the complexity of finding iso-and other morphisms for partial k-trees. *Discrete Mathematics*, 108(1):343–364, 1992.

27 Neil Robertson and Paul D. Seymour. Graph minors. XIII. The Disjoint Paths Problem. *Journal of Combinatorial Theory, Series B*, 63(1):65–110, 1995. `doi:10.1006/jctb.1995.1006`.

28 Maciej M. Sysło. The subgraph isomorphism problem for outerplanar graphs. *Theoretical Computer Science*, 17(1):91–97, 1982. `doi:10.1016/0304-3975(82)90133-5`.

29 Dimitrios M. Thilikos. The Multivariate Algorithmic Revolution and Beyond. chapter Graph Minors and Parameterized Algorithm Design, pages 228–256. Springer-Verlag, Berlin, Heidelberg, 2012.

30 Julian R. Ullmann. An algorithm for subgraph isomorphism. *Journal of the ACM (JACM)*, 23(1):31–42, 1976.