Subgraph Isomorphism, *log*-Bounded Fragmentation and Graphs of (Locally) Bounded Treewidth *

MohammadTaghi Hajiaghayi[1]

Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139, U.S.A.

Abstract. Matoušek and Thomas solved the subgraph isomorphism problem when the source graph has bounded degree and the host graph has bounded treewidth. In this paper, we introduce a new property for graphs called *log-bounded fragmentation*, by which we mean after removing any set of at most k vertices the number of connected components is at most $O(k \log n)$, where n is the number of vertices of the graph. We then extend the result of Matoušek and Thomas to the case in which the source graph is a *log*-bounded fragmentation graph and the host graph has bounded treewidth. In addition, we demonstrate how the subgraph isomorphism problem on minor-closed family of graphs of locally bounded treewidth can be solved in polynomial time, when the source graph is a *log*-bounded fragmentation graph and has constant diameter.

1 Introduction

In this paper we consider the subgraph isomorphism problem, in which we search for a subgraph of host graph H isomorphic to the source (or pattern) graph G. This problem has many applications in different areas like biology and organic chemistry [4].

Since the subgraph isomorphism problem is NP-complete [14], much attention has focused on solving this problem by adding restrictions to the source or host graph. Specific classes of graphs for which there are polynomial-time algorithms are as follows: trees, two-connected outerplanar graphs, and two-connected series-parallel graphs. These are all graphs of bounded treewidth. However, the subgraph isomorphism problem remains NP-complete for general graphs of bounded treewidth [24].

Graphs of bounded treewidth are known for their good algorithmic properties, since they allow us to use a dynamic programming approach for solving problems, especially those which can be described in the language of extended monadic second order logic (EMSOL). Except the case in which the source graph is fixed, the subgraph isomorphism problem can not be expressed in EMSOL, and the general approach of Arnborg et al. [1] can not be used. We need to add more restrictions to solve this problem in polynomial time. Matoušek and Thomas [20] have shown the subgraph isomorphism problem has an $O(n^{k+4.5})$ time algorithm for k-connected partial k-trees and an $O(n^{k+2})$ time algorithm for bounded degree partial k-trees. They have proved that the problem remains NP-complete when the source graph is a tree, and the host graph is a partial 2-tree which has at most one node of degree greater than three. Gupta and Nishimura [16], using a different approach, derive polynomial-time algorithms with the same asymptotic complexity for the k-connected partial k-tree case and other embeddings. Dessmark et al. [11] improve the running time of Gupta and Nishimura's algorithm from $O(n^{k+4.5})$ to $O(n^{k+2})$ for the case of k-connected partial k-trees. Gupta and Nishimura have proved the subgraph isomorphism problem on partial k-trees remains NP-complete when the source graph is not k-connected [17].

Matoušek and Thomas have presented their algorithm in a very complicated manner. Their approach has been stated for the minor containment problem and the solution for subgraph isomorphism can be obtained as a special case of this approach. In this paper, we extend the bounded degree result of Matoušek and Thomas for subgraph isomorphism to handle a more general property, namely bounded fragmentation. A graph is a *log-bounded fragmentation* graph if, after removing any set of at most k vertices, the number of connected components is at most $O(k \log n)$, where n is the number of vertices of the graph. The class of bounded fragmentation graphs contains the class of bounded degree graphs and other classes of graphs such as the class of Hamiltonian graphs (see Section 3). We also extend our results to minor-closed family of graphs of locally bounded treewidth.

This paper is organized as follows. First, we introduce the terminology used throughout the paper in Section 2. In Section 3, we present an overview of the general dynamic programming approach introduced by Arnborg and Proskurowski

* We attached an appendix which may be ready by the program committee at their discretion. Email: hajiagha@theory.lcs.mit.edu

for solving problems on graphs of bounded treewidth. In Section 4, we precisely state the bounded degree result of Matoušek and Thomas for the subgraph isomorphism problem and demonstrate how this result can be generalized to graphs with the *log*-bounded fragmentation property. We generalize testing subgraph isomorphism to graphs of locally bounded treewidth in Section 5. Finally we conclude with a list of open problems and potential extensions for future work in Section 6. It is worth mentioning that the full proofs of most Lemmas and Theorems in this paper can be found in the Appendix and will appear in the full paper.

2 Preliminaries

We assume the reader is familiar with general concepts of graph theory such as directed graphs, trees and planar graphs. The reader is referred to standard references for appropriate background [9].

Our graph terminology is as follows. All graphs are finite, simple and undirected, unless indicated otherwise. A graph G is represented by $G = (V, E)$, where V (or $V(G)$) is the set of vertices and E (or $E(G)$) is the set of edges. We denote an edge e in a graph G between u and v by $\{u, v\}$. We define n to be the number of vertices of a graph when it is clear from context. The maximum degree of G is denoted by $\Delta(G)$ and the minimum degree of G is denoted by $\delta(G)$. We define the *r-neighborhood* of a set $S \subseteq V(G)$, denoted by $N_G^r(S)$, to be the set of vertices at distance at most r from at least one vertex of $S \subseteq V(G)$; if $S = \{v\}$ we simply use the notation $N_G^r(v)$. The *diameter* of G, denoted by $diam(G)$, is the maximum over all distances between pairs of vertices of G. Two disjoint sets S and S' of vertices of undirected (directed) graph G are *adjacent* if and only if there are $u \in S$ and $v \in S'$ such that $\{u, v\} \in E(G)$ ($(u, v) \in E(G)$ or $(v, u) \in E(G)$).

A graph $G' = (V', E')$ is a *subgraph of* G if $V' \subseteq V$ and $E' \subseteq E$. A graph $G' = (V', E')$ is an *induced subgraph of* G, denoted by $G[V']$, if $V' \subseteq V$ and E' contains all edges of E which have both end vertices in V'. G is a *supergraph* of G' if G' is a subgraph (not necessarily induced subgraph) of G.

This paper is devoted to solving the subgraph isomorphism problem in some special cases. Here an *isomorphism* ϕ from (directed) graph G into (directed) graph H is a one-to-one mapping between vertices of G and H such that for each pair $u, v \in V(G)$, $\{u, v\} \in E(G)$ ($(u, v) \in E(G)$) if and only if $\{\phi(u), \phi(v)\} \in E(H)$ ($(\phi(u), \phi(v)) \in E(H)$). For a set $S \subseteq V(G)$, we define $\phi(S) = \bigcup_{v \in S} \phi(v)$. A (directed) graph G is *isomorphic* to a (directed) graph H if and only if there is an isomorphism ϕ from G into H such that $\phi(G) = V(H)$. A graph G is *subgraph isomorphic* to H if there is a subgraph H' of H which is isomorphic to G. A graph G is *induced subgraph isomorphic* to H if there exists an induced subgraph G' of H isomorphic to G.

The set of components of a graph G is represented by $\mathcal{C}(G)$, where each element of $\mathcal{C}(G)$ is a connected graph. For a set $\mathcal{D} \subseteq \mathcal{C}(G)$, we denote the set of vertices which appear in a component of $\mathcal{D}$ by $V(\mathcal{D})$ and the set of edges by $E(\mathcal{D})$. The graph resulting from removal of a set S of vertices and all adjacent edges from G is denoted by $G[V - S]$. A set S is called a *separator* if $|\mathcal{C}(G[V - S])| > 1$. For $k > 0$, graph G is called *k-connected* if every separator has size at least k.

The notion of treewidth was introduced by Robertson and Seymour [21] and plays an important role in their fundamental work on graph minors. To define this notion, first we consider the representation of a graph as a tree, which is the basis of our algorithms.

Definition 1. *[21] A* tree decomposition *of a graph $G = (V, E)$, denoted by $TD(G)$, is a pair (χ, T) in which $T = (I, F)$ is a tree and $\chi = \{\chi_i | i \in I\}$ is a family of subsets of $V(G)$ such that:* **1.** $\bigcup_{i \in I} \chi_i = V$; **2.** *for each edge $e = \{u, v\} \in E$ there exists an $i \in I$ such that both u and v belong to χ_i; and* **3.** *for all $v \in V$, the set of nodes $\{i \in I | v \in \chi_i\}$ forms a connected subtree of T.*

To distinguish between vertices of the original graph G and vertices of T in $TD(G)$, we call vertices of T *nodes* and their corresponding χ_i's *bags*. We define the *terminal subgraph* $G_{[z]}$ for a node z of $TD(G)$ to be the induced subgraph of G over vertices of χ_z and bags of descendants of z in $TD(G)$. The maximum size of a bag in $TD(G)$ minus one is called the *width* of the tree decomposition. The *treewidth* of a graph G ($tw(G)$) is the minimum width over all possible tree decompositions of G. A graph G is a *partial k-tree* if G has treewidth at most k (van Leeuwen [19]).

The *bounded fragmentation* property plays an important role in our results.

Definition 2. *A graph G is a* $(k, g(k, n))$-bounded fragmentation *graph if $|\mathcal{C}(G[V - S])| \leq |g(k, n)|$ for every $S \subseteq V(G)$ of size at most k, where g is a function of k and n. A graph G is a* totally $g(k, n)$-bounded fragmentation *graph if it is a*

$(k, g(k,n))$-*bounded fragmentation graph for all* $0 \leq k \leq n$. *A graph* G *is a* k-log-bounded fragmentation *graph (or just* log-bounded fragmentation *graph if it is clear from context) if* G *is a* $(k, O(k \log n))$-*bounded fragmentation graph. Finally a graph* G *is a* totally log-bounded fragmentation *graph if it is a* k-log-*bounded fragmentation graph for all* $0 \leq k \leq n$.

We present here two classes of log-bounded fragmentation graphs. The proofs of these lemmas are clear and hence omitted.

Lemma 1. *Connected graphs with maximum degree* $O(\log n)$ *are totally* log-*bounded fragmentation graphs.* □

Definition 3. Covering a graph by at most m vertex-disjoint paths *means the vertices of a graph can be partitioned into* m *subsets such that for each set* S, *there exists a path in a graph that contains exactly the vertices in* S.

Lemma 2. *Graphs whose vertices can be covered by at most* $O(\log n)$ *vertex-disjoint paths are totally* log-*bounded fragmentation graphs.* □

Example 1. Consider a Hamiltonian graph F_n which is constructed from a path of length n by connecting one of its vertices to all its non-neighbors. Since vertices of every Hamiltonian graph can be covered by one path, F_n is a totally $(k+1)$-bounded fragmentation graph. The reader is referred to Hajiaghayi [18] for further discussion on bounded fragmentation graphs.

3 General dynamic programming approach for graphs of bounded treewidth

Many NP-complete problems have linear-time or polynomial-time algorithms when they are restricted to graphs of bounded treewidth. There are a few techniques for obtaining such algorithms. The main technique is called *computing tables of characterizations of partial solutions.* This technique is a dynamic programming approach, first introduced by Arnborg and Proskurowski [3]. Bodlaender [7] described a better presentation of this technique. This technique is as follows. First w.l.o.g. we can assume that the given tree decomposition of G ($TD(G)$) is a *nice tree decomposition* [8], which is a rooted binary tree and has three types of nodes: **1.** a **leaf** node which does not have any children; **2.** a **separator** node which has one child, and whose bag is a subset of its child's bag; and **3.** a **join** node which has two children, and whose bag is the union of its children's bags.

Bodlaender [6] found a linear-time algorithm for construction of a tree decomposition of width at most k for a partial k-tree. He also proved that for every graph G of treewidth at most k, if a tree decomposition of G of width k is given, we can transform it into a nice tree decomposition of the same width with $O(k \cdot |V(G)|)$ nodes in linear time [8]. Because of these linear-time construction and linear-time transformation, we assume that we have a nice tree decomposition of G throughout this paper.

Using a nice tree decomposition, the rest of the algorithm is as follows. We compute for each node z of $TD(G)$ a certain table. To compute this table for node z, we only use the tables already constructed for its children (if they exist) and the structure of G restricted to the bag of z (χ_z). We perform this computation in a bottom-up fashion. To solve the original problem, we inspect the table of the root r of $TD(G)$. The reader is referred to the original papers [3, 7] for further information.

4 Subgraph isomorphism for log-bounded fragmentation graphs

As mentioned in the introduction, the subgraph isomorphism problem can be solved in polynomial time when the source graph has bounded degree and the host graph has bounded treewidth. In this section, we extend this result to cover bounded fragmentation.

Matoušek and Thomas [20] proved the following theorem for subgraph isomorphism of bounded degree graphs:

Theorem 1. *(Theorem 5.14 [20]) Suppose graph* G *is connected,* $\Delta(G) \leq c$ *for constant* c, *and* H *is a partial* k-*tree. There are* $O(|V(G)|^{k+1} \cdot |V(H)|)$-*time algorithms which solve isomorphism and its subgraph and induced subgraph versions.* □

We will prove the following extended version of Theorem 1, which includes wider classes of graphs than bounded degree graphs.

Theorem 2. *Suppose G is a $(k+1, g(k+1,n))$-bounded fragmentation graph and H is a partial k-tree for $k \geq 2$. There are $O(g(k+1,n) \cdot 2^{2g(k+1,n)}|V(G)|^{k+1} \cdot |V(H)|)$-time algorithms which solve isomorphism and subgraph and induced subgraph versions of this problem.*

Corollary 1. *Testing graph isomorphism and its subgraph or induced subgraph versions have polynomial-time solutions when graph H has bounded treewidth and graph G is a* log*-bounded fragmentation graph.* □

Using Corollary 1, we obtain two new results. First, by Lemma 1, if the maximum degree of the source graph is bounded by $O(\log n)$ (and not necessarily a constant), then it is a totally log-bounded fragmentation graph and hence the problems can be solved in polynomial time. Second, the subgraph isomorphism problem can be solved in polynomial time for graphs other than bounded degree partial k-trees or k-connected partial k-trees. To justify this result, we consider the graph F_n (Example 1) from the class of bounded fragmentation graphs. The maximum degree of this graph is $n-1$, and its treewidth is two. If the source graph is F_n and the host graph is an arbitrary graph of bounded treewidth, then the current results can not be applied to test subgraph isomorphism for these graphs. However, by Corollary 1, we can solve the problem for these graphs in polynomial time. In addition, log-bounded fragmentation graphs introduced in Lemma 2 are not necessarily connected, but still we can solve the problems for them.

We note that there are properties other than those introduced in Lemmas 1 and 2 which guarantee a graph to be a bounded fragmentation graph, but they do not apply to partial k-trees. For example, as it was shown [18], graphs with a certain minimum degree are bounded fragmentation graphs, but this minimum degree is valid only for graphs with quadratic numbers of edges. Since all partial k-trees have linear numbers of edges [22], the class of graphs to which the result applies is empty.

We now prove Theorem 2. First, we present our proof for the induced subgraph isomorphism problem. Then, we explain how our proof can be applied to the subgraph isomorphism problem. We note that if $|V(G)| = |V(H)|$, then the induced subgraph isomorphism problem is identical to the graph isomorphism problem. Here, we use the dynamic programming on a tree decomposition introduced in Section 3. We use notions such as *solution*, *partial solution*, *characteristic* and *full set of characteristics* introduced by Bodlaender [7].

As mentioned in Section 3, Bodlaender proved that for every graph H of treewidth at most k, a nice tree decomposition of width k can be constructed in linear time [8]. Using this result, we construct a nice tree decomposition of H in $O(|V(H)|)$ time and in the rest of this paper, we assume that the given tree decomposition of H ($TD(H)$) is a nice tree decomposition.

The solution for the induced subgraph isomorphism problem is an isomorphism ϕ from G into H. To define a partial solution, we consider the possible structure of an isomorphism ϕ restricted to $H_{[z]}$ for a node z of $TD(H)$. Intuitively, this restriction maps a subgraph G' of G into $H_{[z]}$; the vertices of G' are those vertices of G whose images are in $V(H_{[z]})$, and the edges of G' are edges of G images of whose end-vertices are adjacent in $H_{[z]}$. This mapping is an isomorphism φ from G' into $H_{[z]}$ such that $\varphi(v) = \phi(v)$ for $v \in G'$, and we call it a *partial isomorphism*.

Here we show that the subgraph G' has a special structure and can not be an arbitrary subgraph of G. To this end, we introduce the vertex set and the edge set of G'. Again, we consider the isomorphism ϕ from G into H from which the partial isomorphism φ is obtained. We suppose $S = \{v \in V(G) | \phi(v) \in \chi_z\}$ and $\mathcal{C}(G[V-S]) = \{C_1, \cdots, C_h\}$. Since each graph isomorphic to a connected graph is connected, $\phi(V(C_i))$, $1 \leq i \leq h$, is a connected subgraph of H. As χ_z is a separator for H, and $\phi(V(C_i))$ is a connected subgraph of H which does not intersect χ_z, each $\phi(V(C_i))$ is completely inside of $H_{[z]}$ or completely outside of $H_{[z]}$. Let $\mathcal{D} = \{D_1, \cdots, D_l\}$ be those components of $\mathcal{C}(G[V-S])$ whose images are completely inside $H_{[z]}$ (the set S and components D_i, $1 \leq i \leq l$, are shown in Figure 1 in the Appendix). In fact, $S \cup V(\mathcal{D})$ is the vertex set of the subgraph G' introduced above. Thus $V(G')$ is always the union of the set $S \subseteq V(G)$ and vertex sets of a number of components of $\mathcal{C}(G[V-S])$. We now consider the edge set of G'. Since $\phi(S \cup V(\mathcal{D}))$ is a subset of $V(H_{[z]})$ and ϕ is an isomorphism form G into H, for all $u, v \in S \cup V(\mathcal{D})$ such that $\{u,v\} \in E(G)$, $\varphi(u) = \phi(u)$ and $\varphi(v) = \phi(v)$ are adjacent in $H_{[z]}$ (see Figure 1 in Appendix). Therefore G' is always an induced subgraph of G over vertices of $S \cup V(\mathcal{D})$ for $S \subseteq V(G)$ and $\mathcal{D} \subseteq \mathcal{C}(G[V-S])$ and our partial solutions are partial isomorphisms from this kind of subgraph of G into $H_{[z]}$. By this definition of a partial solution, it is the restriction of a solution to $H_{[z]}$ and naturally can be extended to a solution.

We are now ready to define the most important notion, namely a characteristic of a partial solution with respect to a node z of $TD(H)$. We define the crucial part of a partial solution which is necessary to know how a partial solution can

be extended to a solution. Here, we label arguments by properties in upcoming Definition 5. First we represent vertices and edges of G'. We specify $V(G')$ by S and $\mathcal{D}$. Two sets S and $\mathcal{D}$ also uniquely determine edges of G' (Property [Pc]). For each vertex $v \in V(\mathcal{D})$, $\varphi(v) = \phi(v)$ is inside of $V(H_{[z]}) - \chi_z$ and all neighbors of v in G are in $S \cup V(\mathcal{D})$. Since images of v and all its neighbors are present in $H_{[z]}$, intuitively, this vertex v is not a crucial vertex of G' and there is no need to know how its image is exactly mapped in $H_{[z]}$. Instead, we need to know more information about images of vertices of S. Thus we maintain a mapping $\psi(v) = \varphi(v)$ for $v \in S$ (Property [Pb]). We also note that since $\phi(v)$ is an isomorphism and $\varphi(v)$ is obtained from $\phi(v)$, $\varphi(u) \neq \varphi(v)$ for all $u, v \in S \cup V(\mathcal{D})$ (Property [Pa]). As we will formally show later, triples $(S, \mathcal{D}, \psi)$, called iso-triples, are our characteristics of partial solutions holding all the information that we maintain about our partial solutions. We note that not each triple $(S, \mathcal{D}, \psi)$ is necessarily a characteristic. We call a partial isomorphism φ from which a characteristic $(S, \mathcal{D}, \psi)$ is obtained an *extension* of this characteristic. We note that the characteristic $(S, \mathcal{D}, \psi)$ might be obtained from several partial isomorphisms and thus have several extensions. Let us define all these terms formally.

Definition 4. *An* iso-triple ξ of G into H relative to a node z of $TD(H)$ *is a triple* $(S, \mathcal{D}, \psi)$ *where:* **1.** $S \subseteq V(G)$*;* **2.** $\mathcal{D} \subseteq \mathcal{C}(G[V - S])$*; and* **3.** ψ *is a one-to-one mapping from* S *into* χ_z*.*

Definition 5. *An* extension φ of an iso-triple $\xi = (S, \mathcal{D}, \psi)$ relative to a node z of $TD(H)$ *is a mapping* φ *from* $S \cup V(\mathcal{D})$ *into* $H_{[z]}$ *with these properties:* [**Pa**] $\varphi(u) \neq \varphi(v)$ *for all* $u, v \in S \cup V(\mathcal{D})$*;* [**Pb**] $\varphi(v) = \psi(v)$ *for* $v \in S$ *and* $\varphi(v) \notin \chi_z$ *for* $v \in V(\mathcal{D})$*; and* [**Pc**] *for each* $u, v \in S \cup V(\mathcal{D})$*,* $\{u, v\} \in E(G)$ *if and only if* $\{\varphi(u), \varphi(v)\} \in E(H)$*.*

We note that all conditions in Definition 5 are formal descriptions of properties of an extension (or a partial isomorphism) corresponding to an iso-triple (if it exists).

Definition 6. *A* characteristic of a partial isomorphism (CPI) ξ of G into H relative to a node z of $TD(H)$ *is an iso-triple* $(S, \mathcal{D}, \psi)$ *which has an extension* φ *(not necessarily unique).*

According to the general dynamic programming approach, we need to identify the full set of characteristics which contains all CPIs relative to a node z of $TD(H)$. This set can be represented by an array indexed by all iso-triples $(S, \mathcal{D}, \psi)$, namely a *full set array*. If an iso-triple $(S, \mathcal{D}, \psi)$ is a CPI its corresponding element in the array is **true**, otherwise it is **false**. Later, we will show how the full set of a node can be built from the full set of its children (if they exist). Now, we show that the size of a full set is polynomial.

Lemma 3. *The number of all iso-triples and the number of CPIs relative to a node* z *of* $TD(H)$ *is in* $O((k+1)! \cdot 2^{g(k+1,n)} \cdot |V(G)|^{k+1})$.

Finally, we state how the problem can be solved efficiently, if we know the full set of CPIs relative to the root r of $TD(H)$.

Definition 7. *A* complete CPI ξ of G into H relative to a node z of $TD(H)$ *is a CPI* $(S, \mathcal{D}, \psi)$ *such that* $\mathcal{D} = \mathcal{C}(G[V - S])$ *(S can be empty).*

Lemma 4. *There exists an induced subgraph isomorphism* ϕ *from* G *into* H *if and only if there exists a complete CPI* ξ *of* G *into* H *relative to the root* r *of* $TD(H)$*.*

It only remains to show how full set arrays of nodes of $TD(H)$ can be filled in. As $H_{[z]} = H[\chi_z]$ for a leaf z and hence $\mathcal{D}$ is empty, ψ is equal to its extension φ. Thus for each iso-triple ξ relative to a leaf, by brute force, we can easily check whether ψ is an extension of ξ or not and construct the full set array (see Algorithm A for further detail). For other nodes, we use Lemmas 5 and 6. First, we consider how the full set array of a separator node z can be constructed from the full set array of its child z'.

Definition 8. *Suppose an iso-triple* $\xi = (S, \mathcal{D}, \psi)$ *relative to a separator node* z *of* $TD(H)$ *and an iso-triple* $\xi' = (S', \mathcal{D}', \psi')$ *relative to the child* z' *of* z *satisfy following conditions: 1.* $S = \{v \in S' | \psi'(v) \in \chi_z\}$*; 2.* $\mathcal{D}' = \{D' \in \mathcal{C}(G[V - S']) | D'$ *is a subgraph of some* $D \in \mathcal{D}\}$*; and 3.* $\psi(v) = \psi'(v)$ *for* $v \in S$*. Then, we say* ξ *is* separator-consistent *with* ξ'*.*

The conditions stated in Definition 8 indicate how elements of a CPI of a separator node, *i.e.* $S, \mathcal{D}$ and ψ, are related to elements of a CPI of the child of this node. We show later (in Algorithm A) how we can use these conditions for obtaining a CPI of a separator node z from a CPI of its child z'.

Lemma 5. *There exists a CPI $\xi = (S, \mathcal{D}, \psi)$ relative to a separator node z of $TD(H)$ if and only if there exists a CPI $\xi' = (S', \mathcal{D}', \psi')$ relative to the child z' of z such that ξ and ξ' are separator-consistent.*

Finally, we consider how the full set array for a join node z can be built from the full set arrays of its children z_1 and z_2.

Definition 9. *Let z be a join node of $TD(H)$ and its children be z_1 and z_2. Suppose an iso-triple ξ relative to z and iso-triples ξ_1 and ξ_2 relative to z_1 and z_2 satisfy following conditions:* **1.** *$S_i = \{v \in S | \psi(v) \in \chi_{z_i}\}$ for $i = 1, 2$;* **2.** *the components of $\mathcal{D}$ are partitioned into $\mathcal{D}_1$ and $\mathcal{D}_2$;* **3.** *$\psi_i(v) = \psi(v)$ for $v \in S_i$ for $i = 1, 2$; and* **4.** *for $u, v \in S$, $\{u, v\} \in E(G)$ if and only if $\{\psi(u), \psi(v)\} \in E(H)$. Then, we say ξ is* join-consistent *with ξ_1 and ξ_2.*

The conditions stated in Definition 9 specify how elements of a CPI of a join node are related to elements of CPIs of its children. We demonstrate in Algorithm A how these conditions can be used for obtaining a CPI of a join node z from CPIs of its children z_1 and z_2.

Lemma 6. *Let z be a join node of $TD(H)$ and its children be z_1 and z_2. There exists a CPI $\xi = (S, \mathcal{D}, \psi)$ relative to z if and only if there exist CPIs $\xi_i = (S_i, \mathcal{D}_i, \psi_i)$ relative to node z_i $(i = 1, 2)$ such that ξ is join-consistent with ξ_1 and ξ_2.*

Proof. (sketch) We prove that if there exists a CPI ξ relative to z, then there exist CPIs ξ_i relative to z_i such that ξ is join-consistent with ξ_1 and ξ_2. We suppose φ is an extension of ξ over χ_z. We are able to obtain ξ_1 and ξ_2 from φ and show that they satisfy conditions of join-consistency.

For the other side, we show if there exist CPIs ξ_i relative to z_i such that an iso-triple ξ is join-consistent with ξ_1 and ξ_2, then ξ is a CPI relative to z. We define a mapping φ for iso-triple ξ from ψ and extensions φ_i's of ψ_i's as follows: for $v \in V(\mathcal{D}_i)$, $\varphi(v) = \varphi_i(v)$ and for $v \in S$, $\varphi(v) = \psi(v)$. We note that each vertex $v \in V(\mathcal{D})$ is either in $V(\mathcal{D}_1)$ or in $V(\mathcal{D}_2)$ (Condition (2)). We now observe that φ is an extension of ξ (See Appendix for further detail). □

We are now ready to present our final algorithm. Lines 1-14 of the algorithm are preprocessing steps. In this algorithm, first we construct $\mathcal{C}(G[V-S])$ and tables *Subs* and *Sup* (lines 3-10) which are used for dealing with components of $G[V-S]$ instead of vertices of $G[V-S]$. Throughout this algorithm, we consider each component as an entity and we label it by its lowest numbered vertex in an arbitrary ordering of vertices of G fixed at beginning. We also compute all iso-triples and maintain them in the set *AllIS* (lines 11-14). Although an iso-triple is defined relative to one node, since elements S and $\mathcal{D}$ of iso-triples are the same relative to different nodes we define one set of iso-triples to all nodes. In fact, we assume ψ is a one-to-one mapping from S into set $\{1, 2, \cdots, k+1\}$ (line 13) and for each node z of $TD(H)$, we order vertices of χ_z arbitrarily and use this ψ as a mapping from S into χ_z. Using the fact that for a CPI ξ relative to a leaf, ψ is equal to the extension of ξ, we fill in full set arrays of leaves (lines 19-22). We find all CPIs of a node from CPIs of its children within two procedures *BuildFSAofSeparatorNode* and *BuildFSAofJoinNode*. Finally, we check whether or not there exists a complete CPI relative to the root r of $TD(H)$ (lines 27-28).

Algorithm A: testing induced subgraph isomorphism
Input: G : a $(k+1, g(k+1, n))$-bounded fragmentation graph
$TD(H)$: a nice tree decomposition of a partial k-tree H
Output: **true** if G is an induced subgraph isomorphic to H, **false** otherwise
Variables:
$Subs[S, S', C']$: specifies components of $G[V-S]$ which are contained in $C' \in \mathcal{C}(G[V-S'])$, where $S' \subset S$
$Sup[S, S', C]$: specifies a component of $G[V-S']$ which contains $C \in \mathcal{C}(G[V-S])$, where $S' \subset S$
$FSA[z, \xi]$: **true** if ξ is in the full set of node z, **false** otherwise
$AllIS$: A set containing all iso-triples
begin
1 **if** $|V(G)| > |V(H)|$ **return false**;
2 **let** $\alpha_1, \alpha_2, \cdots, \alpha_{|TD(H)|}$ be the reverse of breadth first search order of nodes of $TD(H)$;
3 **for** each set S of at most $k+1$ vertices of G

```
4       find C(G[V − S])
5       for each set S' ⊂ S
6           find C(G[V − S'])
7           for each component C'_i ∈ C(G[V − S'])
8               let Subs[S, S', C'_i] ← {C_{j_1}, ··· C_{j_h}} such that V(C_{j_l}) ⊆ V(C'_i), 1 ≤ l ≤ h
9           for each component C_i ∈ C(G[V − S])
10              let Sup[S, S', C_i] ← C'_j such that V(C_i) ⊆ V(C'_j)
11  for each S ⊆ V(G) of size at most k + 1
12      for each D ⊆ C(G[V − S])
13          for each one-to-one mapping ψ from S into set {1, 2, ··· , k + 1}
14              let AllIS ← AllIS ∪ {ξ = (S, D, ψ)}
15  for each node α_i of TD(H), i from 1 to |TD(H)|
16      for each iso-triple ξ = (S, D, ψ) in AllIS
17          FSA[α_i, ξ] ← false;
18      if α_i is a leaf node
19          for each iso-triple ξ in AllIS
20              let φ ← ψ;
21              if D = ∅ and
                for each u, v ∈ S, {u, v} ∈ E(G) if and only if {φ(u), φ(v)} ∈ E(H)
22                  let FSA[α_i, ξ] ← true;
23      else if α_i is a separator node
24          BuildFSAofSeparatorNode(α_i);
25      else if α_i is a join node
26          BuildFSAofJoinNode(α_i);
27  for each iso-triple ξ = (S, D, ψ) in AllIS
28      if FSA[root, ξ] = true and D = C(G[V − S]) return true;
29  return false;
end
```

In procedure *BuildFSAofSeparatorNode*, we find all CPIs of a separator node z from CPIs of its child z'. For each CPI ξ' relative to z' (line 51), we construct an iso-triple ξ relative to z which is separator-consistent with ξ' and thus is a CPI relative to z (lines 52-60). Condition (1) of separator-consistency between ξ and ξ' is checked in line 52, Condition (3) is tested in line 53 and Condition (2) is checked in lines 54-60. To test Condition (2), we use additional set $\mathcal{D}''$. First we find all supergraphs of components $C' \in \mathcal{D}'$ (line 56). Since a component $D \in \mathcal{D}$ is a supergraph of components in $\mathcal{D}'$ and not a supergraph of components in $\mathcal{C}(G[V - S']) - \mathcal{D}'$ (Condition (2) of separator-consistency), we again find all components of $\mathcal{C}(G[V - S'])$ which are contained in components $D \in \mathcal{D}$ (line 59) to make sure $\mathcal{D}'$ and $\mathcal{D}''$ are equal (line 59).

BuildFSAofSeparatorNode(z)

```
Input:  z   : a separator node of TD(H)
begin
50  let z' ← the child of z;
51  for each iso-triple ξ' = (S', D', ψ') in AllIS such that FSA[z', ξ'] is true
52      let S ← {v ∈ S' | ψ'(v) ∈ χ_z};
53      let ψ(v) ← ψ'(v) for v ∈ S;
54      D ← ∅;
55      for each C' ∈ D'
56          let D ← D ∪ {Sup[S', S, C']};
57      D'' ← ∅;
58      for each C ∈ D
59          let D'' ← D'' ∪ Subs[S', S, C];
60      if D'' = D'
61          let ξ ← (S, D, ψ);
62          let FSA[z, ξ] ← true;
end
```

In procedure *BuildFSAofJoinNode*, we find all CPIs of a join node z from CPIs of its children z_1 and z_2. For each iso-triple ξ relative to z, we construct all iso-triples ξ_1 and ξ_2 relative to its children which are join-consistent with ξ (lines 77-85). If ξ_1 and ξ_2 are CPI then ξ is also a CPI (lines 86-87). Condition (1) of join-consistency is checked in line 77, Condition (3) is tested in line 78, Condition (4) is checked in line 79 and finally Condition (2) is checked in lines 80-83. We partition elements of $\mathcal{D}$ into $\mathcal{D}_1$ and $\mathcal{D}_2$ (line 80), but we still have to make sure that each element $D \in \mathcal{D}$, which is in $\mathcal{D}_i$, is a component of $G[V - S_i]$. Since all outgoing edges from $V(D)$ go into S, we only check whether there is any edge between $V(D)$ and $S - S_i$. If there is such edge, then the component $D' \in \mathcal{C}(G[V - S_i])$ which is a supergraph of D has at least one vertex in $S - S_i$. This condition is checked in line 83.

BuildFSAofJoinNode(z)

Input: z : a join node of $TD(H)$
begin
75 **let** $z_i \leftarrow i$th child of z, $i = 1, 2$;
76 **for** each iso-triple ξ in $AllIS$
77 **let** $S_i \leftarrow \{v \in S | \psi(v) \in \chi_{z_i}\}$, $i = 1, 2$;
78 **let** $\psi_i(v) \leftarrow \psi(v)$ for $v \in S_i$, $i = 1, 2$;
79 **if** for all $u, v \in S$, $\{u, v\} \in E(G)$ if and only if $\{\psi(u), \psi(v)\} \in E(H)$
80 **for** each partition of elements of $\mathcal{D}$ into $\mathcal{D}_1$ and $\mathcal{D}_2$
81 **let** *bool* $\leftarrow$ **true**;
82 **for** each $D \in \mathcal{D}$, which is in $\mathcal{D}_i$
83 **if** $V(Sup[S, S_i, D]) \cap S - S_i \neq \emptyset$ **let** *bool* $\leftarrow$ **false**;
84 **if** *bool* = **true**
85 **let** $\xi_i \leftarrow (S_i, \mathcal{D}_i, \psi_i)$, $i = 1, 2$;
86 **if** $FSA[z_1, \xi_1] = FSA[z_2, \xi_2]$ =**true**
87 **let** $FSA[z, \xi] \leftarrow$ **true**;
end

To prove the correctness of the algorithm and obtaining the running time, first we define an invariant and present two lemmas for separator and join nodes.

Definition 10. *We say that the FSA array relative to a node z of $TD(H)$ is in* correct form*, if for each ξ relative to z, $FSA[z, \xi] =$ **true** if and only if ξ is a CPI relative to z.*

Lemma 7. *Given the FSA array of the child z' of a separator node z in correct form, Procedure* BuildFSAofSeparatorNode *fills in the FSA array relative to z in correct form in $O(g(k+1, n) \cdot 2^{g(k+1,n)} |V(G)|^{k+1})$ time.*

Lemma 8. *Given the FSA arrays of children z_1 and z_2 of a join node z in correct form, Procedure* BuildFSAofJoinNode *fills in the FSA array relative to z in correct form in $O(g(k+1, n) \cdot 2^{2g(k+1,n)} |V(G)|^{k+1})$ time.*

We are now ready to prove the correctness and compute the running time of the whole algorithm.

Theorem 3. *The above algorithm solves the induced subgraph isomorphism problem in $O(g(k+1, n) \cdot 2^{2g(k+1,n)} |V(G)|^{k+1} \cdot |V(H)|)$ time.*

Proof. (sketch) The proof follows from our previous discussion and Lemmas 3, 4, 7 and 8. □

We can also solve the subgraph isomorphism problem using the above algorithm. First, we present some definitions and lemmas.

Definition 11. *The* subdivision *of an edge $\{u, w\}$ is the operation of deleting this edge and adding a new vertex v and two new edges $\{u, v\}$ and $\{v, w\}$. The graph obtained from graph G by subdivisions of all its edges is denoted by G^*.*

Lemma 9. *If G is a partial k-tree and $k \geq 2$, then G^* is a partial k-tree.*

Lemma 10. *If G is a $(k, g(k, n))$-bounded fragmentation graph and k is a constant, G^* is a $(k, O(g(k, n)))$-bounded fragmentation graph.*

Lemma 11. *(Lemma 1.4 [20]) Let G and H be two graphs. G is subgraph isomorphic to H if and only if G^* is induced subgraph isomorphic to H^*.* □

By Lemmas 9, 10 and 11, we can solve the subgraph isomorphism problem by means of induced subgraph versions. Here we finish the proof of Theorem 2.

By considering directions of edges in consistency checking, we can generalize the algorithm mentioned in the proof of Theorem 2 to directed graphs (we replace edge $\{u, v\}$ by (u, v) in lines 21 and 79). Hence if G is a directed graph with a $(k+1, g(k+1, n))$-bounded fragmentation underlying graph and H is a directed graph, whose underlying graph is a partial k-tree, then there are $O(g(k+1, n) \cdot 2^{2g(k+1,n)} |V(G)|^{k+1} \cdot |V(H)|)$-time algorithms which solve isomorphism, subgraph isomorphism and induced subgraph isomorphism.

It is worth mentioning that all current algorithms for constructing a tree decomposition of a graph of treewidth at most k have a hidden constant factor that is at least exponential in k, and hence they are impractical in general. However, in the cases $k = 2, 3, 4$, practical linear-time algorithms exist [2, 20, 23] and for these cases, our polynomial-time algorithm presented in this section is practical.

5 Extension to graphs of locally bounded treewidth

In this section, we extend our polynomial-time algorithm for testing subgraph isomorphism to graphs of locally bounded treewidth. Eppstein [13] introduced the notion of *bounded local treewidth*, which is a generalization of the notion of treewidth.

Definition 12. *The* local treewidth *of a graph G is the function* $\mathrm{ltw}^G : \mathbb{N} \to \mathbb{N}$ *that associates with every $r \in \mathbb{N}$ the maximum treewidth of an r-neighborhood in G. We set* $\mathrm{ltw}^G(r) = \max_{v \in V(G)}\{\mathrm{tw}(G[N_G^r(v)])\}$*, and we say that a graph class $\mathcal{C}$ has* bounded local treewidth (or locally bounded treewidth) *when there is a function $f : \mathbb{N} \to \mathbb{N}$ such that for all $G \in \mathcal{C}$ and $r \in \mathbb{N}$, $\mathrm{ltw}^G(r) \leq f(r)$.*

Theorem 4. *Subgraph and induced subgraph isomorphism can be solved in* $O((ltw(diam(G)) \log |V(G)|) \cdot 2^{O(ltw(diam(G)) \log |V(G)|)} |V(G)|^{ltw(diam(G))+1} \cdot |V(H)|)$ *time when graph H is of minor-closed family of graphs of locally bounded treewidth and graph G is a totally* log*-bounded fragmentation graph and has constant diameter.*

Proof. The idea of the proof follows Baker's idea [5]. For graph H and integers $0 \leq i \leq j$, we let $L^H[i,j] = \bigcup_{i \leq k \leq j} L_k$, where L_k (the kth layer) consists of all vertices at distance k from a fixed vertex v.

We suppose d is the number of layers in graph H. For $0 \leq i \leq d - diam(G) + 1$, let $L_{i,diam(G)} = L^H[i, i + diam(G) - 1]$ and let $H[L_{i,diam(G)}]$ be the induced graph on vertices in $L_{i,diam(G)}$. Eppstein proved for graph h, $tw(H[L_{i,diam(G)}]) \leq ltw(diam(G))$ and thus $H[L_{i,diam(G)}]$ has bounded treewidth [13]. We can construct a tree decomposition of each graph of bounded treewidth in linear time [6].

If G is (induced) subgraph isomorphic to H, then it is contained in one of the graphs $H[L_{i,diam(G)}]$. Using Corollary 1, we can solve graph isomorphism separately for each $H[L_{i,diam(G)}]$. The desired running time follows from Theorem 2 and the fact that each vertex of H has participated in at most $diam(G)$ iterations of the algorithm. □

The point that the source graph G has bounded diameter is very important and without it, the problem remains NP-complete.

Theorem 5. *Let graph H have locally bounded treewidth with $\Delta(H) = 3$ and graph G have bounded treewidth with $\Delta(G) = 2$. The subgraph isomorphism problem for the source graph G and the host graph H is NP-complete.*

One application of Theorem 4 is that it gives a linear-time algorithm for testing subgraph isomorphism for fixed patterns. More precisely:

Corollary 2. *For a fixed pattern G and a graph H of minor-closed family of graphs of locally bounded treewidth, subgraph isomorphism and induced subgraph isomorphism can be tested in $O(|V(H)|)$ time.*

Corollary 2 is the same as Eppstein's result for testing subgraph isomorphism of fixed patterns on graphs of locally bounded treewidth [12, 13]. Using this result, Eppstein also showed that other problems such as finding diameter (if we know the graph has bounded diameter), h-clustering for constant h and finding girth (if we know the graph has bounded girth) can be tested in $O(n)$ time for these graphs. The reader is referred to the original papers for details.

6 Conclusions and future work

In this paper, we discussed the subgraph isomorphism problem. We presented a polynomial-time algorithm for this problem when the source graph is *log*-bounded-fragmentation and the host graph is bounded treewidth. In addition, we demonstrated how this algorithm can be generalized to graphs of locally bounded treewidth. Here, we present some open problems that can be considered as possible extensions of this paper.

Gupta and Nishimura proved that when both the pattern graph G and the source graph H are k-connected partial k-trees, the subgraph isomorphism problem has a polynomial-time solution [16]. In fact, their polynomial-time algorithm applies when both G and H are k-connected and only H is a partial k-tree. The algorithm for k-connectivity uses a different approach from that for bounded fragmentation. An open problem is how the result can be generalized when H

has locally bounded treewidth. Solving the minor containment problem by adding more restrictions when H has locally bounded treewidth is another open problem.

Another problem related to subgraph isomorphism is the problem of finding the largest common subgraph of two graphs. Brandenburg [10] showed that if two graphs G and H are k-connected partial k-trees, the problem of finding the largest common k-connected subgraph can be solved in polynomial time. In addition, using Brandenburg's ideas [10] and Gupta and Nishimura's ideas [17], one can observe that finding the largest common $(k-1)$-connected subgraph is NP-complete. It would be interesting to know whether or not the result can be generalized to bounded fragmentation graphs.

Acknowledgments. I would like to thank professor Jim Geelen and professor Naomi Nishimura at University of Waterloo for their thoughtful comments.

References

1. Stefan Arnborg, Jens Lagergren, and Detlef Seese. Problems easy for tree-decomposable graphs (extended abstract). In *Automata, languages and programming (Tampere, 1988)*, pages 38–51. Springer, Berlin, 1988.
2. Stefan Arnborg and Andrzej Proskurowski. Characterization and recognition of partial 3-trees. *SIAM J. Algebraic Discrete Methods*, 7(2):305–314, 1986.
3. Stefan Arnborg and Andrzej Proskurowski. Linear time algorithms for NP-hard problems restricted to partial k-trees. *Discrete Appl. Math.*, 23(1):11–24, 1989.
4. Peter J. Artymiuk, Peter A. Bath, Hans M. Grindley, Carine A. Pepperrell, Andrzej R. Poirrette, Dan W. Rice, Daniel A. Thorner, Douglas J. Wild, Peter Willet, Frank H. Allen, and Robert Taylor. Similarity searching in databases of three-dimensional molecules and macromolecules. *J. Chemical Information and Computer Sciences*, 32:617–630, 1992.
5. Brenda S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *J. Assoc. Comput. Mach.*, 41(1):153–180, 1994.
6. Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.
7. Hans L. Bodlaender. Treewidth: algorithmic techniques and results. In *Mathematical foundations of computer science 1997 (Bratislava, 1997)*, pages 19–36. Springer, Berlin, 1997.
8. Hans L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoret. Comput. Sci.*, 209(1-2):1–45, 1998.
9. John A. Bondy and U. S. R. Murty. *Graph Theory with Applications.* American Elsevier Publishing Co., Inc., New York, 1976.
10. Franz J. Brandenburg. Pattern matching problems in graphs. Manuscript, 2001.
11. Anders Dessmark, Andrzej Lingas, and Andrzej Proskurowski. Faster algorithms for subgraph isomorphism of k-connected partial k-trees. *Algorithmica*, 27(3-4):337–347, 2000.
12. David Eppstein. Subgraph isomorphism in planar graphs and related problems. *J. Graph Algorithms Appl.*, 3:no. 3, 27 pp. (electronic), 1999.
13. David Eppstein. Diameter and treewidth in minor-closed graph families. *Algorithmica*, 27(3-4):275–291, 2000.
14. Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness.* W. H. Freeman and Co., San Francisco, Calif., 1979.
15. Michael R. Garey, David S. Johnson, and Robert E. Tarjan. The planar hamiltonian circuit problem is NP-complete. *SIAM J. Comput.*, 5(6):704–714, 1976.
16. Arvind Gupta and Naomi Nishimura. Sequential and parallel algorithms for embedding problems on classes of partial k-trees. In *Algorithm theory—Scandinavian Workshop on Algorithm Theory 1994 (Aarhus, 1994)*, pages 172–182. Springer, Berlin, 1994.
17. Arvind Gupta and Naomi Nishimura. The complexity of subgraph isomorphism for classes of partial k-trees. *Theoret. Comput. Sci.*, 164(1-2):287–298, 1996.
18. MohammadTaghi Hajiaghayi. Algorithms for Graphs of (Locally) Bounded Treewidth. Master's thesis, University of Waterloo, September 2001.
19. Jan van Leeuwen. Graph algorithms. In *Handbook of theoretical computer science, Vol. A*, pages 525–631. Elsevier, Amsterdam, 1990.
20. Jiří Matoušek and Robin Thomas. On the complexity of finding iso- and other morphisms for partial k-trees. *Discrete Math.*, 108(1-3):343–364, 1992. Topological, algebraical and combinatorial structures. Frolík's memorial volume.
21. Neil Robertson and Paul D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986.
22. Donald J. Rose. On simple characterizations of k-trees. *Discrete Math.*, 7:317–322, 1974.
23. Daniel P. Sanders. On linear recognition of tree-width at most four. *SIAM J. Discrete Math.*, 9(1):101–117, 1996.
24. Maciej M. Sysło. The subgraph isomorphism problem for outerplanar graphs. *Theoret. Comput. Sci.*, 17(1):91–97, 1982.

7 Appendix (will be omitted in the conference version)

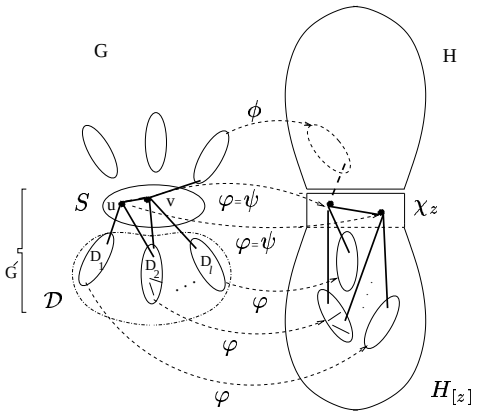

Fig. 1. Elements of the characteristic of a partial isomorphism

We often use the following property of tree decompositions, especially for designing polynomial-time algorithms on graphs of bounded treewidth.

Lemma 12. *Let $T_1, T_2, \cdots, T_p$ be subtrees of a tree decomposition of G formed by removing a node z from the tree decomposition and let V_i, $1 \leq i \leq p$, be the sets of vertices of G appear in bags of nodes of T_i except those appear in χ_z. The set χ_z is a separator for G. More precisely, after removal of χ_z from G, there is no edge between V_i and V_j, $i \neq j$.* □

Proof. of Lemma 3: Since the number of CPIs is bounded above by the number of iso-triples, it suffices to bound the number of iso-triples. By the definition of iso-triples (Point 3, Definition 4), for $v \in S$, $\psi(v)$ is an element of set χ_z. Since $|\chi_z| \leq k+1$ and the images of ψ are different for $u \neq v$ (Point 3, Definition 4), $|S| \leq k+1$ and hence there are $|V(G)|^{k+1}$ different ways to choose S. We have at most $2^{g(k+1,n)}$ choices for $\mathcal{D}$, because after choosing S, each connected component of $G[V-S]$ either belongs to $\mathcal{D}$ or does not (Point 2, Definition 4). Since $\psi(v)$ is a one-to-one mapping from S into χ_z, we have at most $|\chi_z|! \leq (k+1)!$ ways of constructing ψ ($|\chi_z|$ choices for the image of the first vertex of S, $(|\chi_z|-1)$ choices for the second one and so on). By multiplying all factors described above, we have in total at most

$$|V(G)|^{k+1} \cdot 2^{g(k+1,n)} \cdot (k+1)!$$

choices of iso-triples. Since k is a constant, the above number is in $O(2^{g(k+1,n)}|V(G)|^{k+1})$. □

Proof. of Lemma 4: If an isomorphism ϕ from G into H exists, we can construct a complete CPI by restriction of ϕ to χ_r. Formally, we let $S = \{v|\phi(v) \in \chi_r\}$; $\mathcal{D} = \mathcal{C}(G[V-S])$; and $\psi(v) = \phi(v)$ for $v \in S$. We can observe that $\varphi = \phi$ is an extension for $\xi = (S, \mathcal{D}, \psi)$. More precisely, Property [Pa] of φ follows from the fact that ϕ is a one-to-one mapping. Property [Pb] follows from definitions of S and ψ and finally Property [Pc] follows from the fact that $\varphi = \phi$ is an isomorphism. On the other hand, if a complete CPI ξ of G into H relative to the root r of $TD(H)$ exists, then the extension φ of this CPI is an isomorphism ϕ from G into H. In fact, Property [Pa] of an extension guarantees ϕ to be a one-to-one mapping and Property [Pc] of an extension together the fact that $\mathcal{D} = \mathcal{C}(G[V-S])$ guarantees ϕ to be an isomorphism from $H_{[r]} = G$ into H. □

Proof. of Lemma 5:

Since z is a separator node, we have $\chi_z \subseteq \chi_{z'}$ and $H_{[z]} = H_{[z']}$.

First, we prove if there exists a CPI ξ relative to z then there exists a CPI ξ' relative to z' such that ξ and ξ' are separator-consistent. To this end, we extend ξ to its extension φ over $H_{[z]}$. Then we obtain ξ' by restriction of φ to $\chi_{z'}$ and show that ξ' satisfies all conditions of separator-consistency. Formally, we define $S' = \{v|\varphi(v) \in \chi_{z'}\}$; $\mathcal{D}' = \{D' \in \mathcal{C}(G[V-S'])|D'$ is a subgraph of some $D \in \mathcal{D}\}$; and $\psi'(v) = \varphi(v)$ for $v \in S'$. Using this definition of ξ', we

observe that $\varphi' = \varphi$ is an extension of ξ'. First we note that by our definitions of S' and $\mathcal{D}'$, $S \cup V(D) = S' \cup V(D')$. Properties [Pa] and [Pc] of the extension φ' for ξ' follow from this fact and Properties [Pa] and [Pc] of φ for ξ. Property [Pb] of φ' follows from the definition of ψ'.

We show that all conditions of separator-consistency have been satisfied. By the definition of ψ', we have $\psi'(w) = \varphi(w)$ for $w \in S' \supseteq S$. Since by Property [Pb] of φ, $\varphi(w) = \psi(w)$ for $w \in S$ and $\chi_z \subseteq \chi_{z'}$ we have $\psi'(w) = \varphi(w) = \psi(w)$ for $w \in S$. Condition (3) is satisfied by this fact. By the definition of S' and the fact that $\psi'(w) = \varphi(w)$ for $w \in S'$, Condition (1) is also satisfied. Finally, Condition (2) is satisfied by the definition of D'.

We now prove if there exists a CPI ξ' relative to z' such that an iso-triple ξ is separator-consistent with ξ', then ξ is a CPI relative to z. We suppose there is a ξ' which satisfies the above conditions. We extend ξ' to its extension φ' over $H_{[z]}$ and show that φ' is an extension of ξ too, that is, the mapping $\varphi = \varphi'$ satisfies all properties of an extension for ξ. Since $S \subseteq S'$ (see Condition (1)), by Condition (2) we have $S \cup V(D) = S' \cup V(D')$. Thus Properties [Pa] and [Pc] of φ' follow from Property [Pa] and [Pc] of φ. By Property [Pb] of φ', we have $\varphi(v) = \varphi'(v) \notin \chi_{z'}$ for $v \in V(G) - S'$, and since $\chi_z \subseteq \chi_{z'}$, we have $\varphi(v) \notin \chi_z$ for $v \in V(G) - S'$. For $v \in S' - S$, $\varphi(v) = \varphi'(v) = \psi'(v) \notin \chi_z$ by Condition (1). By Condition (3), $\varphi(v) = \varphi'(v) = \psi'(v) = \psi(v)$ for $v \in S$. Thus Property [Pb] of φ holds. □

Proof. of Lemma 6: Throughout this proof, we assume $i = 1, 2$. Since z is a join node, we know $\chi_{z_i} \subseteq \chi_z$ and $H_{[z_i]}$ is an induced subgraph of $H_{[z]}$.

We prove that if there exists a CPI ξ relative to z, then there exist CPIs ξ_i relative to z_i such that ξ is join-consistent with ξ_1 and ξ_2. We suppose φ is an extension of ξ over χ_z. We obtain ξ_1 and ξ_2 from φ and show that they satisfy conditions of join-consistency.

We first state the following claim about elements of $\mathcal{D}$.

Claim. Let $S_i = \{v \in S | \varphi(v) \in \chi_{z_i}\}$. For each $D \in \mathcal{D}$, either $\varphi(V(D)) \subseteq V(H_{[z_1]}) - \chi_{z_1}$ and $D \in \mathcal{C}(G[V - S_1])$ or $\varphi(V(D)) \subseteq V(H_{[z_2]}) - \chi_{z_2}$ and $D \in \mathcal{C}(G[V - S_2])$.

Proof. Since by Property [Pb] of φ, $\varphi(V(D))$ is contained in $H_{[z]}$ but is disjoint from χ_z, $\chi_{z_i} \subseteq \chi_z$ and χ_z is a separator of $H_{[z]}$ (Lemma 12), either $\varphi(V(D)) \subseteq V(H_{[z_1]}) - \chi_{z_1}$ or $\varphi(V(D)) \subseteq V(H_{[z_2]}) - \chi_{z_2}$. Without loss of generality, we can assume the former case holds. We show $D \in \mathcal{C}(G[V - S_1])$. First we note that since $D \in \mathcal{D} \subseteq \mathcal{C}(G[V - S])$, all outgoing edges from $V(D)$ go into S. We now prove there is no edge between $V(D)$ and $S - S_1$, thus all outgoing edges from $V(D)$ go into S_1, and hence D is a component of $G[V - S_1]$. If there are $v \in V(D)$ and $w \in S - S_1$ such that $\{v, w\} \in E(G)$, then there exists an edge $\{\varphi(v), \varphi(w)\} \in E(H)$ where $\varphi(v) \in V(H_{[z_1]}) - \chi_{z_1}$ (by our assumption about D). By the definition of S_1, $\varphi(w) \notin \chi_{z_1}$. Since $\varphi(w)$ appears in χ_z and not χ_{z_1} and each vertex appears in bags of nodes of a connected subtree of a tree decomposition (Property (3) of tree decompositions), $\varphi(w) \notin V(H_{[z_1]})$. Thus $\varphi(v) \in V(H_{[z_1]}) - \chi_{z_1}$ and $\varphi(w) \in V(H) - V(H_{[z_1]})$. But since χ_{z_1} is a separator (Lemma 12), there is no such edge $\{\varphi(v), \varphi(w)\} \in E(H)$. Using this contradiction, we finish the proof of our claim. □

We now construct CPIs ξ_1 and ξ_2. First we define $\mathcal{D}_i = \{D \in \mathcal{D} | \varphi(V(D)) \in V(H_{[z_i]}) - \chi_{z_i}\}$. Using our claim, $\mathcal{D}_i \subseteq \mathcal{C}(G[V - S_i])$ and $\mathcal{D}_1$ and $\mathcal{D}_2$ partition $\mathcal{D}$. We now define mapping $\varphi_i(v) = \varphi(v)$ for $v \in S_i \cup V(\mathcal{D}_i)$ and mapping $\psi_i(v) = \varphi_i(v)$ for $v \in S_i$. We show that φ_i is an extension of iso-triple $\xi_i = (S_i, \mathcal{D}_i, \psi_i)$ and thus ξ_i is a CPI. . By the definition of S_i, $S_i \subseteq S$, and by the definition of D_i, $V(\mathcal{D}_i) \subseteq V(\mathcal{D})$, and thus $S_i \cup V(\mathcal{D}_i) \subseteq S \cup V(\mathcal{D})$. Using this fact, Properties [Pa] and [Pc] of φ_i follow from Properties [Pa] and [Pc] of φ. Property [Pb] of φ_i holds by definitions of S_i and ψ_i and the fact that $\varphi(v) \notin \chi_z$ for $v \in V(\mathcal{D})$ (by Property [Pb] for φ).

We prove that all conditions of join-consistency are satisfied. Let $v \in S_i$. By the definition of ψ_i, $\psi_i(v) = \varphi_i(v)$ for $v \in S_i$. By the definition of φ_i, we have $\varphi_i(v) = \varphi(v)$ and by Property [Pb] of φ, $\varphi(v) = \psi(v)$. Thus $\psi_i(v) = \psi(v)$ for $v \in S_i$ and Condition (3) is satisfied. By the definition of S_i and the fact that $\varphi_i(v) = \varphi(v) = \psi(v)$ for $v \in S$, Condition (1) is satisfied. Condition (4) follows from Property [Pc] of φ and the fact that $\varphi(v) = \psi(v)$ for $v \in S$. Condition (2) follows from the definition of $\mathcal{D}_i$ and our claim.

We now prove if there exist CPIs ξ_i relative to z_i such that an iso-triple ξ is join-consistent with ξ_1 and ξ_2, then ξ is a CPI relative to z. We define a mapping φ for iso-triple ξ from ψ and extensions φ_i's of ψ_i's as follows: for $v \in V(\mathcal{D}_i)$, $\varphi(v) = \varphi_i(v)$ and for $v \in S$, $\varphi(v) = \psi(v)$. We note that each vertex $v \in V(\mathcal{D})$ is either in $V(\mathcal{D}_1)$ or in $V(\mathcal{D}_2)$ (Condition (2)). We now show that φ is an extension of ξ. By Property [Pa] of φ_i, $\varphi(u) \neq \varphi(v)$ for $u, v \in V(\mathcal{D}_i)$. Since ψ is a one-to-one mapping, $\varphi(u) \neq \varphi(v)$ for $u, v \in S$. For $u \in S$ and $v \in V(D_i)$, since $\psi(v) \in \chi_z$ and $\varphi(u) \notin \chi_z$, $\varphi(u) \neq \varphi(v)$.

Thus Property [Pa] of φ holds. Since by the definition of φ, $\varphi(v) = \psi(v) \in \chi_z$ for $v \in S$, and $\varphi(v) = \varphi_i(v) \notin \chi_z$ for $v \in V(\mathcal{D}) = V(\mathcal{D}_1) \cup V(\mathcal{D}_2)$, Property [Pb] of φ holds. We now show Property [Pc] of φ holds. By Condition (4) of join-consistency, for $u, v \in S$, $\{u, v\} \in E(G)$ if and only if $\{\varphi(u), \varphi(v)\} \in E(H)$. By Property [Pc] of φ_1 and φ_2, $\{u, v\} \in E(G)$ if and only if $\{\varphi(u), \varphi(v)\} \in E(H)$ for $u, v \in V(\mathcal{D}) = V(\mathcal{D}_1) \cup V(\mathcal{D}_2)$. In the last case in which $u \in S$ and $v \in V(D)$, where $D \in \mathcal{D}$, by Condition (2) of join-consistency, either $D \in \mathcal{D}_1$ or $D \in \mathcal{D}_2$. Without loss of generality, we assume $D \in \mathcal{D}_1$. Since $\mathcal{D}_1 \subseteq \mathcal{C}(G[V - S_1])$ (by the definition of $\mathcal{D}_1$), D is a component of $G[V - S_1]$, and thus all outgoing edges from $V(D)$ go into S_1. Thus $u \in S_1$ and by Property [Pc] of φ_1, $\{u, v\} \in E(G)$ if and only if $\{\varphi(u), \varphi(v)\} \in E(H)$. Hence Property [Pc] of φ holds. □

Proof. of Lemma 7: By Lemma 5, for each CPI ξ relative to z, there exists a CPI ξ' relative to the child z' of z. Using this fact, we try to find a CPI ξ relative to z which can be constructed from a CPI ξ' relative to z'. We construct an iso-triple ξ separator-consistent with CPI ξ'. Lines 52 and 53 correspond to Conditions (1) and (3) of separator-consistency. Element $\mathcal{D}$ of ξ is constructed in lines 54-60 in which we find all components $D \in G[V - S]$ which are supergraphs of components in $\mathcal{D}'$ and not supergraphs of components in $\mathcal{C}(G[V - S']) - \mathcal{D}'$ (see Condition (2) of separator-consistency). Thus iso-triple ξ is separator-consistent with the CPI ξ' and by Lemma 5, it is a CPI.

We now compute the running time. For each element of the full set array of z', we specify S and ψ for an iso-triple $\xi = (S, \mathcal{D}, \psi)$ relative to z in constant time, because the number of vertices in S and S' is at most constant $k + 1$ (lines 52-53). Constructing $\mathcal{D}$ takes $O(g(k + 1, n))$ time (lines 54-59), since we consider each $C' \in \mathcal{C}(G[V - S'])$ in line 55 and each $C \in \mathcal{C}(G[V - S])$ in line 58 whose number is $O(g(k + 1, n))$. Thus executing lines 52-62 takes at most $O(g(k + 1, n))$ time. Since the number of iso-triples checked in line 51 is at most $O(2^{g(k+1,n)}|V(G)|^{k+1})$, the procedure *BuildFSAofSeparatorNode* takes $O(g(k + 1, n) \cdot 2^{g(k+1,n)}|V(G)|^{k+1})$ time. □

Proof. of Lemma 8: By Lemma 6, for each CPI ξ relative to z, there exist CPIs ξ_1 and ξ_2 relative to its children z_1 and z_2 such that ξ is join-consistent with them. Intuitively, we try to find them and use them as a certificate for ξ. Lines 77, 78 and 79 correspond to Conditions (1), (3) and (4) of join-consistency. By Condition (2) of join-consistency, we know that $\mathcal{D}_1$ and $\mathcal{D}_2$ partitions $\mathcal{D}$. Thus we examine all partitions by brute force (line 80) and for each of them we check whether $\mathcal{D}_i$ is a subset of $\mathcal{C}(G[V - S_i])$, $i = 1, 2$ (lines 81-84). We note that, in line 83, we check whether a component $D' \subseteq \mathcal{C}(G[V - S_i])$. Thus if we find two iso-triples ξ_1 and ξ_2 which are also CPIs (line 86) then ξ is also a CPI (line 87).

To compute the running time, we first observe that we can execute lines 77-79 in constant time since $|S|$ is bounded above by $k+1$. Since $\mathcal{D}$ has at most $O(g(k+1, n))$ elements, we have $O(2^{g(k+1,n)})$ choices for partitioning elements of $\mathcal{D}$ between $\mathcal{D}_1$ and $\mathcal{D}_2$ (line 80). Executing lines 82-83 takes $O(g(k+1, n))$ time, since $\mathcal{D}$ has at most $O(g(k+1, n))$ elements. Therefore, at most $O(g(k+1, n) \cdot 2^{g(k+1,n)})$ time is required for executing lines 77-83. Since the number of iso-triples checked in line 76 is at most $O(2^{g(k+1,n)}|V(G)|^{k+1})$, the running time of this procedure is in $O(g(k + 1, n) \cdot 2^{2g(k+1,n)}|V(G)|^{k+1})$. □

Proof. of Theorem 3: We first prove the correctness of the algorithm. We show that FSA array relative to each node z of $TD(H)$ is in correct form, that is, for each ξ relative to a node z, $FSA[z, \xi] =$ **true** if and only if ξ is a CPI relative to z. Using this fact, we check whether there exists a complete CPI relative to the root r of $TD(H)$ (line 28), and the correctness of the algorithm immediately follows from Lemma 4.

To prove our claim, we use induction on the height of a node z in $TD(H)$. If the height is zero, then z is a leaf. We consider an iso-triple ξ relative to z. Since $H_{[z]} = H[\chi_z]$ for a leaf, if ξ is a CPI then $\mathcal{D} = \emptyset$. In addition by Property [Pb] of extensions and the fact that ψ is a one-to-one mapping, we have $\varphi = \psi$. Using these facts, we only need to check Property [Pc] of extensions. We check all these conditions in line 21 and thus the claim is true for a leaf node. For a separator node z, by the induction hypothesis, the claim is true for the child z' of z and by Lemma 7, the claim is true for z. For a join node z, again by the induction hypothesis, the claim is true for children z_1 and z_2 of z and by Lemma 8 the claim is true for z.

Now, we compute the running time of the algorithm. We choose every set S with at most $k+1$ vertices in line 3, and using depth first search, we find connected components of $G[V - S]$ in $O(|V(G)|)$ time (line 4). Similarly, we can find connected components of $G[V - S']$ in line 6 in $(O(|V(G)|)$ time for each set $S' \subset S$. We can execute line 8 in $O(|V(G)|)$ time for each component $C'_i \in \mathcal{C}(G[V - S'])$, $S' \subset S$. To do that we only need to have two arrays A and A'. Array A (A') is of size $|V(G)|$ and $A[v] = C$ ($A'[v] = C'$) if vertex v is in $V(C)$ ($V(C')$), where $C \in \mathcal{C}(G[V - S])$ ($C' \in \mathcal{C}(G[V - S'])$).

Then we can execute line 8 by only one pass of array A' to find the corresponding component of each vertex $v \in C'_i$ in array A. Using this fact and since the number of C_i's for each set S is at most $O(g(k+1,n))$ (line 7), we can execute lines 7-8 in $O(g(k+1,n) \cdot |V(G)|)$ time. Similarly, we can execute lines 9-10 in $O(g(k+1,n) \cdot |V(G)|)$ time. As the number of such sets S' for each set S is a constant (since k is a constant), this step takes $O(g(k+1,n) \cdot |V(G)|)$ time for each set S (lines 4-10). Since the number of choices of S is in $O(|V(G)|^{k+1})$ (line 3), the overall running time of lines 3-10 is in $O(2^{g(k+1,n)}|V(G)|^{k+1})$ time. In lines 11-14, we construct all iso-triples. As the number of iso-triples is bounded above by $O(2^{g(k+1,n)}|V(G)|^{k+1})$ by Lemma 3 and processing each of them takes constant time (line 14), we can execute lines 11-14 in $O(2^{g(k+1,n)}|V(G)|^{k+1})$ time. Therefore preprocessing steps (lines 1-14) takes the sum of the two aforementioned times which is in $O(g(k+1,n) \cdot |V(G)| \cdot 2^{g(k+1,n)} \cdot |V(G)|^{k+1}) = O(g(k+1,n) \cdot |V(G)|^{k+1} \cdot |V(H)|)$ time.

For a leaf, the check in line 21 takes constant time because the number of vertices in S is at most $k+1$. Since the number of iso-triples checked in line 19 is at most $O(2^{g(k+1,n)}|V(G)|^{k+1})$, the running time of lines 19-22 is at most $O(2^{g(k+1,n)}|V(G)|^{k+1})$. For a separator node z, by Lemma 7, executing the procedure *BuildFSAofSeparatorNode* takes $O(g(k+1,n) \cdot 2^{g(k+1,n)}|V(G)|^{k+1})$ time. Finally, for a join node, the procedure *BuildFSAofJoinNode* takes $O(g(k+1,n) \cdot 2^{2g(k+1,n)}|V(G)|^{k+1})$ time (Lemma 8). Therefore, the running time for a node of $TD(H)$ is at most $O(g(k+1,n) \cdot 2^{2g(k+1,n)}|V(G)|^{k+1})$ due to join nodes.

Since the number of nodes of $TD(H)$ is in $O(|V(H)|)$ and executing lines 16-17 takes $O(2^{g(k+1,n)}|V(G)|^{k+1})$ time whose time dominated by the others, the running time of the main loop of this algorithm (lines 15-26) is in $O(g(k+1,n) \cdot 2^{2g(k+1,n)}|V(G)|^{k+1} \cdot |V(H)|)$. The condition in line 28 also can be computed in constant time and thus executing line 27-28 takes $O(2^{g(k+1,n)}|V(G)|^{k+1})$ time. Therefore the running time of the whole algorithm is in $O(g(k+1,n) \cdot 2^{2g(k+1,n)}|V(G)|^{k+1} \cdot |V(H)|)$. □

Proof. of Lemma 9: First we consider $TD(G)$ of width k. By Property (2) of tree decompositions, for each edge $\{u,v\} \in E(G)$, there exists a node z such that both u and v belong to χ_z. To construct a tree decomposition of width k for G^*, we first construct $TD(G)$ and then for each edge $\{u,v\} \in E(G)$, we append a node z' with $\chi_{z'} = \{u,v,w\}$ to the node z, where w is the added vertex in subdivision of $\{u,v\}$. □

Proof. of Lemma 10: Suppose a set S removed from G^* has l_1 vertices originally from G and l_2 vertices added by subdivisions. Without loss of generality, we can assume that first vertices originally from G are removed and then the remaining vertices of S are removed one at a time. Suppose we remove vertices originally from G in both G and G^*. We call a component in G^* a *new component*, if it has no corresponding component in G. We bound the number of new components. For each pair u and v of vertices originally from G, there can exist edge $\{u,v\} \in E(G)$. Subdivision of $\{u,v\}$ in G^* adds a new component which only contains w. After removing vertices originally from G, we can only have this kind of new component. Hence removing these vertices can produce at most $\binom{l_1}{2} \leq k^2 - k$ new components. We now remove the rest of vertices of S one at a time. Each of these vertices is added on one edge of G, and hence its removal in G^* can increase the number of components by one. Thus removal of these vertices can increase the number of components by at most $l_2 \leq k$. Therefore $|\mathcal{C}(G^*[V-S])|$ is at most $g(k,n) + k^2$. Since k is a constant, this number is in $O(g(k,n))$. □

Proof. of Theorem 5: Let H be a planar cubic graph (every vertex has either degree three or degree zero) in which there is no face with fewer than five edges. Finding a Hamiltonian circuit in H is NP-complete [15]. As mentioned above, planar graphs have locally bounded treewidth. By choosing G to be a cycle of length n, where n is the size of H, the subgraph isomorphism problem is equivalent to finding a Hamiltonian cycle in H and thus it is NP-complete. □