Information and Computation 217 (2012) 71–83

Contents lists available at SciVerse ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

Restricted space algorithms for isomorphism on bounded treewidth graphs ☆

Bireswar Das [a], Jacobo Torán [b,*], Fabian Wagner [b,1]

[a] Indian Institute of Technology, Gandhinagar, India
[b] Institut für Theoretische Informatik, Universität Ulm, Germany

ARTICLE INFO

Article history:
Received 3 December 2010
Revised 3 May 2012
Available online 29 May 2012

Keywords:
Complexity
Algorithms
Graph Isomorphism problem
Treewidth
LogCFL

ABSTRACT

The Graph Isomorphism problem restricted to graphs of bounded treewidth or bounded tree distance width are known to be solvable in polynomial time. We give restricted space algorithms for these problems proving the following results:

- Isomorphism for bounded tree distance width graphs is in L and thus complete for the class. We also show that for this kind of graphs a canon can be computed within logspace.
- For bounded treewidth graphs, when both input graphs are given together with a tree decomposition, the problem of whether there is an isomorphism which respects the decompositions (i.e. when only isomorphisms are considered, mapping bags in one decomposition blockwise onto bags in the other decomposition) is in L.
- For bounded treewidth graphs, when one of the input graphs is given with a tree decomposition the isomorphism problem is in LogCFL.
- As a corollary the isomorphism problem for bounded treewidth graphs is in LogCFL. This improves the known TC^1 upper bound for the problem given by Grohe and Verbitsky.

1. Introduction

The Graph Isomorphism problem consists in deciding whether two given graphs are isomorphic, or in other words, whether there exists a bijection between the vertices of both graphs preserving the edge relation. Graph Isomorphism is a well studied problem in NP because of its many applications and also because it is one of the few natural problems in this class not known to be solvable in polynomial time nor known to be NP-complete (see [12]). Although for the case of general graphs no efficient algorithm for the problem is known, the situation is much better when certain parameters in the input graphs are bounded by a constant. For example the isomorphism problem for graphs of bounded degree [14], bounded genus [16], bounded color classes [15], or bounded treewidth [2] is known to be in P. Recently some of these upper bounds have been improved with the development of space efficient techniques, most notably Reingold's deterministic logarithmic

☆ A preliminary version of this paper has appeared at the conference STACS 2010.

* Corresponding author.
E-mail addresses: bireswar@iitgn.ac.in (B. Das), jacobo.toran@uni-ulm.de (J. Torán), fabian.wagner@uni-ulm.de (F. Wagner).

[1] Supported by DFG grant TO 200/2-2.

0890-5401/$ – see front matter © 2012 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.ic.2012.05.003

space algorithm for connectivity in undirected graphs [17]. In some cases logarithmic space algorithms have been obtained. For example graph isomorphism for trees [13], planar graphs [5] or k-trees [11] is known to be in the class L. In other cases the problem has been classified in some other small complexity classes below P. The isomorphism problem for graphs of bounded treewidth is known to be in TC^1 [9] and the problem restricted to graph with bounded color classes is known to be in the #L hierarchy [1].

In this paper we address the question of whether the isomorphism problem restricted to graphs of bounded treewidth and bounded tree distance width can be solved in logarithmic space. Intuitively speaking, the treewidth of a graph measures how much it differs from a tree. This concept has been used very successfully in algorithmics and fixed-parameter tractability (see e.g. [3,4]). For many complex problems, efficient algorithms have been found for the cases when the input structures have bounded treewidth. As mentioned above Bodlaender showed in [2] that graph isomorphism can be solved in polynomial time when restricted to graphs of bounded treewidth. More recently Grohe and Verbitsky [9] improved this upper bound showing that the isomorphism problem for this kind of graphs can be solved by a uniform family of threshold circuits of logarithmic depth and polynomial size and lies therefore in the class TC^1.

In this paper we improve this result, showing that the isomorphism problem for bounded treewidth graphs lies in LogCFL, the class of problems logarithmic space reducible to a context free language. LogCFL can be alternatively characterized as the class of problems computable by a uniform family of polynomial size and logarithmic depth circuits with bounded AND and unbounded OR gates, and is therefore a subclass of TC^1. LogCFL is also the best known upper bound for computing a tree decomposition of a graph of bounded treewidth [19,8], which is one bottleneck in our isomorphism algorithm. We prove that if tree decompositions of both graphs are given as part of the input, the question of whether there is an isomorphism respecting the vertex partition defined by the decompositions can be solved in logarithmic space. Our proof techniques are based on methods from recent isomorphism results [5,6] and are very different from those in [9].

The notion of tree distance width, a stronger version of the treewidth concept, was introduced in [20]. There it is shown that for graphs with bounded tree distance width the isomorphism problem is fixed-parameter tractable, something that is not known to hold for the more general class of bounded treewidth graphs. We prove that for graphs of bounded tree distance width it is possible to obtain a tree distance decomposition within logspace. Using this result we show that graph isomorphism for bounded tree distance width graphs can also be solved in logarithmic space. Since it is known that the question is also hard for the class L under AC^0 reductions [10], this exactly characterizes the complexity of the problem. We show that in fact a canon for graphs of bounded tree distance width, i.e. a fixed representative of the isomorphism equivalence class, can be computed in logspace.

2. Preliminaries

We introduce the complexity classes used in this paper. L is the class of decision problems computable by deterministic logarithmic space Turing machines. LogCFL consists of all decision problems that can be Turing reduced in logarithmic space to a context free language. There are several alternative more intuitive characterizations of LogCFL. Problems in this class can be computed by uniform families of polynomial size and logarithmic depth circuits over bounded fan-in AND gates and unbounded fan-in OR gates. We will also use the characterization of LogCFL as the class of decisional problems computable by non-deterministic auxiliary pushdown machines (NAuxPDA). These are Turing machines with a logarithmic space worktape, an additional pushdown and a polynomial time bound [18]. The class TC^1 contains the problems computable by uniform families of polynomial size and logarithmic depth threshold circuits. The known relationships among these classes are

$$L \subseteq \text{LogCFL} \subseteq TC^1.$$

In this paper we consider undirected simple graphs with no self loops. For a graph $G = (V, E)$ and two vertices $u, v \in V$, $d_G(u, v)$ denotes the distance between u and v in G (number of edges in the shortest path between u and v in G). For a set $S \subseteq V$, and a vertex $u \in V$, $d_G(S, u)$ denotes $\min_{v \in S} d_G(v, u)$. $\Gamma(S)$ denotes the set of neighbors of S in G.

In a connected graph G, a separating set is a set of vertices such that deleting the vertices in S (and the edges connected to them) produces more than one connected component.

For $G = (V, E)$ and two disjoint subsets U, W of v we use the following notion for an *induced bipartite subgraph* $B_G[U, W]$ of G on vertex set $U \cup W$ with edge set $\{\{u, w\} \in E \mid u \in U,\ w \in W\}$. Let $G[U]$ be the *induced subgraph* of G on vertex set U.

Definition 2.1. A *tree decomposition* of a graph $G = (V, E)$ is a pair $(\{X_i \mid i \in I\}, T = (I, F))$, where $\{X_i \mid i \in I\}$ is a collection of subsets of v called bags, and T is a tree with node set I and edge set F, satisfying the following properties:

i) $\bigcup_{i \in I} X_i = V$,
ii) for each $\{u, v\} \in E$, there is an $i \in I$ with $u, v \in X_i$, and
iii) for each $v \in V$, the set of nodes $\{i \mid v \in X_i\}$ forms a subtree of T.

The *width* of a tree decomposition of G, $(\{X_i \mid i \in I\}, T = (I, F))$ is defined as $\max\{|X_i| \mid i \in I\} - 1$. The *treewidth* of a graph G is the minimum width over all possible tree decompositions of G.

Definition 2.2. A *tree distance decomposition* of a graph $G = (V, E)$ is a triple $(\{X_i \mid i \in I\}, T = (I, F), r)$, where $\{X_i \mid i \in I\}$ is a collection of subsets of v called *bags*, $X_r = S$ a set of vertices and T is a tree with node set I, edge set F and root r, satisfying the following properties:

i) $\bigcup_{i \in I} X_i = V$ and for all $i \neq j$, $X_i \cap X_j = \emptyset$,
ii) for each $v \in V$, if $v \in X_i$ then $d_G(X_r, v) = d_T(r, i)$, and
iii) for each $\{u, v\} \in E(G)$, there are $i, j \in I$ with $u \in X_i$, $v \in X_j$ and $i = j$ or $\{i, j\} \in F$ (for every edge in G its two endpoints belong to the same or to adjacent bags in T).

Let $D = (\{X_i \mid i \in I\}, T = (I, F), r)$ be a tree distance decomposition of G. X_r is the *root bag* of D. The *width* of D is the maximum number of elements of a bag X_i. The *tree distance width* of a graph G is the minimum width over all possible tree distance decompositions of G.

The tree distance decomposition D is called *minimal* if for each $i \in I$, the set of vertices in the bags with labels in the subtree rooted at i in T induce a connected subgraph in G. In [20] it is shown that for every root set $S \subseteq V$ there is a unique minimal tree distance decomposition of G with root set S. The width of such a decomposition is minimal among the tree distance decompositions of G with root set S.

An isomorphism from G onto H *respects* their tree decompositions D, D' if vertices in a bag of D in G are mapped blockwise onto vertices in a bag of D' in H. Not every isomorphism has this property. In the case of tree distance decompositions the situation is different. Suppose that D is a minimal tree distance decomposition of G with root bag X_r and let φ be an isomorphism between G and H mapping the vertex set X_r in G blockwise to a set $X'_{r'}$ in H. Since there is only one *minimal* tree distance decomposition D' of H with root bag $X'_{r'}$, φ respects the minimal tree distance decompositions of G and H with respect to the root bags X_r and $X'_{r'}$.

$Sym(V)$ is the *symmetric group* on a set V. For two permutations σ, ϕ, the notation $\sigma\phi(X)$ means that the permutations are applied to X from right to left, this is $\sigma(\phi(X))$.

3. Graphs of bounded tree distance width

3.1. Tree distance decomposition in L

We describe an algorithm that on input a graph G and a subset $S \subset V$ produces the minimal tree distance decomposition $D = (\{X_i \mid i \in I\}, T = (I, F), r)$ of G with root set $X_r = S$. The algorithm works within space $c \cdot k \log n$ for some constant c, where k is the width of the minimal tree distance decomposition of G with root set S. The output of the algorithm is a sequence of strings of the form (bag label, bag depth, $v_{i_1}, v_{i_2}, \ldots, v_{i_l}$), indicating the number of the bag, the distance of its elements to S and the list of the elements in the bag.

The algorithm basically performs a depth-first traversal of the tree T in the decomposition while constructing it. Starting at S the algorithm uses three functions for traversing T. These functions perform queries to a logspace subroutine computing reachability [17].

Parent(X_i): On input the elements of a bag X_i the function returns the elements of the parent bag in T. These are the vertices $v \in V$ with the following two properties: $v \in \Gamma(X_i) \setminus X_i$ and v is reachable from S in $G \setminus X_i$. For a vertex v these two properties can be tested in space $O(\log n)$ by an algorithm with input G, S and X_i. In order to find all the vertices in the parent set, the algorithm searches through all the vertices in v.

First Child(X_i): This function returns the elements of the first child of i in T. This is the child with the vertex $v_j \in V$ with the smallest index j. v_j satisfies that $v_j \in \Gamma(X_i) \setminus X_i$ and that v_j is not reachable from S in $G \setminus X_i$. It can be found cycling through the vertices of G (in order as they are given on the input tape) until the first one satisfying the properties is found. The other elements $w \in X_i$ must satisfy the same two properties as v_j and additionally, they must be in the same connected component in $G \setminus X_i$ where v_j is contained. In case X_i does not have any children, the function outputs some special symbol.

Next Sibling(X_i): This function first computes $X_p := \mathsf{Parent}(X_i)$ and then searches for the child of P in T next to X_i. Let v_i be the vertex with the smallest label in X_i. This is done similarly as the computation of First Child. The next sibling is the bag containing the unique vertex v_j with the following properties: v_j is the vertex with the smallest label in this bag, $label(v_j) > label(v_i)$ and there is no other bag which has a vertex with a label between v_i and v_j. The vertex v_j is not reachable from S in $G \setminus X_p$. The other elements in the bag are the vertices satisfying these properties and which are in the same connected component of $G \setminus X_p$ where v_j is contained.

With these three functions the algorithm performs a depth-first traversal of T. It only needs to remember the initial bag $X_0 = S$ which is part of the input, and the elements of the current bag. On a bag X_i it searches for its first child. If it does not exist then it searches for the next sibling. When there are no further siblings, the next move goes up in the

tree T. The algorithm finishes when it returns to S. It also keeps two counters in order to be able to output the number and depth of the bags. The three mentioned functions only need to keep at most two bags (X_i and the root S) in memory and work in logarithmic space. On input a graph G with n vertices and a root set S, the space used by the algorithm is therefore bounded by $c \cdot k \log n$, for a constant c, and k being the minimum width of a tree distance decomposition of G with root set S. When considering how the three functions are defined it is clear that the algorithm constructs a tree distance decomposition with root set S. Also they make sure that for each i the subgraph corresponding to the subtree of T rooted at i (i.e. the subgraph induced by the vertices of the bags in this subtree) is connected, thus producing a minimal decomposition. As observed in [20], this is the unique minimal tree distance decomposition of G with root set S.

3.2. Isomorphism algorithm for bounded tree distance width graphs

For our isomorphism algorithm we use a structure called the *augmented tree* which is based on the underlying tree of a minimal tree distance decomposition. This augmented tree, apart from the bags, contains information about the separating sets which separate bags.

Definition 3.1. Let G be a graph with a minimal tree distance decomposition $D = (\{X_i \mid i \in I\}, T = (I, F), r)$. The *augmented tree* $\mathcal{T}_{(G,D)} = (I_{(G,D)}, F_{(G,D)}, r)$ corresponding to G and D is a tree defined as follows:

- The set of nodes of $\mathcal{T}_{(G,D)}$ is $I_{(G,D)}$ which contains two kinds of nodes, namely $I_{(G,D)} = I \cup J$. Those in I form the set of *bag nodes* in D, and those in j the *separating set nodes*. For each bag node $a \in I$ and each child b of a in T we consider the set $X_a \cap \Gamma(X_b)$, i.e. the *minimum separating set* in X_a which separates X_b from the root bag X_r in G. Let $M_{s_1^a}, \ldots, M_{s_{l(a)}^a}$ be the set of all minimum separating sets in X_a, free of duplicates. There are nodes for these sets $s_1^a, \ldots, s_{l(a)}^a$, the separating set nodes. We define $J = \bigcup_{a \in I} \{s_1^a, \ldots, s_{l(a)}^a\}$. The node $r \in I$ is the root in $\mathcal{T}_{(G,D)}$.
- The set of edges $F_{(G,D)}$ contains edges between bag nodes $a \in I$ and the separating set nodes $s_1^a, \ldots, s_{l(a)}^a \in J$ (edges between bag nodes and their children in the augmented tree). It also contains edges between nodes $b \in I$ and s_j^a if a is the parent node of b in I and $M_{s_j^a}$ is the minimum separating set in X_a which separates X_b from X_r (edges between bag nodes and their parents).

To simplify notation, we will say for example that $s_1, \ldots, s_l$ are the children of a bag node a if the context is clear. To each separating set node s_i, we will address the set of vertices by X_{s_i}. The odd levels of the augmented tree correspond to bag nodes and the even levels correspond to separating set nodes.

Observe that for each node in the augmented tree, we associate a bag to a bag node and a minimum separating set to a separating set node. Hence, every vertex v in the original graph occurs in at least one associated component and it might occur in more than one, e.g. if v is contained in a bag and in a minimum separating set.

Let $T_{(G,D)}$ be an augmented tree of some minimal tree distance decomposition D of a graph G. Let a be a node of $T_{(G,D)}$. The subtree of $T_{(G,D)}$ rooted at a is denoted by T_a. Note that $T_{(G,D)} = T_r$ where X_r is the bag corresponding to the root of the tree distance decomposition D. We define $\mathsf{size}(T_a)$ as sum of the sizes of the components (bags and separating sets) associated to the nodes of T_a. We also define $\mathsf{graph}(T_a)$ as the induced graph in G of the vertices associated to nodes in T_a.

When given a tree distance decomposition, the augmented tree can be computed in logspace. Using the result in Section 3.1 we immediately get:

Lemma 3.2. *There is a function f and an algorithm that on input of a graph G with n vertices and of tree distance width k, computes an augmented tree for G in space $O(f(k) \log n)$.*

Isomorphism order of augmented trees. We describe an isomorphism order procedure for comparing two augmented trees $S_{(G,D)}$ and $T_{(H,D')}$ corresponding to the graphs G and H and their minimal tree distance decompositions D and D', respectively. This isomorphism order is an extension of the one for trees given by Lindell [13]. The trees $S_{(G,D)}$ and $T_{(H,D')}$ are rooted at bag nodes r and r'. The rooted trees are denoted then S_r and $T_{r'}$ as shown in Fig. 1. We will show that two graphs of bounded tree distance width are isomorphic if and only if for some root nodes r and r' the augmented trees corresponding to the minimal tree distance decompositions have the same isomorphism order.

The isomorphism order is defined recursively based on the two order procedures $\prec_{\mathrm{T}}$ and $\prec_{\mathrm{T}}$. The first one $\prec_{\mathrm{T}}$ will be used for comparing augmented subtrees rooted at bag nodes, while $\prec_{\mathrm{T}}$ compares augmented subtrees rooted at separating set nodes.

We introduce some notation needed for the definition of the isomorphism order. For sets of structures $\{A_1, \ldots, A_k\}$ and $\{B_1, \ldots, B_k\}$ and a total order $<$ between such structures the notation $(A_1, \ldots, A_k) < (B_1, \ldots, B_k)$ represents that the structures are ordered within the tuples according to $\leqslant$ and that for some $i \in \{1, \ldots, k\}$: $A_i < B_i$ and for all $j \in \{1, \ldots, i-1\}$: $A_j = B_j$.

The isomorphism order procedure compares first the sets of vertices X_r and $X'_{r'}$ where r and r' are the root nodes in the decompositions D and D'. For this we consider pairs of permutations $(\sigma, \sigma') \in Sym(X_r) \times Sym(X'_{r'})$. The notation

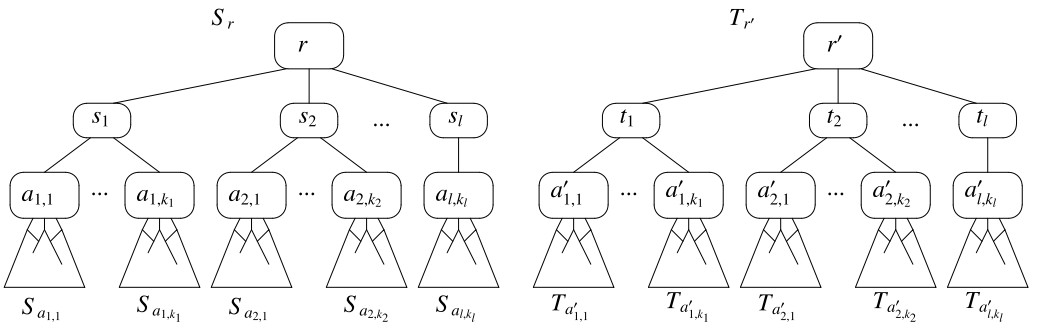

Fig. 1. The augmented trees S_r and $T_{r'}$ rooted at bag nodes r and r'. Node r has separating set nodes $s_1, \ldots, s_l$ as children. The children of s_1 are again bag nodes $a_{1,1}, \ldots, a_{1,k_1}$. $S_{a_{i,j}}$ is the subtree rooted at $a_{i,j}$. Bag nodes and separating set nodes alternate in the tree.

$\sigma(G[X_r])$ and $\sigma'(H[X'_{r'}])$ describes a fixed labeling of the vertices of the corresponding induced subgraphs, given by the permutations. We say $\sigma(G[X_r]) < \sigma'(H[X'_{r'}])$ if $|X_r| < |X'_{r'}|$ or if the adjacency matrix of the induced subgraph $G[X_r]$ with its vertices ordered according to σ is lexicographically smaller than that of the induced subgraph $H[X'_{r'}]$ ordered according to σ'.

Furthermore, we need a function $\mathrm{pos}_r : X_r \mapsto \{1, \ldots, |X_r|\}$ which gives labels to vertices according to their order in V. For example if $X_r = \{v_1, v_5, v_7\}$ then $\mathrm{pos}_r(v_1) = 1$, $\mathrm{pos}_r(v_5) = 2$ and $\mathrm{pos}_r(v_7) = 3$. Accordingly, we define $\mathrm{pos}'_{r'} : X'_{r'} \mapsto \{1, \ldots, |X'_{r'}|\}$.

Recall that for a graph $G = (V, E)$ and two disjoint vertex sets $U, W \subseteq V$, $B_G[U, W]$ denotes the bipartite graph with vertices $U \cup W$ and edge set $\{\{u, w\} \in E(G) \mid u \in U,\ w \in W\}$. For $\sigma \in Sym(U)$ and $\phi \in Sym(W)$, $\sigma\phi(B_G[U, W])$ describes the adjacency matrix of $B_G[U, W]$ with the vertices in U ordered according to σ and those in W ordered according to ϕ.

For two permutations $(\sigma, \sigma') \in Sym(X_r) \times Sym(X_{r'})$ we define now the order $S_r^\sigma <_{\mathrm{T}} T_{r'}^{\sigma'}$.

Definition 3.3. For two augmented trees rooted at bag nodes r and r' we will say $S_r <_{\mathrm{T}} T_{r'}$ if there exists a permutation $\sigma \in Sym(X_r)$ such that for all $\sigma' \in Sym(X_{r'})$: $S_r^\sigma <_{\mathrm{T}} T_{r'}^{\sigma'}$.

We say, $S_r^\sigma <_{\mathrm{T}} T_{r'}^{\sigma'}$ is true if one of the following holds:

1) $\sigma(G[X_r]) < \sigma'(H[X'_{r'}])$, or
2) $\sigma(G[X_r]) = \sigma'(H[X'_{r'}])$ but $\mathrm{size}(S_r) < \mathrm{size}(T_{r'})$, or
3) $\sigma(G[X_r]) = \sigma'(H[X'_{r'}])$ and $\mathrm{size}(S_r) = \mathrm{size}(T_{r'})$ but $\#r < \#r'$ where $\#r$ and $\#r'$ is the number of children of r and r', respectively, or
4) $\sigma(G[X_r]) = \sigma'(H[X'_{r'}])$ and $\mathrm{size}(S_r) = \mathrm{size}(T_{r'})$ and $\#r = \#r' = l$ but $(S^\sigma_{s_1}, \ldots, S^\sigma_{s_l}) \prec_{\mathrm{T}} (T^{\sigma'}_{t_1}, \ldots, T^{\sigma'}_{t_l})$. The order $\prec_{\mathrm{T}}$ for subtrees rooted at separating set nodes is defined as in the following way, $S^\sigma_{s_i} \prec_{\mathrm{T}} T^{\sigma'}_{t_j}$ if:
   i) $\sigma(X_{s_i}) < \sigma'(X'_{t_j})$, i.e. for $X_{s_i} = \{v_{i_1}, \ldots, v_{i_h}\}$, $X'_{t_j} = \{v'_{j_1}, \ldots, v'_{j_{h'}}\}$: $(\mathrm{pos}_r\, \sigma(v_{i_1}), \ldots, \mathrm{pos}_r\, \sigma(v_{i_h})) < (\mathrm{pos}'_{r'}\, \sigma'(v'_{j_1}), \ldots, \mathrm{pos}'_{r'}\, \sigma'(v'_{j_{h'}}))$, or
   ii) $\sigma(X_{s_i}) = \sigma'(X'_{t_j})$ but $k_i < k'_j$, where k_i and k'_j are the number of children of s_i and t_j, respectively, or
   iii) $\sigma(X_{s_i}) = \sigma'(X'_{t_j})$, $k_i = k'_j = m$ but $(B_G[X_{s_i}, X_{a_{i,1}}]^\sigma, \ldots, B_G[X_{s_i}, X_{a_{i,m}}]^\sigma) < (B_H[X'_{t_j}, X'_{a'_{j,1}}]^{\sigma'}, \ldots, B_H[X'_{t_j}, X'_{a'_{j,m}}]^{\sigma'})$ where $B_G[X_{s_i}, X_{a_{i,i'}}]^\sigma < B_H[X'_{t_j}, X'_{a'_{j,j'}}]^{\sigma'}$ if there exists $\phi \in Sym(X_{a_{i,i'}})$ such that for all $\phi' \in Sym(X'_{a'_{j,j'}})$, $\sigma\phi(B_G[X_{s_i}, X_{a_{i,i'}}]) < \sigma'\phi'(B_H[X'_{t_j}, X'_{a'_{j,j'}}])$ via lexicographical comparison of the adjacency matrices of both induced bipartite subgraphs where all vertices are ordered according to σ, ϕ and σ', ϕ' respectively, or
   iv) $\sigma(X_{s_i}) = \sigma'(X'_{t_j})$, $k_i = k'_j = m$, $(B_G[X_{s_i}, X_{a_{i,1}}]^\sigma, \ldots, B_G[X_{s_i}, X_{a_{i,m}}]^\sigma) = (B_H[X'_{t_j}, X'_{a'_{j,1}}]^{\sigma'}, \ldots, B_H[X'_{t_j}, X'_{a'_{j,m}}]^{\sigma'})$, but there exists $q \in \{1, \ldots, m\}$ such that for every $p \in \{1, \ldots, q-1\}$:

$$\Big[\forall \phi_p \in Sym(X_{a_{i,p}})\ \exists \phi'_p \in Sym\big(X'_{a'_{j,p}}\big)\colon \sigma\phi_p\big(B_G[X_{s_i}, X_{a_{i,p}}]\big) = \sigma'\phi'_p\big(B_H\big[X'_{t_j}, X'_{a'_{j,p}}\big]\big) \text{ and } S^{\phi_p}_{a_{i,p}} =_{\mathrm{T}} T^{\phi'_p}_{a'_{j,p}}\Big] \quad \text{and}$$

$$\Big[\exists \phi_q \in Sym(X_{a_{i,q}})\ \forall \phi'_q \in Sym\big(X'_{a'_{j,q}}\big) \text{ if } \sigma\phi_q\big(B_G[X_{s_i}, X_{a_{i,q}}]\big) = \sigma'\phi'_q\big(B_H\big[X'_{t_j}, X'_{a'_{j,q}}\big]\big) \text{ then } S^{\phi_q}_{a_{i,q}} <_{\mathrm{T}} T^{\phi'_q}_{a'_{j,q}}\Big].$$

We say that two augmented trees S_r and $T_{r'}$ are *equal according to the isomorphism order*, denoted $S_r =_{\mathrm{T}} T_{r'}$, if neither $S_r <_{\mathrm{T}} T_{r'}$ nor $T_{r'} <_{\mathrm{T}} S_r$ holds.

Correctness of the isomorphism order. It is not hard to see that the isomorphism order defines a total order on augmented trees. We show now that it is a good tool for testing graph isomorphism since two graphs are isomorphic if and only if for

some choice of the root bags, the augmented trees associated with the corresponding minimal tree distance decompositions have the same order under $=_{\mathrm{T}}$.

Theorem 3.4. *Let $G = (V_1, E_1)$ and $H = (V_2, E_2)$ be two graphs and $X_r \subseteq V_1$ and $X'_{r'} \subseteq V_2$ root bags producing minimal tree distance decompositions of the graphs G and H with augmented trees S_r and $T_{r'}$ respectively.*

There is an isomorphism between G and H mapping setwise X_r to $X'_{r'}$ if and only if for some permutations $\sigma, \sigma' \in Sym(X_r) \times Sym(X'_{r'})$, $S_r^{\sigma} =_{\mathrm{T}} T_{r'}^{\sigma'}$.

Proof. From left to right, let G and H be isomorphic graphs with an isomorphism Π mapping X_r to $X'_{r'}$. Let us denote by π the restriction from Π to the domain X_r and let $\sigma \in Sym(X_r)$ be a permutation minimizing $\sigma(G[X_r])$. Define $\sigma' = \pi\sigma\pi^{-1}$. σ' is a permutation in $Sym(X'_{r'})$ and $\sigma(G[X_r]) = \sigma'(H[X'_{r'}])$. Since G and H are isomorphic with an isomorphism mapping X_r to $X'_{r'}$ and since the minimal tree distance decomposition is unique, the augmented trees of G and H with respect to the root bags X_r and $X'_{r'}$ are also isomorphic and we have $\sigma(G[X_r]) = \sigma'(H[X'_{r'}])$, $\mathsf{size}(S_r) = \mathsf{size}(T_{r'})$ and $\#r = \#r' = l$ for some l. The isomorphism also implies $(S^{\sigma}_{s_1}, \ldots, S^{\sigma}_{s_l}) =_{\mathrm{T}} (T^{\sigma'}_{t_1}, \ldots, T^{\sigma'}_{t_l})$ (where the equality refers here to the order $\prec_{\mathrm{T}}$ defined between subtrees rooted at separating set nodes) and for all $i \in \{1, \ldots, l\}$, $\sigma(X_{s_i}) = \sigma'(X'_{t_i})$, the number of children of S_{s_i}, k_i, coincide with that of T_{t_i}, and for all $j \in \{1, \ldots, k_i\}$, $B_G[X_{s_i}, X_{a_{i,j}}]^{\sigma} = B_H[X'_{t_i}, X'_{a'_{i,j}}]^{\sigma'}$ and the subtree $S_{a_{i,j}}$ is isomorphic to $T_{a'_{i,j}}$ via an isomorphism $\varphi_{i,j}$ mapping $X_{a_{i,j}}$ to $X'_{a'_{i,j}}$. For any permutation $\phi_{i,j} \in Sym(X_{a_{i,j}})$ consider $\phi'_{i,j} = \varphi_{i,j}\phi_{i,j}\varphi^{-1}_{i,j}$. $\phi'_{i,j} \in Sym(X'_{a'_{i,j}})$ and $\phi'_{i,j}(H[X'_{a'_{i,j}}]) = \phi_{i,j}(G[X_{a_{i,j}}])$ which implies $S^{\phi_{i,j}}_{a_{i,j}} =_{\mathrm{T}} T^{\phi'_{i,j}}_{a'_{i,j}}$. Since this is true for every i and j, by the definition of the isomorphism order we have $S_r^{\sigma} =_{\mathrm{T}} T_{r'}^{\sigma'}$.

The direction from right to left is proven by induction on the number of levels with bag nodes in the augmented tree. The base case is when there is only one bag node in each of the augmented trees, i.e. all vertices in G and H are associated to the single bags X_r and $X'_{r'}$ respectively.

By hypothesis there exists a pair of permutations $(\sigma, \sigma') \in Sym(X_r) \times Sym(X'_{r'})$ with $S_r^{\sigma} = T_{r'}^{\sigma'}$. This means $\sigma(G[X_r]) = \sigma'(H[X'_{r'}])$ and since $G = G[X_r]$ and $H = H[X'_{r'}]$, both graphs are isomorphic, with isomorphism $\sigma'^{-1}\sigma$.

For the induction step, since $S_r^{\sigma} =_{\mathrm{T}} T_{r'}^{\sigma'}$, it holds $\sigma(G[X_r]) = \sigma'(H[X'_{r'}])$, $\mathsf{size}(S_r) = \mathsf{size}(T_{r'})$ and $\#r = \#r' = l$ for some l. Moreover $(S^{\sigma}_{s_1}, \ldots, S^{\sigma}_{s_l}) =_{\mathrm{T}} (T^{\sigma'}_{t_1}, \ldots, T^{\sigma'}_{t_l})$. This means that for all $i \in \{1, \ldots, l\}$, $\sigma(X_{s_i}) = \sigma'(X'_{t_i})$, the number of children of S_{s_i}, k_i, coincide with that of T_{t_i}, and for all $j \in \{1, \ldots, k_i\}$, $B_G[X_{s_i}, X_{a_{i,j}}]^{\sigma} = B_H[X'_{t_i}, X'_{a'_{i,j}}]^{\sigma'}$. Let ϕ_j be any permutation in $Sym(X_{a_{i,j}})$ and let $\phi'_j \in Sym(X'_{a_{i,j}})$ satisfying $\sigma\phi_j(B_G[X_{s_i}, X_{a_{i,j}}]) = \sigma'\phi_j(B_H[X'_{t_i}, X'_{a'_{i,j}}])$. Such a permutation ϕ'_j always exists since $B_G[X_{s_i}, X_{a_{i,j}}]^{\sigma} = B_H[X'_{t_i}, X'_{a'_{i,j}}]^{\sigma'}$. For all $j \in \{1, \ldots, k_i\}$, we have $S^{\phi_j}_{a_{i,j}} =_{\mathrm{T}} T^{\phi'_j}_{a'_{i,j}}$ and by induction hypothesis for all $j \in \{1, \ldots, k_i\}$, $\mathsf{graph}(S_{a_{i,j}})$ is isomorphic to $\mathsf{graph}(T_{a'_{i,j}})$ with an isomorphism that maps $X_{a_{i,j}}$ to $X'_{a'_{i,j}}$. Observe that since the nodes $a_{i,j}$ are bag nodes, for $j \neq j'$ the graph vertices associated to $S_{a_{i,j}}$ are disjoint from those associated to $S_{a_{i,j'}}$. All these isomorphisms between subgraphs of G and H are consistent each other and also with $\sigma'^{-1}\sigma$ and can therefore be extended to an isomorphism between G and H mapping X_r to $X'_{r'}$ via $\sigma\sigma'^{-1}$. □

Corollary 3.5. *Two graphs G and H are isomorphic if and only if there is a pair of root sets producing minimal tree distance decompositions of the graphs with augmented trees S_r and $T_{r'}$ with $S_r =_{\mathrm{T}} T_{r'}$.*

We describe now an algorithm for computing the isomorphism order. After this, we analyze the complexity of the algorithm showing that if the tree distance width is constant then the isomorphism order of the corresponding augmented trees can be computed in logarithmic space.

Isomorphism of two subtrees rooted at bag nodes r and r'. We are interested in finding the mappings σ and σ' which lead to the minimum isomorphism order of the trees S_r and $T_{r'}$. For this we define a set of permutation pairs $\Theta_{(r,r')} \subseteq Sym(X_r) \times Sym(X'_{r'})$ related to the pair of nodes (r, r'). The order procedure cycles through all permutation pairs contained in $\Theta_{(r,r')}$. Initially $\Theta_{(r,r')} = Sym(X_r) \times Sym(X'_{r'})$. We will see, if r and r' are not the root of the overall tree, the set $\Theta_{(r,r')}$ may be restricted to a subset. The algorithm sets up a table for $\Theta_{r,r'}$ which contains at most $(k!)^2$ entries. The algorithm records for each entry $(\sigma, \sigma') \in \Theta_{r,r'}$ the result of the comparison of S_r^{σ} with $T_{r'}^{\sigma'}$.

In Step 1, we have constant size components associated to the bag nodes. We simply compare the adjacency matrices of $G[X_r]$ and $H[X'_{r'}]$ bitwise, where the elements are arranged in rows and columns in increasing order according to the permutations σ and σ'.

Steps 2 and 3 can be done in logspace by comparing the tree size and the number of children of r and r'. In Step 4 the subtrees rooted at separating set nodes are compared. This requires similar arguments as in [13]. We run through the children of r and r' in a fixed order using the functions FirstChild and NextSibling. First, we find the minimum subtrees S_{s_i} and T_{t_j} according to $\prec_{\mathrm{T}}$. If they are $\prec_{\mathrm{T}}$-equal then we compute the number of $\prec_{\mathrm{T}}$-equal siblings for s_i and t_j, by running

through all children of S_r and $T_{r'}$. If the numbers are equal, then we proceed with the minimum subtrees larger than S_{s_i} and T_{t_j} according to $\prec_{\mathrm{T}}$. If they are not equal, then we know that $S_r^\sigma \prec_{\mathrm{T}} T_{r'}^{\sigma'}$ (or $T_{r'}^{\sigma'} \prec_{\mathrm{T}} S_r^\sigma$). If all the tests are equal then we know that $S_r^\sigma =_{\mathrm{T}} T_{r'}^{\sigma'}$ and proceed with the next entry in $\Theta_{r,r'}$. In $\Theta_{r,r'}$, the permutation σ is the smallest if $S_r^\sigma \leqslant_{\mathrm{T}} T_{r'}^{\sigma'}$ for all σ'.

In the next paragraph we consider one such comparison of two subtrees rooted at two separating set nodes.

Isomorphism of two subtrees rooted at separating set nodes s_i and t_j. In Step 4i), we compare s_i with t_j only if the vertices of X_{s_i} can be mapped onto X'_{t_j} blockwise. In order to compute the relation $\prec_{\mathrm{T}}$, we have to decide whether $\sigma(X_{s_i}) < \sigma'(X'_{t_j})$ for pairs X_{s_i} and X'_{t_j}. We explain the definition of the ordering $\sigma(X_{s_i}) < \sigma'(X'_{t_j})$ here in more detail with an example. Let $X_r = \{v_1, v_3, v_5, v_7\}$ and $\sigma \in Sym(X_r)$ the permutation (v_3, v_5, v_7, v_1) (written in cyclic notation). For $X_{s_i} = \{v_3, v_7\} \subseteq X_r$, $\sigma(X_{s_i})$ is defined as $\sigma(X_{s_i}) = \{\mathrm{pos}_r\,\sigma(v_3), \mathrm{pos}_r\,\sigma(v_7)\} = \{1, 3\}$. Analogously, let $X'_{r'} = \{v'_6, v'_7, v'_8, v'_9\}$ and $X_{t_j} = \{v'_6, v'_7\} \subseteq X'_{r'}$ and $\sigma' \in Sym(X'_{r'})$ be the identity permutation. $\sigma'(X'_{t_j}) = \{\mathrm{pos}_{r'}\,\sigma'(v'_6), \mathrm{pos}_{r'}\,\sigma'(v'_7)\} = \{1, 2\}$. Since $(1, 2) < (1, 3)$ (lexicographical comparison of the sets with the entries arranged in increasing order) we have $\sigma(X'_{t_j}) < \sigma(X_{s_i})$.

In Step 4ii), we compare k_i with k_j, the number of children of s_i and of t_j. To compute k_i, we can use the functions FirstChild(s_i) and count how often NextSibling returns a further child of s_i and increment this number once at the end.

In Step 4iii), assume $k_i = k_j = m$. We consider the induced bipartite subgraphs $B_G[X_{s_i}, X_{a_{i,1}}], \ldots, B_G[X_{s_i}, X_{a_{i,m}}]$ and $B_H[X'_{t_j}, X'_{a'_{j,1}}], \ldots, B_H[X'_{t_j}, X'_{a'_{j,m}}]$. Intuitively, we partition the children of s_i and t_j into classes where the bipartite subgraphs are isomorphic. Again we use similar arguments as in [13]. We run through the children of s_i and t_j in a fixed order, using the functions FirstChild and NextSibling. We find the bipartite subgraph, say $B_G[X_{s_i}, X_{a_{i,1}}]^\sigma$ which is the smallest, i.e. for which there exists a mapping $\phi \in Sym(X_{a_{i,1}})$ such that for all $j' \in \{1, \ldots, m\}$ and $\phi' \in Sym(X'_{a'_{j,j'}})$ it holds $\sigma\phi(B_G[X_{s_i}, X_{a_{i,1}}]) \leqslant \sigma'\phi'(B_H[X'_{t_j}, X'_{a'_{j,j'}}])$. Via cross comparisons, the algorithm runs through all bipartite subgraphs (of s_i with all siblings of $a_{i,1}$) in increasing order (and also through all bipartite subgraphs of t_j and all its children $a_{j,j'}$).

This is done as follows: When comparing $B_G[X_{s_i}, X_{a_{i,1}}]$ and $B_H[X'_{t_j}, X'_{a'_{j,j'}}]$ for some j', the algorithm records all that mappings ϕ and ϕ', where $\sigma\phi(B_G[X_{s_i}, X_{a_{i,1}}])$ and $\sigma'\phi'(B_H[X'_{t_j}, X'_{a'_{j,j'}}])$ become minimal. This builds the set $\Theta_{a_{i,1},a'_{j,j'}}$. If the set is empty, then both bipartite subgraphs are not isomorphic and we proceed with the next pair of bipartite subgraphs in the cross comparison procedure. Thereby, the algorithm compares the number of bipartite subgraphs which are found to be isomorphic to the current one. For example, if $B_G[X_{s_i}, X_{a_{i,1}}]$ has less isomorphic siblings than the isomorphic bipartite subgraph $B_H[X'_{t_j}, X'_{a'_{j,j'}}]$, then we return $S_{s_i} \prec_{\mathrm{T}} T_{t_j}$. If the numbers are equal then we invoke Step 4iv) for all these isomorphic siblings. After this, we proceed with the next class of isomorphic bipartite subgraphs larger than $B_G[X_{s_i}, X_{a_{i,1}}]^\sigma$.

In Step 4iv), we start with two isomorphic bipartite subgraphs $B_G[X_{s_i}, X_{a_{i,1}}]$ and $B_H[X'_{t_j}, X'_{a'_{j,j'}}]$. For both of them, we consider all the siblings which have an isomorphic bipartite subgraph. Thereby, we traverse the children of node s_i in a fixed order. Namely, we can reach all these siblings of $a_{i,1}$ with the function NextSibling. Again, this is done via cross comparisons. Consider one such pair, say $a_{i,1}$ and $a'_{j,j'}$. We recompute the set $\Theta_{a_{i,1},a'_{j,j'}}$ and go into recursion at the corresponding subtrees $S_{a_{i,1}}$ and $T_{a'_{j,j'}}$ with the set $\Theta_{a_{i,1},a'_{j,j'}}$. We compute the number of siblings of $a_{i,1}$ which are equal up to Step 4iv) and if this number is equal to the number of siblings of $a'_{j,j'}$ then we proceed with the next test. We run through all the siblings and we do this test for all classes of isomorphic bipartite subgraphs and return $S_{s_i} =_{\mathrm{T}} T_{t_j}$.

Complexity of the isomorphism order algorithm. We show that the isomorphism order between two augmented trees of bounded tree distance width graphs can be computed in logspace.

Steps 1, 2 and 3 of the isomorphism order can be done in logspace, as we compute the size of subgraphs, the number of children and check the correctness of a partial isomorphism. Because we have a tree distance decomposition where bags have constant size, the whole graph is partitioned into separating sets of constant size. Hence, for a partial isomorphism from bag X_r onto bag $X'_{r'}$ we store the current mappings σ and σ' with $O(1)$ bits on the work-tape. We also store $\Theta_{(r,r')}$ in $O(1)$ bits. Thereby, we rename the vertices according to the lexicographical order of their labels from the input. The mapping from X_r onto $X'_{r'}$ is given by $\sigma\sigma'^{-1}$. Whether this is a partial isomorphism which fits to the partial isomorphism of the parents of X_r and $X'_{r'}$ (if we are in recursion, having a look at the work-tape contents stored one level up in recursion) can be checked with constant effort.

For this task, in Step 4 we have counters on the work-tape. For the partitioning in Step 4i), we need $O(1)$ bits on the work-tape, because there are at most $O(1)$ different separating sets only. For the partitioning in Step 4ii), we need $O(\log k)$ bits when considering nodes like s_i with $\geqslant k$ children. We can recompute these numbers and do not keep them after the comparison. For the partitioning in Step 4iii), we need $O(1)$ bits, because the bipartite graphs (like e.g. $B[s_i, a_{i,i'}]$) are of constant size and therefore there are at most $O(1)$ different bipartite graphs only. For the partitioning in Step 4iv), we need $O(\log k)$ bits when considering an isomorphism class with members, say like $a_{i,i'}$, of size $|S_{a_{i,i'}}| = n/k$. Note, there are $\leqslant k$ such members in that class.

In order to have only a logarithmic number of recursive calls there is one special situation in which we have to diverge from the isomorphism order procedure.

Definition 3.6. In an augmented tree of size n, a *large child* of a node is a subtree rooted at a child, which is of size $\geqslant n/2$.

It is important for the logspace bound not to store bits on the work-tape before going into recursion on such a large child. Each node can have only one large child. Say s_1 and t_1 are large children of r and r', respectively (accordingly, we can have a and a' as large children of s_1 and t_1). Before doing any computation we directly go into recursion. When returning from the recursion we return a constant size table Θ_0 of all the partial isomorphisms from X_{s_1} onto X'_{t_1} which correspond to the minimal isomorphism order. If the table is not empty then we recompute $\Theta_{(r,r')}$ and update it, i.e. $\Theta_{(r,r')} \leftarrow \Theta_{(r,r')} \cap \Theta_0$. If there is no partial isomorphism from X_{s_1} onto X'_{t_1} then there is no isomorphism from X_a onto $X'_{a'}$ and we return one further level up in recursion.

We summarize, when going into recursion at bag nodes we only store $O(1)$ bits, i.e. the current mapping σ (or a table of mappings of the large child) of the bag node r. This order also gives an order on the children of r. Note, r can have only $O(1)$ children because the children correspond to minimum separating sets, i.e. different subgraphs of X_r and there are only $O(1)$ possibilities for this.

When going into recursion at a separating set node s_1, there can be many children. Let $|T_{s_1}| = n$. To partition these children into isomorphism classes we keep counters on the work-tape. First, we distinguish them by the fixed order of the parent bag node, we can recompute this primary order. Second, we distinguish them by the size of their subtrees. Hence, in one isomorphism class are only children of equal size. Therefore we keep counters on the work-tape to distinguish the children in the current isomorphism class. With cross comparisons, as done by Lindell in [13], we can compute and check the number of isomorphic children in each class. For these counters we need $O(\log k_j)$ bits if the j-th isomorphism class has k_j members. Since in an isomorphism class the members have equal size, the subtrees have size $\leqslant N/k_j$, where N be the size of the augmented tree. We conclude that we get the same recurrence for the space $\mathcal{S}(N)$ as Lindell:

$$\mathcal{S}(N) \leqslant \max_j \mathcal{S}\left(\frac{N}{k_j}\right) + O(\log k_j),$$

where $k_j \geqslant 2$ for all j. Thus $\mathcal{S}(N) = O(\log N)$. Note that the number n of vertices of G is in general smaller than N, because the vertices of the separating sets (of a separating set node) occur also in the bag associated to the parent node in the augmented tree. Since there are only a constant number of children for a bag node, the size of the augmented tree is polynomial in the size of the associated graph. This proves the theorem.

Theorem 3.7. *The isomorphism problem for graphs of bounded tree distance width is in* L.

Canonization of bounded tree distance width graphs. We use the isomorphism order algorithm as a subroutine for the canonization of the augmented tree S. We traverse S while computing the tree isomorphism order as in Lindell [13] to output the canon of each of the nodes along with delimiters. That is, we output a '[' while going down a subtree, and ']' while going up a subtree.

We need to choose a bag node as root for the tree. Since there is no distinguished bag, we simply cycle through all of them in logspace, determining the set which, when chosen as the root, leads to the lexicographically minimum canon of the augmented tree S. We describe the canonization procedure for a fixed root r.

The canonization procedure has two steps. In the first step we compute what we call a *canonical list* for S_r. Assume, that we can pre-compute a table where we have for each graph of size $\leqslant k$ its canon. We can use this for example, if the isomorphism order algorithm reaches a leaf in the augmented tree. This canon is given by arranging the edges of the graphs in a unique order. In the second step we compute the final canon from the canonical list.

For the canon of a subtree rooted at a bag node r, we compute the minimal mapping σ for r, invoking the isomorphism order algorithm. The canon begins with σ. According to the order of σ we order the children and output their canons in increasing isomorphism order.

For the canon of a subtree rooted at a separating set node s_1, we invoke the isomorphism order algorithm to arrange the children of s_1. This is done via cross comparisons of the subtrees rooted at these children. The canon begins with $\sigma|_{s_1}$ (i.e. the order of σ restricted to the vertices of s_1), followed by the canons of the subtrees in increasing isomorphism order.

We give an example: Consider the canonical list $l(S, r)$ of edges for the tree S_r of Fig. 1. Let $\sigma_{i,j}$ be the minimum mapping of the subtree rooted at bag node $a_{i,j}$.

$$\begin{aligned}
&l(S, r) = \big[(\sigma)\, l(S_{s_1}, s_1)\ \ldots\ l(S_{s_l}, s_l)\big], \quad \text{where}\\
&l(S_{s_1}, s_1) = \big[(\sigma|_{s_1})\, l(\sigma_{1,1}, a_{1,1})\ \ldots\ l(\sigma_{1,k_1}, a_{1,k_1})\big],\\
&\quad\vdots\\
&l(S_{s_l}, s_l) = \big[(\sigma|_{s_l})\, l(\sigma_{l,k_l}, a_{l,k_l})\big].
\end{aligned}$$

Canon for the original graph of bounded tree distance width. This list is now almost the canon, except that the names of the vertices are still the ones they have in G. Clearly, a canon must be independent of the original names of the nodes. The

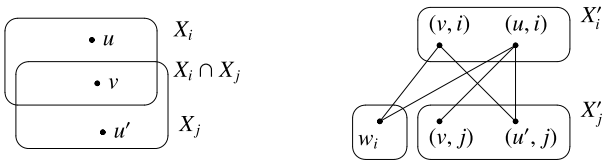

Fig. 2. The figure shows the situation where X_i and X_j are copied. For the vertices u, u', v and w_i the new edges from Step 2 and Step 3 are drawn.

final canon for S_r can be obtained by a logspace transducer which relabels the vertices in the order of their first occurrence in this canonical list and outputs the list using these new labels.

To get the canon for G, we remove the delimiters '[' and ']' in the canon for S_r and order the edges of G in lexicographical order using the new labels. This is sufficient, because we describe here a bijective function f which transforms an automorphism ϕ of S_r into an automorphism $f(\phi)$ for G with X_r fixed. This proves the theorem.

Theorem 3.8. *A graph of bounded tree distance width can be canonized in logspace.*

4. Graphs of bounded treewidth

In this section we consider several isomorphism problems for graphs of bounded treewidth. We are interested in isomorphisms *respecting* the decompositions (i.e. vertices are mapped blockwise from a bag to another bag). We show first that if the tree decomposition of both input graphs is part of the input then the decomposition respecting isomorphism problem can be decided in L. We then show that if a tree decomposition of only one of the two given graphs is part of the input, then the isomorphism problem is in LogCFL. It follows that the isomorphism problem for graphs of bounded treewidth is also in LogCFL.

Assume first the decompositions of both input graphs are given. In order to prove that this problem is in L, we show that given tree decompositions together with designated bags as roots for G and H the question of whether there is an isomorphism between the graphs mapping root to root and respecting the decompositions can be reduced to the isomorphism problem for graphs of bounded tree distance width. We argued in the previous section that this problem belongs to L.

Theorem 4.1. *The isomorphism problem for bounded treewidth graphs with given tree decompositions reduces to isomorphism for bounded tree distance width graphs under* AC^0 *many-one reductions.*

Proof. Let (G, D, r), (H, D', r') be two graphs together with tree decompositions D and D' of width k with root bags X_r and $X'_{r'}$. We describe a function which transforms (G, D, r) into $(\widehat{G}, S)$ where $\widehat{G}$ is a graph of bounded tree distance width k and S is a root set for a minimum tree distance decomposition of width k of $\widehat{G}$. This function also transforms (H, D', r') into $(\widehat{H}, S')$. We will show that this happens in such a way that $(\widehat{G}, S)$ is isomorphic to $(\widehat{H}, S')$ if and only if there is an isomorphism between G and H respecting the decompositions D and D' and mapping the vertices in the root bag r to vertices in the root bag r'.

Let $D = (\{X_i \mid i \in I\}, T = (I, F), r)$ be the given tree decomposition of G. W.l.o.g. we can assume that every bag in D includes at least two vertices. We define $S := X_r$. $\widehat{G}$ is defined as follows (see Fig. 2):

1. For each bag X_i in D and each vertex v in X_i, we define the vertex (v, i) in $V(\widehat{G})$. If $u, v \in X_i$ and there is an edge $\{u, v\} \in E(G)$ then we define the edge $\{(u, i), (v, i)\} \in E(\widehat{G})$.
2. For all $\{i, j\} \in F$, $u \in X_i$ and $v \in X_j$ with $u \neq v$, we define an edge between (u, i) and (v, j).
3. For all i, we define a vertex $w_i \in V(\widehat{G})$ which is connected to (v, i), for all $v \in X_i$.

From H, the graph $\widehat{H}$ is defined in the same way. We consider the minimal tree distance decomposition $\widehat{D}$ of $\widehat{G}$ with root set S. For each bag X_i in D, in Step 1 the vertices of this bag are copied in $\widehat{G}$. By considering these bags as a tree distance decomposition of $\widehat{G}$, the distance from the root set S to such a bag is equal to the distance of r to i in T. This is, because if (i, j) is an edge in T then for every vertex (u, i) in $\widehat{G}$ there is an edge to at least one vertex (v, j). It also holds that the minimal tree distance decomposition of $\widehat{G}$ with root S has width k.

If G is isomorphic to H with an isomorphism respecting the decompositions and mapping the vertices from the root bag X_r of G to the root bag of H, then clearly $\widehat{G}$ is isomorphic to $\widehat{H}$. For the other direction, observe that the edges connecting the vertices inside each bag are kept by Step 1 in the definition of $\widehat{G}$ and $\widehat{H}$. By Step 3, we guarantee, that in an isomorphism between $\widehat{G}$ and $\widehat{H}$ the vertices in one bag are all mapped blockwise to vertices in some bag, i.e. they are not split and mapped onto vertices of two or more bags. In Step 2, we distinguish between vertices in $X_i \cap X_j$ and the other vertices. That is, for an edge $(i, j) \in T$, every vertex (u, i) is connected to every vertex (v, j) except to (u, j) (in case u belongs to $X_i \cap X_j$ in D). Since all the copies of vertex u (all the vertices (u, i) for some i in $\widehat{G}$) belong to a connected subtree, this implies that in a possible isomorphism between $\widehat{G}$ and $\widehat{H}$ all copies of vertex u in $\widehat{G}$ have to be mapped blockwise to copies of the same vertex in $\widehat{H}$.

It follows that there is an isomorphism between $\widehat{G}$ and $\widehat{H}$ if and only if there is an isomorphism between G and H which respects the bags in the decompositions D and D' together with r and r', accordingly. □

From Theorem 4.1 we get the following corollary.

Corollary 4.2. *For every $k \geqslant 1$ there is a logarithmic space algorithm that, on input a pair of graphs together with a tree decompositions of width k for each of them, decides whether there is an isomorphism between the graphs, respecting the decompositions.*

Proof. The result follows from the previous reduction and Theorem 4.1. Thereby we fix a root in D and run through all possibilities for bags as roots in D'. As there are only polynomially many bags in D' this can be done by a logspace machine. □

In the previous reduction, we have transformed a graph G given together with a tree decomposition D of width k and with root bag r into a new graph $\widehat{G}$ and a root set S such that the minimal tree distance decomposition of $\widehat{G}$ has width k. As done in Section 3 we can compute within logspace an augmented tree for $\widehat{G}$. Moreover we can use the defined total isomorphism order on augmented trees to compare in this way graphs of bounded treewidth given together with tree decompositions.

Corollary 4.3. *For any $k > 1$ there is a total order $<_{\mathrm{T}}$ defined on the set of tuples (G, D, r) where G is a graph of treewidth k, and D a tree decomposition of G with root set r. $<_{\mathrm{T}}$ can be computed in logarithmic space and $(G, D, r) =_{\mathrm{T}} (H, D', r')$ if and only if there is an isomorphism between G and H respecting the decompositions and mapping the root set of r to that of r' (i.e. X_r to $X'_{r'}$ blockwise).*

This result will be used in the next section for computing an isomorphism when just one of the decompositions is given.

4.1. A LogCFL algorithm for isomorphism

We consider now the more difficult situation in which only one of the input graphs is given together with a tree decomposition.

Theorem 4.4. *Isomorphism testing for two graphs of bounded treewidth, when a tree decomposition for one of them is given, can be done in* LogCFL.

Proof. We describe an algorithm which runs on a non-deterministic auxiliary pushdown automaton (NAuxPDA). Besides a read-only input tape and a finite control, this machine has access to a stack of polynomial size and an $O(\mathrm{L}n)$ space bounded work-tape. On the input tape we have two graphs G, H of treewidth k and a tree decomposition $D = (\{X_i \mid i \in I\}, T = (I, F), r)$ for G. For $j \in I$ we define G_j to be the subgraph of G induced on the vertex set $\{v \mid v \in X_i,\ i \in I$ and $i = j$ or i a descendant of j in $T\}$. That is, G_j contains the vertices which are separated by the bag X_j from X_r and those in X_j. We define $D_j = (\{X_i \mid i \in I_j\}, T_j = (I_j, F_j), j)$ as the tree decomposition of G_j corresponding to T_j, the subtree of T rooted at j. We also consider a way to order the children of a node in the tree decomposition:

Definition 4.5. Given a graph G together with a tree decomposition D, let $1, \ldots, l$ be the children of a node r in the decomposition tree T. We define the *lexicographical subgraph order*, as the order among the subgraphs $G_1, \ldots, G_l$ which is given by: $G_i < G_j$ iff there is a vertex $w \in V(G_i) \setminus X_r$ which has a smaller label than every vertex in $V(G_j) \setminus X_r$.

The algorithm non-deterministically guesses two main structures. On the one hand it guesses a tree decomposition of width k for H. This is done in a similar way as in the LogCFL algorithm from Wanke [19] for testing that a graph has bounded treewidth. We briefly sketch this method which is the basis of our algorithm. Second, we guess an isomorphism ϕ from G to H by extending partial mappings from bag to bag.

Algorithm for tree decomposition testing. For completeness we include here a sketch of Wanke's algorithm [19] for testing whether a graph has treewidth k. On input a graph G the algorithm guesses non-deterministically the bags in the decomposition using the pushdown to test that these bags fulfill the properties of a tree decomposition and that every edge in G is included in some bag. If the guessed bags determine a tree decomposition of width k the algorithms accepts.

Let G be the connected input graph. P and Q denote vertex sets of size $\leqslant k+1$ in G which are additionally separating sets and play the role of bags in the tree decomposition. For a separating set P in G and a vertex $v \notin P$, let $\Phi_G(P, v)$ be the split component of P in G containing v. For technical reasons initially we extend G defining an extra vertex v_0 and connecting it arbitrarily to a vertex u with the property that $\{u\}$ is not an articulation point in G. We will assume that v_0 has a label with smaller number than all the original vertices in G. We also consider arbitrarily some other vertex $w \neq u$ in G. The algorithm is started with the initial bag $P = \{v_0, u\}$ and the initial vertex w, representant of the unique split component of P. Then it guesses non-deterministically a new bag Q in the decomposition and goes in recursion with each

Algorithm 1 Tree decomposition testing Decompose(G, v_0, P, v).

Input: Graph G, vertex v_0, separating set P with $|P| \leqslant k+1$, vertex v in a split component of P
Output: Accept iff the graph induced by $P \cup \Phi_G(P, v)$ has a tree decomposition of width k

1: Non-deterministically choose Q of size $\leqslant k+1$ in $\Phi_G(P, v) \cup P$
2: **if** $Q \subseteq P$ or $P \subseteq Q$ or
$\exists \{u_1, u_2\} \in E(G)$: $u_1 \in \Phi_G(P, v) \wedge u_2 \notin \Phi_G(P, v) \cup Q$
then halt and reject
3: **for all** w having smallest number in a split component of Q except v_0
4: go into recursion with Decompose(G, v_0, Q, w)
5: **end for**
6: **if** the stack is not empty **then** go one level up in recursion
7: halt and accept

of the split components defined by Q. This is done with the procedure Decompose(G, v_0, P, v) where G is the original graph, v_0 the extra vertex, P the actual separating set (bag) and v a representant of a split component of P. The initial call is Decompose$(G, v_0, \{v_0, u\}, w)$ where u and w are chosen arbitrarily, where u is not an articulation point in G and $w \neq u$.

P and the sequence of bags Q defined in an accepting non-deterministic computation define a tree decomposition of G (once the vertex v_0 is deleted from them).

In every iteration the algorithm chooses Q, a neighbor bag of P in a tree decomposition of G. In Line 2 it is required that Q is not contained in P nor P in Q and Q must separate its split components in the subgraph $\Phi_G(P, v)$ from the vertices in $P \setminus Q$. This requirement guarantees that the new bag fulfills condition iii) in the definition of tree decomposition. In Line 3 the algorithm goes into recursion at each split component of Q, except the one which contains v_0. The algorithm recursively chooses separating sets this way from the root through the whole graph.

Algorithm for isomorphism testing. We modify the algorithm of Wanke in a way that we can test isomorphism in parallel. Namely, our algorithm simulates Wanke's algorithm as a subroutine. In the description of the new algorithm we concentrate on the isomorphism testing part and hide the details of how to choose the bags. For simplicity the sentence "guess a bag X_j in H according to Wanke's algorithm" means that we simulate the guessing steps from Wanke, checking at the same time that the constructed structure is in fact a tree decomposition. Note, if the bags were not chosen appropriately, then the algorithm would halt and reject.

The algorithm starts guessing a root bag $X'_{r'}$ of size $\leqslant k+1$ for a decomposition of H. With $X'_{r'}$ as root bag it guesses step by step the tree decomposition D' of H which corresponds to D and its root r. It also constructs a mapping ϕ describing a partial isomorphism from the vertices of G onto the vertices of H. At the beginning, ϕ is the empty mapping and the algorithm guesses an extension of ϕ from X_r onto $X'_{r'}$ that is stored on the top of the stack. In general when dealing with a set of vertices from a bag X_a in D, the algorithm cycles through all possible subsets S of X_a and considers the children i of a in D with $X_a \cap X_i = S$. Note, S is the separating set of minimum size which separates $X_i \setminus X_a$ from the root bag X_r. The corresponding subgraphs G_i of G are then partitioned in isomorphism classes respecting the decomposition D_i given in the input. This is done considering the total isomorphism order of (G_i, D_i, X_i) as defined in Corollary 4.3. The algorithm compares the children of a with separating set S with guessed children of a' (with separating set $\phi(S)$) testing that for each isomorphism class there is the same number of isomorphic subgraphs with root a' in H. For this the algorithm uses the lexicographical subgraph order (Definition 4.5) to go through the isomorphic siblings from left to right, just keeping a pointer to the current child on the work-tape, so that no child is counted twice. For two such children, say s_1 of a and t_1 of a', the algorithm checks then recursively that (G_1, D_1) is isomorphic to the corresponding subgraph of t_1 in H, by an extension of ϕ. Inside an isomorphism class the subgraphs have the same intersection with X_a and are isomorphic to each other. Because of this, they are interchangeable and could be mapped to subgraphs in the corresponding isomorphism class from $X'_{a'}$ in any order. The isomorphism computed by the algorithm maps these subgraphs to subgraphs in the isomorphism class from $X'_{a'}$ in lexicographical subgraph order.

When the algorithm goes into recursion, it pushes on the stack $O(\log n)$ bits for a description of X_a and $X'_{a'}$ as well as a description of the partial mapping ϕ from X_a onto $X'_{a'}$ and the description of S.

In general, not all the information about ϕ is kept on the stack. We only have the partial isomorphism $\phi : \{v \mid v \in X_r \cup \cdots \cup X_a\} \to \{v \mid v \in X'_{r'} \cup \cdots \cup X'_{a'}\}$, where $r, \ldots, a$ ($r', \ldots, a'$, respectively) is a simple path in T from the root to the node at the current level of recursion. After the algorithm runs through all children of some node (going through all its subsets S) it goes one level up in recursion and recomputes all the other information which is given implicitly by the subtrees from which it returned. Suppose the control of the algorithm returned to the bag X_a, from a child X_i with $X_a \cap X_i = S$ after checking that the partial isomorphism can be extended to map X_i to $X'_{i'}$. It then has to do the following:

- Copy from the top of the stack into the work-tape the partial isomorphism ϕ of the bags X_a onto $X'_{a'}$.
- Compute the lexicographical next isomorphic sibling of X_i with minimum sized separating set S and guess the lexicographical next isomorphic sibling of $X'_{i'}$ with minimum sized separating set $\phi(S)$. Check that the guessed sibling satisfies the decomposition properties that the isomorphism can be extended to include the new subgraphs.

Algorithm 2 Isomorphism testing with one tree decomposition TWIso($G, H, D, X_a, X'_{a'}$).

Input: Graphs G, H, tree decomposition D for G, bags X_a in G and $X'_{a'}$ in H
Top of Stack: Partial isomorphism ϕ mapping the vertices in the parent bag of X_a onto the vertices in the parent bag of $X'_{a'}$
Output: Accept, iff G_a is isomorphic to H_a by an extension of ϕ

1: Guess an extension of ϕ to a partial isomorphism from X_a onto $X'_{a'}$
2: **if** ϕ cannot be extended to a partial isomorphism which maps X_a onto $X'_{a'}$
then halt and reject
3: **for** each subset $S \subseteq X_a$ **do**
4: Let $1, \ldots, l$ be the children of a in T with $X_a \cap X_i = S$ partition the subgraphs corresponding to the subtrees of T rooted at $1, \ldots, l$ into (decomposition respecting) isomorphism classes $E_1, \ldots, E_p$
5: **for** each class E_j from $j = 1$ to p **do**
6: **for** each subtree $T_i \in E_j$ (in lexicographical subgraph order) **do**
7: guess a bag $X'_{i'}$ in H in increasing lexicographical subgraph order of $H_{i'}$
8: **if** $X'_{i'}$ is not a correct child bag of $X'_{a'}$ (see Wanke's algorithm)
then halt and reject
9: Invoke TWIso($G_i, H_{i'}, D_i, X_i, X'_{i'}$) recursively and push X_a, $X'_{a'}$ and the partial isomorphism ϕ on the stack
10: After recursion pop these informations from the stack
11: **end for**
12: **end for**
13: **end for**
14: **if** the stack is not empty **then** go one level up in recursion
15: halt and accept

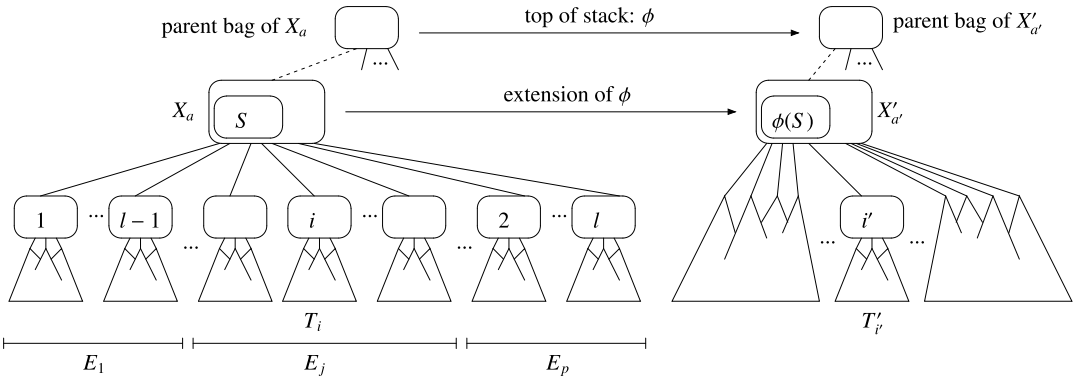

Fig. 3. The figure illustrates the setting of TWIso($G, H, D, X_a, X'_{a'}$) in Algorithm 2. Let T be the tree of tree decomposition D, where X_a and $X'_{a'}$ are root bags. The connection to their parents is indicated by dashed lines. Exemplarily, on the left side a subset S of X_a is shown which induces children $1, \ldots, l$ as computed in Algorithm 1. The children are partitioned into isomorphism classes $E_1, \ldots, E_p$. Correspondingly, the extension of ϕ induces on the right side $\phi(S)$, a subset of $X'_{a'}$.

- If there is no such sibling then look for the next isomorphism class (Corollary 4.3) of a subtree with minimum sized separating set S and look for the lexicographical first child of X_a inside this class.
- If there is no higher isomorphism class of subtrees with intersection S then go to the next subset $S' \subseteq X_a$.
- If there is no further subset S' then the algorithm has visited all children of X_a and it is ready to further return one level up in the recursion.

Algorithm 2 summarizes the above considerations. In Line 1, it guesses an extension of the partial isomorphism ϕ to include a mapping from X_a onto $X'_{a'}$. We the partial isomorphism of their parent bags can be found on the top of the stack.

In Line 3 the algorithm cycles through all the subsets of X_a. The partition in Line 4 can be obtained in logspace by decomposition respecting isomorphism tests of the tree structures. Two subtrees rooted at X_i and X_j are in the same isomorphism class if and only if $(G_i, D_i, S) =_T (G_j, D_j, S)$.

In Lines 7 to 10, the algorithm guesses the bag $X'_{i'}$ in H which corresponds to X_i and tests recursively whether the corresponding subgraphs G_i and $H_{i'}$ are isomorphic with an extension of the partial isomorphism ϕ. Observe that the algorithm cannot guess the same bag $X'_{i'}$ in H for two different bags X_i and X_j in G. This is because if the corresponding subgraphs G_i and G_j are isomorphic the bags in H are chosen in increasing lexicographical order (Line 7) and must be different. On the other hand if G_i and G_j are not isomorphic then the subgraph of H defined by $X'_{i'}$ cannot be isomorphic to both of them.

In Line 8, the algorithm checks whether $X'_{i'}$ fulfills the properties of a correct tree decomposition as in Wanke's algorithm (i.e. $X'_{i'}$ must be a separating set which separates its split components from the vertices in$X'_{a'} \setminus X'_{i'}$) (see Fig. 3).

To show that the algorithm correctly computes an isomorphism, we make the following observation. A bag X_a and a subset $S \subseteq X_a$ constitute a separating set defining the connected subgraphs $G_1, \ldots, G_l$. These subgraphs do not contain

the root X_r and $V(G_i) \cap V(G_j) = S$ since we have a tree decomposition D. The algorithm guesses and keeps from the partial isomorphism ϕ exactly those parts which correspond to the path from the roots X_r and $X'_{r'}$ to the current bags X_a and $X'_{a'}$. Once it verified a partial isomorphism from one child component (e.g. G_i) of X_a onto a child component (e.g. $H_{i'}$) of $X'_{a'}$, for the other child components it suffices to know the partial mapping of ϕ from X_a onto $X'_{a'}$.

Observe that for each v in G in a computation path from the algorithm there can only be a value for $\phi(v)$, since in the decomposition all the appearances of vertex v belong to bags from a connected subtree in D. Clearly, if G and H are isomorphic then the algorithm can guess the decomposition of H which fits to D, and the extensions of ϕ correctly. In this case the NAuxPDA has some accepting computation. On the other hand, if the input graphs are non-isomorphic then in every non-deterministic computation either the guessed tree decomposition of H does not fulfill the conditions of a tree decomposition (and would be detected) or the partial isomorphism ϕ cannot be extended at some point. □

Wanke's algorithm decides in LogCFL whether the treewidth of a graph is at most k by guessing all possible tree decompositions. Using a result from [8] it follows that there is also a (functional) LogCFL algorithm that on input a bounded treewidth graph computes a particular tree decomposition for it. Since LogCFL is closed under composition, from this result and Theorem 4.4 we get:

Corollary 4.6. *The isomorphism problem for bounded treewidth graphs is in* LogCFL.

5. Conclusions and open problems

We have shown that the isomorphism problem for graphs of bounded treewidth is in the class LogCFL and that for the more restricted case of bounded tree distance width graphs the problem is complete for L. Moreover for this second class of graphs we also give a logspace algorithm for the canonization problem. By using standard techniques in the area it can be shown that the same upper bounds apply for other problems related to isomorphism on these graph classes. For example the problem of deciding whether a given graph has a non-trivial automorphism or the functional versions of automorphism and isomorphism can be done within the same complexity classes. The main question remaining is whether the LogCFL upper bound for isomorphism of bounded treewidth graphs can be improved. No LogCFL-hardness result for the isomorphism problem is known, so maybe the result can be improved. Proving a logspace upper bound for the isomorphism problem of bounded treewidth graphs requires more than the computation of tree decompositions within logarithmic space, a result that has been recently obtained [7]. Another interesting open question is whether bounded treewidth graphs can be canonized in LogCFL.

References

[1] V. Arvind, P. Kurur, T.C. Vijayaraghavan, Bounded color multiplicity graph isomorphism is in the #L hierarchy, in: Proc. 20th IEEE CCC, 2005, pp. 13–27.
[2] H.L. Bodlaender, Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees, Journal of Algorithms 11 (1990) 631–643.
[3] H.L. Bodlaender, A partial k-arboreum of graphs with bounded treewidth, Theoretical Computer Science 209 (1998) 1–45.
[4] H.L. Bodlaender, A. Koster, Combinatorial optimization of graphs of bounded treewidth, The Computer Journal (2007) 631–643.
[5] S. Datta, N. Limaye, P. Nimbhorkar, T. Thierauf, F. Wagner, Planar graph isomorphism is in Logspace, in: Proc. 24th IEEE CCC, 2009, pp. 203–214.
[6] S. Datta, P. Nimbhorkar, T. Thierauf, F. Wagner, Isomorphism of $K_{3,3}$-free and K_5-free graphs is in Logspace, in: Proc. 29th FSTTCS, 2009, pp. 145–156.
[7] M. Elberfeld, A. Jakoby, Till Tantau, Logspace versions of the theorems of Bodlaender and Courcelle, in: Proceedings of the 51st Annual Symposium on Foundations of Computer Science (FOCS), 2010.
[8] G. Gottlob, N. Leone, F. Scarcello, Computing LOGCFL certificates, Theoretical Computer Science 270 (2002) 761–777.
[9] M. Grohe, O. Verbitsky, Testing graph isomorphism in parallel by playing a game, in: Proc. 33rd ICALP, 2006, pp. 3–14.
[10] B. Jenner, J. Köbler, P. McKenzie, J. Torán, Completeness results for graph isomorphism, Journal of Computer and System Sciences 66 (2003) 549–566.
[11] J. Köbler, S. Kuhnert, The isomorphism problem of k-trees is complete for Logspace, in: Proc. 34th MFCS, 2009, pp. 537–448.
[12] J. Köbler, U. Schöning, J. Torán, The Graph Isomorphism Problem, Birkhäuser, 1993.
[13] S. Lindell, A Logspace algorithm for tree canonization, in: Proc. 24th ACM STOC, 1992, pp. 400–404.
[14] E. Luks, Isomorphism of graphs of bounded valence can be tested in polynomial time, Journal of Computer and System Sciences 25 (1982) 42–65.
[15] E. Luks, Parallel algorithms for permutation groups and graph isomorphism, in: Proc. 27th IEEE FOCS, 1986, pp. 292–302.
[16] G. Miller, Isomorphism testing for graphs of bounded genus, in: Proc. 12th ACM STOC, 1980, pp. 225–235.
[17] O. Reingold, Undirected connectivity in logspace, Journal of the ACM 55 (4) (2008).
[18] I. Sudborrough, Time and tape bounded auxiliary pushdown automata, Mathematical Foundations of Computer Science (1977) 493–503.
[19] E. Wanke, Bounded tree-width and LOGCFL, Journal of Algorithms 16 (1994) 470–491.
[20] K. Yamazaki, H.L. Bodlaender, B. de Fluiter, D.M. Thilikos, Isomorphism for graphs of bounded distance width, Algorithmica 24 (1999) 105–127.