Restricted Space Algorithms for Isomorphism on Bounded Treewidth Graphs

Bireswar Das
Inst. of Mathematical Sciences
Chennai
bireswar@imsc.res.in

Jacobo Torán*
Inst. für Theoretische Informatik
Universität Ulm
jacobo.toran@uni-ulm.de

Fabian Wagner*
Inst. für Theoretische Informatik
Universität Ulm
fabian.wagner@uni-ulm.de

November 24, 2009

Abstract

The Graph Isomorphism problem restricted to graphs of bounded treewidth or bounded tree distance width are known to be solvable in polynomial time [2],[19]. 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 respecting the decompositions 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 [8].

Topics: Complexity, Algorithms, Graph Isomorphism Problem, Treewidth, LogCFL.

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. 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 [13], bounded genus [15], bounded color classes [14], or bounded treewidth [2] is known to be in P. Recently some of these upper bounds have been improved with the development

*Supported by DFG grant TO 200/2-2.

of space efficient techniques, most notably Reingold's deterministic logarithmic space algorithm for connectivity in undirected graphs [16]. In some cases logarithmic space algorithms have been obtained. For example graph isomorphism for trees [12], planar graphs [5] or k-trees [10] is known to be in the class L. In other cases the problem has been classified in a small complexity class bellow P . For example, the isomorphism problem for graphs with bounded treewidth is known to be in TC^1 [8] 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 with 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 [8] 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 by proving 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 with bounded treewidth [18, 7], 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 [8].

The notion of tree distance width, a stronger version of the treewidth concept, was introduced in [19]. 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 [9], 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 representant of the isomorphism equivalence class, can be computed in logspace. Our proof techniques extend the methods from recent logspace bounded isomorphism results [5, 6].

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 nondeterministic auxiliary pushdown machines (NAuxPDA). These are Turing machines with a logarithmic space work tape, an additional pushdown and a polynomial time bound [17]. 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:

$$L \subseteq LogCFL \subseteq TC^1.$$

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 $V \setminus U$.

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.

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 [19] 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.

Let $Sym(V)$ be the *symmetric group* on a set V.

3 Graphs with 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 [16].

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 in order through the vertices of G 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 $X_p :=$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 $> v_i$ 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.

Using these three functions the algorithm performs a depth-first traversal of T. For this 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 this 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. The algorithm 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 its father) in memory, and work in logarithmic space. On input a graph G with n nodes, and a root set S, the space used by the algorithm is therfore 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 induced by the vertices of the bags in the subtree rooted at i is connected thus producing a minimal decomposition. As observed in [19], 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 tree 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 bounded tree distance width 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 $\mathcal{T}_{(G,D)}$.*

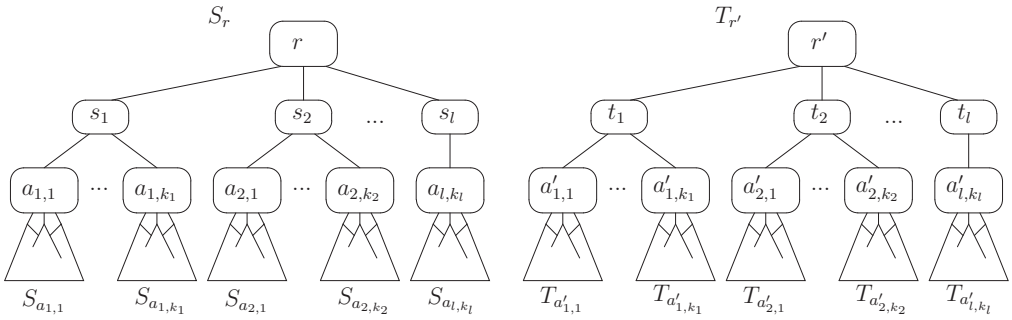

Figure 1: The augmented trees S_r and $T_{r'}$ rooted at bag nodes r and r'. Node r has separating set nodes $s_1, \ldots, s_l$ as children. The children of s_1 are again bag nodes $a_{1,1}, \ldots, a_{1,k_1}$. $S_{a_{i,j}}$ is the subtree rooted at $a_{i,j}$. Bag nodes and separating set nodes alternate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

- *In $F_{(G,D)}$ there are edges between bag nodes $a \in I$ and the separating set nodes $s^a_1, \ldots, s^a_{l(a)} \in J$ (edges between bag nodes and their children in the augmented tree). There are also edges between nodes $b \in I$ and s^a_j if $M_{s^a_j}$ 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 later say for example that $s_1, \ldots, s_l$ are the children of a bag node a if the context is clear. The odd levels of the augmented tree T' 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 might occur in more than one associated component e.g. if v is contained in a bag and 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 $\mathsf{graph}(T_a)$ as the subgraph of G induced by all the vertices associated to at least one of the nodes of T_a. The *size* of T_a, denoted $|T_a|$ is the number of vertices which occur in at least one component which is associated to a node in T_a. Note, $|T_a|$ is polynomially related to $|\mathsf{graph}(T_a)|$, i.e. the number of vertices in the corresponding subgraph of G.

When given a tree distance decomposition then the augmented tree can be computed in log-space. Using the result in Section 3.1 we immediately get:

Lemma 3.2 *Let G be a graph of bounded tree distance width. The augmented tree for G can be computed in logspace.*

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 tree distance decompositions D and D', respectively. This isomorphism order algorithm is an extension of the one for trees given by Lindell [12] and it is different from that for planar graphs given by Datta et.al.[5]. 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 Figure 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 depends on the order of the vertices in the bags r and r'. Let X_r and $X'_{r'}$ be the corresponding bags in D and D'. We define the sets of mappings $\Theta_{(r,r')} = Sym(X_r) \times Sym(X'_{r'})$. If r is not the top-level root of the augmented tree then $\Theta_{(r,r')}$ may

become restricted to a subset, when going into recursion. The isomorphism order is defined to be $S_r <_{\mathtt{T}} T_{r'}$ if there exist mappings $(\sigma, \sigma') \in \Theta_{(r,r')}$ such that one of the following holds:

1) $(G[X_r], \sigma) < (H[X'_{r'}], \sigma')$ via lexicographical comparison of both ordered subgraphs

2) $(G[X_r], \sigma) = (H[X'_{r'}], \sigma')$ but $|S_r| < |T_{r'}|$

3) $(G[X_r], \sigma) = (H[X'_{r'}], \sigma')$ and $|S_r| = |T_{r'}|$ but $\#r < \#r'$ where $\#r$ and $\#r'$ is the number of children of r and r'

4) $(G[X_r], \sigma) = (H[X'_{r'}], \sigma')$ and $|S_r| = |T_{r'}|$ and $\#r = \#r' = l$ but $(S_{s_1}, \ldots, S_{s_l}) <_{\mathtt{T}} (T_{t_1}, \ldots, T_{t_l})$ where we assume that $S_{s_1} \leq_{\mathtt{T}} \cdots \leq_{\mathtt{T}} S_{s_l}$ and $T_{t_1} \leq_{\mathtt{T}} \cdots \leq_{\mathtt{T}} T_{t_l}$ are ordered subtrees of S_r and $T_{r'}$, respectively. To compute the order between the subtrees $S_{s_i} \leq_{\mathtt{T}} T_{t_j}$ we consider

   i the lexicographical order of the minimal separating sets (s_i and t_j) in X_r and $X'_{r'}$ according to σ and σ', as the primary criterion (observe that the separating sets are subsets of X_r (resp. $X_{r'}$) and are therefore ordered by σ and σ') and

   ii pairwise the children $a_{i,i'}$ of s_i and $a'_{j,j'}$ of t_j (for all i' and j' via cross-comparisons) such that the induced bipartite graphs $B_G[s_i, a_{i,i'}]$ and $B_H[t_j, a'_{j,j'}]$ can be *matched* according to σ and σ' (i.e. $\sigma\sigma'^{-1}$ is an isomorphism) and

   iii *recursively* the subtrees rooted at the children of s_i and t_j. Note, that these children are again bag nodes. For the cross camparison of bag nodes $a_{i,i'}$ and $a'_{j,j'}$ we restrict the set $\Theta_{(a_{i,i'}, a'_{j,j'})}$ to a subset of $Sym(X_{a_{i,i'}}) \times Sym(X'_{a'_{j,j'}})$. Namely, $\Theta_{(a_{i,i'}, a'_{j,j'})}$ contains the pair $(\phi, \phi') \in Sym(X_{a_{i,i'}}) \times Sym(X'_{a'_{j,j'}})$ if $\phi\phi'^{-1}$ extends the partial isomorphism $\sigma\sigma'^{-1}$ from child $a_{i,i'}$ onto $a'_{j,j'}$ blockwise and which induces an isomorphism from $B_G[s_i, a_{i,i'}]$ onto $B_H[t_j, a'_{j,j'}]$.

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

Isomorphism of two subtrees rooted at bag nodes r and r' We have constant size components associated to the bag nodes. A log-space machine can easily run through all the mappings of X_r and $X'_{r'}$ and record the mappings which gives the minimum isomorphism order. This can be done with cross-comparison of trees (S_r, σ) and $(T_{r'}, \sigma')$ with all possible mappings σ, σ'. Later we will see, that in recursion not all possible mappings for σ and σ' are considered. Observe that $|Sym(X_r)| \in O(1)$.

The comparison of (S_r, σ) and $(T_{r'}, \sigma')$ itself can be done simply by renaming the vertices of X_r and X'_r according to the mappings σ and σ' and then comparing the ordered sequence of edges lexicographically. When equality is found then we recursively compute the isomorphism order of the subtrees rooted at the children of r and r'.

Isomorphism of two subtrees rooted at separating set nodes s_i and t_j Datta et.al. [5] decompose biconnected planar graphs into triconnected components and obtain a tree on these components and separating pairs, i.e. separating sets of size two. We have separating sets of arbitrary constant size.

Since s_i and t_j correspond to subgraphs of X_r and $X'_{r'}$, we have an order for them given by the fixed mappings σ and σ'. Therefore, we can order the children $s_1, \ldots, s_l$ and $t_1, \ldots, t_l$ according to their occurrence in X_r and $X'_{r'}$ (e.g. assume $s_i = (1, 2, 3, 7)$ according to the mapping σ and also $s_j = (1, 2, 4, 7)$, then we get $(s_i, \sigma) <_{\mathtt{T}} (s_j, \sigma)$). Hence, when comparing s_i with t_j we have to check whether both come on the same position in that order of $s_1, \ldots, s_l$ and $t_1, \ldots, t_l$. If so, then we go to the next level in the tree, to the children of s_i and t_j.

Now we have a cross comparison among the children of s_i and the children of t_j. In Steps $4i$, $4ii$ and $4iii$ we partition the children $a_{i,1}, \ldots, a_{i,l_i}$ of s_i and $a'_{j,1}, \ldots, a'_{j,l_j}$ of t_j, respectively, into isomorphism classes, step by step.

The membership of a child to a class according to Step $4i$ and $4ii$ can be recomputed. It suffices to keep counters on the work-tape to notice the current class and traversing the siblings from left to right. After these two steps, $a_{i,i'}$ and $a'_{j,j'}$ are in the same class if and only if vertices of s_i and t_j appear lexicographically at the same positions in σ and σ' and the bipartite graphs $B[s_i, a_{i,i'}]$ and $B[t_j, a'_{j,j'}]$ are isomorphic where s_i is mapped onto t_j blockwise corresponding to $\sigma\sigma'^{-1}$ in an isomorphism. In step $4iii$ we go into recursion and compare members of one class which are rooted at subtrees of the same size. When going into recursion at $a_{i,i'}$ and $a'_{j,j'}$ we consider only those mappings from $(\phi, \phi') \in \Theta_{(a_{i,i'}, a'_{j,j'})}$ which induce an isomorphism $\phi\phi'^{-1}$ from $B[s_i, a_{i,i'}]$ onto $B[t_j, a'_{j,j'}]$.

Correctness of the isomorphism order Both the bag nodes and the separating set nodes correspond to subgraphs which are basically separating sets. A bag separates all its subtrees from the root and the separating set nodes refine the bag to separating sets of minimum size. Hence, a partial isomorphism is constructed and extended from each node to its child nodes, traversing the augmented tree (the whole graph, accordingly) in depth first manner. In the recursion, the isomorphism between the roots of the current subtrees, say S_r and $T_{r'}$, is partially fixed by the partial isomorphism between their parents. With an exhaustive search we check every possible remaining isomorphism from X_r onto $X'_{r'}$ and go into recursion again partially fixing the isomorphism for the subtrees rooted at children of r and r'. By an inductive argument, the partial isomorphism described for the augmented tree can be followed simultaneously in the original graph and we get:

Theorem 3.3 *The graphs G and H of bounded tree distance width are isomorphic if and only if there is a choice of a root bag r and r' producing augmented trees S_r and $T_{r'}$ such that $S_r =_{\mathrm{T}} T_{r'}$.*

Complexity of the isomorphism order algorithm We analyze the complexity of the isomorphism order algorithm showing 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 of 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 can even afford to 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 $O(1)$ bits, because the bipartite graphs, say we consider $B[s_i, a_{i,i'}]$, are of constant size and therefore there are at most $O(1)$ different bipartite graphs only. For the partitioning in step $4iii$, 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 $\leq k$ such members in that class.

In oder 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 3.4 *Consider an augmented tree of size n which has a subtree rooted at a child of size $\geq n/2$. This is called a* large child.

It is important for the log-space bound not to store bits on the work-tape before going into recursion on such a large child. For each node there can be only one large child. Say s_1 and t_1 are large children of r and r', respectively (the same holds for the case where the bag nodes a and a' are 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. $\Theta_{(r,r')} = \Theta_{(r,r')} \cap \Theta_0$. If there is no partial isomorphism from X_{s_1} onto X'_{t_1} then there is no hope to find an 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, say 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 [12], 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. As in an isomorphism class the members have equal size, the subtrees have size $\leq n/k_j$.

Let N be the size of the augmented tree. We conclude that we get the same recurrence for the space $\mathcal{S}(N)$ as Lindell:

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

where $k_j \geq 2$ 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. As 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 3.5 *The isomorphism order between two augmented trees of bounded tree distance width graphs can be computed in logspace.*

Canonization of bounded tree distance width graphs. Once we know the order among the subtrees, it is straight forward to output the canon of the augmented tree T. We traverse T while computing the tree isomorphism order as in Lindell [12], outputting 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, say 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 constant size graph its canon. This is needed for example, if the isomorphism order algorithm reaches a leaf in the augmented tree. Assume, that the canon is given by arranging the edges of the graphs in a canonical 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 partially 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 Figure 1. Let $\sigma_{i,j}$ be the minimum mapping of the subtree rooted at bag node $a_{i,j}$.

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

Canon for the original graph with bounded tree distance width. This list is now almost the canon, except that the names of the nodes are still the ones they have in G. Clearly, a canon must be independent of the original names of the nodes. The final canon for S_r can be obtained by a log-space 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, 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 3.6 *A graph with bounded tree distance width can be canonized in logspace.*

4 Graphs with bounded treewidth

In this section we consider several isomorphism problems for graphs with bounded treewidth. We first show that if the tree decomposition of both input graphs is part of the input and we only search for isomorphisms respecting the decompositions (vertices in the same bag have to be mapped to vertices in the same bag) then the isomorphism problem can be decided in L. We also 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 with bounded tree width is also in LogCFL.

Assume the decompositions of both input graphs are given. Let $(G,D),(H,D')$ be two bounded treewidth graphs together with tree decompositions D and D', respectively. We consider the isomorphism problem for the graphs G and H respecting the decomposition. We look for an isomorphism between G and H satisfying the condition that the images of the vertices in one bag in D belong to the same bag in D'.

We prove that this problem is in L. For this 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 with bounded tree distance decomposition. We proved in the previous section that this problem belongs to L. The proof of this result is given in the appendix.

Theorem 4.1 *Isomorphism for bounded treewidth graphs with given tree decompositions reduces to isomorphism for bounded 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 $\widehat{G}$. 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

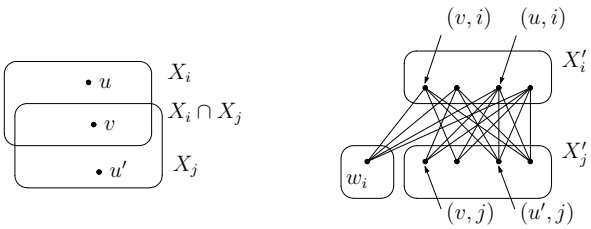

Figure 2: We reduce a treewidth graph into a new graph which has a tree distance width decomposition. The situation is shown where X_i and X_j are copied. For the vertices u, u', v and w_i the new edges from Step 2 and Step 3 are drawn. In X'_i and X'_j the left two vertices indicate the vertices from the intersection $X_i \cap X_j$ where the copies of v are contained.

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. We define $S = X_r$. $\widehat{G}$ is defined as follows:

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

From H, the graph $\widehat{H}$ is defined similarly. 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 i to r in T. That 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). Because of this there is a minimal tree distance decomposition of $\widehat{G}$ with root S and 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 nodes 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 together 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 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. □

Corollary 4.2 *Two graphs G, H together with tree decompositions D and D' of width k. Isomorphism testing for G and H such that vertices in bags in D are mapped to the same bags in D' can be solved in* L.

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. □

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. We prove the following theorem.

Theorem 4.3 *Isomorphism testing for two graphs of bounded treewidth when given a tree decomposition for one of them is in* LogCFL*.*

Proof: We describe an algorithm which runs on a nondeterministic 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 a $O(\log n)$ space bounded work-tape. On the input tape we have two graphs G, H of tree width 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 $D_j = (\{X_i, |, i \in I_j\}, T_j = (I_j, F_j), j)$ 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 4.4 *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 nondeterministically 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 [18] for testing that a graph has bounded treewidth. We briefly sketch this algorithm 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 [18] for testing whether a graph has treewidth k. On input a graph G the algorithm guesses nondeterministically 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.

Let G be the connected input graph. P and Q denote vertex sets of size $\leq k+1$ 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 we extend G defining an extra vertex v_0 and connecting it to an arbitrary vertex u that is not a separating set in G (I.e. $G\{u\}$ is connected). We will asume that v_0 has a smaller number than all the original vertices in G. We also consider arbitrarily some other vertex $w \neq u$ in G. The algorithim is started with the initial bag $P = \{v_0, u\}$ and the initial vertex w representant of the unique split componente of P. The initial call is thus DECOMPOSE$(G, v_0, \{v_0, u\}, w)$. P and the sequence of bags Q defined in an accepting nondeterministic computation define a tree decomposition of G (once the vertex v_0 is deleted from them).

Vertex v_0 and edge $\{u_0, v_0\} \in E(G)$ are chosen arbitrarily. We start the algorithm with $(G, v_0, \emptyset, v_0)$.

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 from the vertices in $P \setminus Q$. 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. These separating sets form a tree decomposition of G. If a separating set Q is not chosen appropriately, the algorithm will not be able to find later in the recursion an approriate set of $k+1$ vertices which further decomposes the rest of the graph and rejects.

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

Input: graph G, vertex v_0, separating set P with $|P| \leq k+1$, 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: choose Q of size $\leq k+1$ 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 (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.

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.

We start guessing a root bag $X'_{r'}$ of size $\leq k+1$ for a decomposition of H. With $X'_{r'}$ as root bag we guess the tree decomposition D' of H which corresponds to D and its root r. We also construct 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 we guess an extension of ϕ from X_r onto $X'_{r'}$. The algorithm starts with $a = r$ (and $a' = r'$), i.e. with $(G, H, D, X_r, X'_{r'})$. Then we describe isomorphism classes for $1, \ldots, l$, the children of a in D. First, the children of a can be distinguished because $X_1, \ldots, X_l$ may intersect with X_a differently. Second, we further partition the children within one class according to the number of decomposition respecting isomorphic siblings in that class. This can be done in logspace with cross comparisons of pairs among $(G_1, D_1), \ldots, (G_l, D_l)$, see Corollary 4.2. It suffices to order the isomorphism classes according to the lexicographical subgraph order of the members in the classes. We compare then the children of a with guessed children of a' keeping the following information: For each isomorphism class we check whether there is the same number of isomorphic subtrees of a' in H and whether those intersect with $X'_{a'}$, accordingly. For this we use the lexicographical subgraph order to go through the isomorphic siblings from left to right, just keeping a pointer to the current child on the work tape. For two such children, say s_1 of a and t_1 of a', we check then recursively whether (G_1, D_1) is isomorphic to the corresponding subgraph of t_1 in H, by an extension of ϕ.

When we go into recursion, we push 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 $X'_{a'}$.

In general, we do not keep all the information of ϕ 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 we ran through all children of some node we go one level up in recursion and recompute all the other information which is given implicitly by the subtrees from which we returned. Suppose now, we returned to the bag X_a, we have to do the following:

- Pop from the stack the partial isomorphism ϕ of the bags X_a onto $X'_{a'}$
- Compute the lexicographical next isomorphic sibling. For this we consider the partition into isomorphism classes according to ϕ and the lexicographical subgraph order of Definition 4.4.
- If there is no such sibling then we look for the lexicographical first child of X_a inside the same isomorphism class. From this child of X_a we compute the sibling which is not in the

same isomorphism class and which comes next to the right in the lexicographical subgraph order.

- If there is neither a further sibling in the same isomorphism class nor a non-isomorphic sibling of higher lexicographical order then we have visited all children of X_a and we are ready to further return one level up in recursion.

Algorithm 2 summarizes the above considerations.

Algorithm 2 Treewidth Isomorphism with one tree decomposition

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, if G is isomorphic to H 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** reject
3: Let $1, \ldots, l$ be the children of a in T. Partition the subgraphs corresponding to the subtrees of T rooted at $1, \ldots, l$ into p (decomposition respecting) isomorphism classes $E_1, \ldots, E_p$
4: **for** each class E_j from $j = 1$ to p **do**
5: **for** each subtree $T_i \in E_j$ (in lexicographical subgraph order) **do**
6: guess a bag $X'_{i'}$ in H in increasing lexicographical subgraph order of $H_{i'}$ where $H_{i'}$ is the subgraph of H induced by the vertices in $X'_{i'}$ and by those which are separated from $X'_{r'}$ in $H \setminus X'_{i'}$
7: **if** $X'_{i'}$ is not a correct child bag of $X'_{a'}$ (see Wanke's algorithm) **then** reject.
8: Invoke this algorithm with input $(G_i, H_{i'}, D_i, X_i, X'_{i'})$ recursively and push X_a, $X'_{a'}$ and the partial isomorphism ϕ on the stack
9: After recursion pop these informations from the stack
10: **end for**
11: **end for**
12: **if** the stack is not empty **then** go one level up in recursion
13: accept and halt

In Step 1 we guess an extension of the partial isomorphism ϕ to include a mapping from X_a onto $X'_{a'}$. We know the partial isomorphism of their parent bags because this information can be found on the top of the stack. In Step 3, we have for example the partition $E_1 = \{T_1, \ldots, T_{l_1}\}$, $E_2 = \{T_{l_1+1}, \ldots, T_{l_2}\}$ and so on. The partition can be obtained in logspace by testing decomposition respecting isomorphism of the tree structures $(G_1, D_1), \ldots, (G_l, D_l)$. Two subtrees rooted at X_i and X_j are in the same isomorphism class if and only if there is an automorphism in G which maps X_i onto X_j and fixes their parent X_a setwise. In Steps 6 to 9, we guess the bag $X'_{i'}$ in H which corresponds to X_i and we test recursively whether the corresponding subgraphs G_i and $H_{i'}$ are isomorphic with an extension of the partial isomorphism ϕ. Observe that we 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 choosen in increasing lexicographical order (Line 6) 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 7, we check 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 $X'_{a'} \setminus X'_{i'}$).

To see that the algorithm correctly computes an isomorphism, we make the following observation. A bag X_a is a separating set which defines the connected subgraphs $G_1, \ldots, G_l$. These subgraphs do not contain the root X_r and $V(G_i) \cap V(G_j) \subseteq X_a$ since we have a tree decomposition D. We guess and keep 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 we 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 neighboring bags. 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 nondeterministic 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 [7] 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 4.3 we get:

Corollary 4.5 *The isomorphism problem for bounded treewidth graphs is in* LogCFL.

5 Conclusions and open problems

We have shown that the isomorphism problem for graphs with 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 graph we also give a canonizing logspace algorithm. 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. On the one hand, no LogCFL-hardness result for the isomorphism problem is known, even for the case of general graphs, so maybe the result can be improved. On the other hand we believe that proving a logspace upper bound for the isomorphism problem of bounded treewidth graphs would require to compute tree decompositions within logarithmic space, which is also a long standing open question. Another interesting open question is whether bounded tree width graphs can be canonized in LogCFL.

References

[1] V. Arvind, P. Kurur and T.C. Vijayaraghavan, Bounded color multiplicity graph isomorphism is in the #L hierarchy, in *Proc.20th IEEE CCC* (2005) 13–27.

[2] H.L. Bodlaender, Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees, *J. 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 and A. Koster, Combinatorial optimization of graphs of bounded treewidth, *The Computer Journal* (2007), 631–643.

[5] S. Datta, N. Limaye, P. Nimbhorkar, T. Thierauf and F. Wagner, Planar graph isomorphism is in Logspace, In *Proc. 24th IEEE CCC* (2009), 203–214.

[6] S. Datta, P. Nimbhorkar, T. Thierauf and F. Wagner, Isomorphism of $K_{3,3}$-free and K_5-free graphs is in Logspace, To appear in *Proc. 29th FSTTCS* (2009).

[7] G. Gottlob, N. Leone and F. Scarcello, Computing LOGCFL certificates, In *Theoretical Computer Science* **270** (2002), 761–777.

[8] M. Grohe and O. Verbitsky, Testing graph isomorphism in parallel by playing a game, In *Proc. 33rd ICALP* (2006), 3–14.

[9] B. Jenner, J. Köbler, P. McKenzie and J. Torán, Completeness results for Graph Isomorphism, *Journal of Computer and System Sciences* **66** (2003) 549–566.

[10] J. Köbler and S. Kuhnert, The isomorphism problem of k-trees is complete for Logspace, In *Proc. 34th MFCS* (2009), 537–448.

[11] J. Köbler, U. Schöning and J. Torán, *The Graph Isomorphism problem*, Birkhäuser (1993).

[12] S. Lindell, A Logspace algorithm for tree canonization, In *Proc. 24th ACM STOC* (1992), 400–404.

[13] E. Luks, Isomorphism of graphs of bounded valence can be tested in polynomial time, *Journal of Computer and System Sciences* **25** (1982), 42–65.

[14] E. Luks, Parallel algorithms for permutation groups and graph isomorphism. In *Proc. 27th IEEE FOCS* (1986), 292–302.

[15] G. Miller, Isomorphism testing for graphs of bounded genus, In *Proc.12th ACM STOC*, (1980), 225–235.

[16] O. Reingold, Undirected connectivity in logspace In *Journ. of ACM*, **55** (4) (2008).

[17] I. Sudborrough, Time and tape bounded auxiliary pushdown automata. *Mathematical Foundations of Computer Science* (1977), 493–503.

[18] E. Wanke, Bounded tree-width and LOGCFL *Journal of Algorithms* **16** (1994), 470–491.

[19] K. Yamazaki, H.L. Bodlaender, B. de Fluiter and D.M. Thilikos, Isomorphism for Graphs of Bounded Distance Width, *Algorithmica* **24** (1999), 105–127.