Electronic Colloquium on Computational Complexity, Report No. 32 (2011)

Graphs of Bounded Treewidth can be Canonized in AC^1

Fabian Wagner
Inst. für Theoretische Informatik
Universität Ulm
89069 Ulm, Germany
fabian.wagner@uni-ulm.de

March 11, 2011

Abstract

In recent results the complexity of isomorphism testing on graphs of bounded treewidth is improved to TC^1 [GV06] and further to LogCFL [DTW10]. The computation of canonical forms or a canonical labeling provides more information than isomorphism testing. Whether canonization is in NC or even TC^1 was stated as an open question in [Köb06]. Köbler and Verbitsky [KV08] give a TC^2 canonical labeling algorithm. We show that a canonical labeling can be computed in AC^1. This is based on several ideas, e.g. that approximate tree decompositions of logarithmic depth can be obtained in logspace [EJT10a], and techniques of Lindells tree canonization algorithm [Lin92]. We define recursively what we call a *minimal description* which gives with respect to some parameters in a logarithmic number of levels a canonical invariant together with an arrangement of all vertices. From this we compute a canonical labeling.

1 Introduction

The *graph isomorphism problem* (GI) consists in deciding whether two given graphs are isomorphic, i.e. whether there is a permutation of all vertices that keeps the edge relation unchanged. GI is a well-studied problem in theoretical computer science 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.

By studying GI, graph canonization receives a great attention because of its strong connection to GI. Thereby, it provides more information than isomorphism testing. Let $\mathcal{G}$ be a class of graphs. A *complete invariant* is a function $f : \mathcal{G} \to \{0,1\}^*$ where $f(G) = f(H)$ if and only if both graphs $G, H \in \mathcal{G}$ are isomorphic. With this function f, the isomorphism classes can be distinguished. A *canonical form* is a function $f : \mathcal{G} \to \mathcal{G}$ where $f(G)$ is a representative of the (equivalence) class of isomorphic graphs to G in $\mathcal{G}$. For example, if $f(G)$ is defined to be the lexicographically least graph in $\mathcal{G}$ isomorphic to G, then the computation of f is in general NP-hard (c.f. [BL83, Luk93]). Clearly, graph isomorphism or the computation of a complete invariant is polynomial time reducible to graph canonization, the reverse direction is open. A *canonical labeling* assigns to each graph G in $\mathcal{G}$ a map σ that is an automorphism from G to the representative G^σ of the class of isomorphic graphs to G in $\mathcal{G}$. Note, that function f with $f(G) = G^\sigma$ is a canonical form. Thus, canonical labeling is the strongest among the above defined notions.

ISSN 1433-8092

Robertson and Seymour [RS86] introduced the concept of bounded treewidth graphs, also known as *partial k-trees*. 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 for example [Bod98, BK08]. Thereby, many problems that are NP-hard in general become efficiently solvable when restricted to graphs of bounded treewidth. Bodlaender showed in [Bod90] that GI can be solved in polynomial time when restricted to this class of graphs. Grohe and Verbitsky [GV06] give a TC^1 upper bound, they use the Weisfeiler-Lehman algorithm that can be implemented as a logspace uniform family of TC^1-circuits. This bound was improved recently to LogCFL in [DTW10]. Recall the relationships among some of the complexity classes which are used in this section:
$\mathsf{L} \subseteq \mathsf{LogCFL} \subseteq \mathsf{AC}^1 \subseteq \mathsf{TC}^1 \subseteq \mathsf{TC}^2 \subseteq \mathsf{NC}$. We refer to Section 2 for more details.

For the canonization problems the situation is different. Köbler [Köb06] states an open problem, namely whether graphs of bounded treewidth admit an NC or even TC^1 canonization. Köbler and Verbitsky [KV08] give an NC (in fact TC^2) canonical labeling algorithm. They prove a theorem showing that, for some special classes of graphs from a complete invariant that is computable in TC^k, a canonical labeling is computable in $\mathsf{TC}^{\mathsf{k+1}}$. With this theorem and with the fact that for bounded treewidth graphs a complete invariant can be computed in TC^1 [GV06], the result follows.

For subclasses of bounded treewidth graphs, i.e. for trees [Lin92] and partial 2-trees [ADK08] logspace algorithms are known for the canonization problems. Completeness for logspace is shown in [JKMT03]. Also for planar graphs [DLN$^+$09] and the more general classes of $K_{3,3}$-minor free and of K_5-minor free graphs [DNTW09], the canonization problem is in logspace. For full k-trees the isomorphism problem is in logspace [KK09].

Das, Torán, and Wagner [DTW10] give an isomorphism algorithm that runs in LogCFL. The algorithm uses the fact that an arbitrary tree decomposition for one of the input graphs G can be computed in LogCFL [Wan94] and by a recent result in L [EJT10a]. They set up the tree decomposition for the other graph H in parallel to the isomorphism test. For a graph there could be an exponential number of tree decompositions. That is one reason, why their algorithm cannot be generalized to a canonization algorithm.

The logspace-version of Courcelle's Theorem in [EJT10a] puts many problems on bounded treewidth graphs which can be formulated in *monadic second-order logic* in L, e.g. *3-colorability*, *Hamiltonicity*, or reachability problems. But it is not known how to formulate isomorphism testing or canonization problems in this logic.

The research on the difficulty of computing tree decompositions of width at most k has a long history, there is a linear time bound of Bodlaender's Theorem [Bod96] improving previous results [ACP87, Ree92]. Also the parallel time complexity has been reduced to $O(\log n)$ in a line of papers [CH88, Bod89, Lag90, BH98]. The space complexity has been reduced to LogCFL in [Wan94] and recently to L by Elberfeld, Jakoby, and Tantau [EJT10a], where a logspace-version of Bodlaender's Theorem is shown: a tree decomposition for graphs of treewidth at most k can be computed in logspace. The following lemmas are important, to guarantee the treewidth bound of the input graph and a logarithmic depth bound.

Lemma 1.1 ([EJT10a]) *For every $k \geq 1$ the language* Tree-Width-k*, which contains exactly the graphs of treewidth at most k, is* L*-complete under first-order reductions.*

Lemma 1.2 ([EJT10a]) *Let G be a graph with n vertices and treewidth at most k. There is a tree decomposition that has width $4k+3$ and depth at most $c \log_2 n$, where c is a constant depending only on k.*

The idea is, that for a split component of size m a child bag is defined in such a way that the split components of this child bag have size at most $m/2 + c$ with c a constant. This tree decomposition is called *approximate*.

Our contribution. The *existence* of a tree decomposition of logarithmic depth is the basis for our algorithm, to compute a canonical labeling in AC^1. The following observations are the key ingredients to guarantee this tight depth bound.

- Each bag is a separator, i.e. for each split component we have to find a child bag. Two child bags are connected via their parent bag only. Hence, the canonization process can be done for each child individually and in parallel.
- Many tasks are computed in preprocessing steps, e.g. whether the graph has treewidth k, all possible bags of size $4k + 3$, for each bag its possible children and the parent (with respect to a fixed root), or the split components for each possible bag and their size.
- The total number of possible bags of size $4k + 3$ in arbitrary tree decompositions is bounded by n^{4k+3}, where n is the size of the input graph. For each bag we define locally a circuit with unbounded fan-in gates. The total size of the circuit is also polynomial.
- We compute for each possible bag what we call a *minimal description*. It depends on some parameters, it consists of a unique description of the bag itself and a minimal description for the children, it is defined recursively.
- We limit the number of recursion levels of the minimal description to $O(\log n)$. Here we use Lemma 1.2, namely the *existence* of tree decompositions of logarithmic depth.
- A circuit of constant depth selects locally valid child bags to get the smallest minimal description. The difficult task is here sorting minimal descriptions. For this we use ideas from Lindells tree canonization algorithm [Lin92]. Instead of a logarithmic space bound we have here logarithmic depth circuits.
- The length of the minimal description for a balanced subtree depends on its depth only. In the minimal description, a bag is described by its adjacency matrix in at most $(4k+3)^2$ bits, if necessary we fill up to this total length repeating an extra symbol. We use the fact that a tree can be made *balanced* and also *binary* where its depth increases just by a constant factor (c.f. [EJT10a]).

The minimal description depends on the selection of the root bag, too. We run through all bags as roots and some further parameters in parallel. The minimal description so far is a *canonical invariant* for the input graph G.

A *canonizing function* is obtained then as follows. While computing the minimal description we bring the vertices in a unique order. Then we use a simple logspace-computable procedure (see e.g. [DLN08]) where vertices are renamed according to this order. This renaming is a *canonical labeling*. We get the main Theorem.

Theorem 1.3 *For every k, there is an AC^1-computable function that computes a canonical labeling for graphs of treewidth at most k.*

2 Preliminaries

We use the notion *interval* $[p, q]$ for the set $\{p, p+1, \dots, q\}$. The *symmetric group* $Sym(V)$ is the set of all permutations of the elements in V.

A *word* or *string* is a tuple of symbols of an *alphabet* Σ. For example, we use in some places $\Sigma = \{0, 1, 2\}$. The *concatenation* of two words $W = w_1 w_2 \dots w_k, W' = w'_1 w'_2 \dots w'_k$ is denoted $WW' = w_1 w_2 \dots w_k w'_1 w'_2 \dots w'_k$.

Graphs. A *graph* G is a pair (V, E) with a set of *vertices* V (or $V(G)$) and *edges* $E \subseteq V \times V$ (or $E(G)$). If not stated otherwise, we consider simple graphs, i.e. without loops, directed edges or multi-edges. $G[X]$ is a subgraph of G *induced* on vertex set X, i.e. $G[X]$ has vertices X and edges $E(G[X]) = (X \times X) \cap E(G)$. Let $X \subseteq V$ then we write in short $G \setminus X = G[V(G) \setminus X]$. Let H be a subgraph of G, then $G \setminus H = G[V(G) \setminus V(H)]$.

A graph G is *connected* if there is a path between every pair of vertices in G. Let $U \subseteq V$ be a set of vertices. Let C be a connected component in $G \setminus U$ and let U' be those vertices from U connected to $V(C)$ in G. The induced subgraph of G on the set of vertices $V(C) \cup U'$ is a *split component* of U in G. We call U' the *minimal separating set of C in U*.

A *tree* is a connected graph that is free of cycles. Vertices in trees are also called *nodes*. A *root* of a tree is one designated node. A neighbor of a node is called *parent* if it is closer to the root and it is called *child* otherwise. A *leaf* of a tree has no children. In a *binary tree*, every node has at most two children. A binary tree with root is *balanced*, if for every node the number of nodes in its left and right subtrees differs by at most one. A binary tree is *perfect*, if it is balanced and every leaf is at the same depth. The *depth* of a tree is the longest distance from the root to a leaf. Let T and T' be trees rooted at r and r' and consider edges to be directed from roots to leafs. An *embedding of T into T'* is an injective mapping $\iota : V(T) \to V(T')$ where $\iota(r) = r'$ and for every pair $(a, b) \in V(T)$ there is a directed path from a to b iff there is a directed path from $\iota(a)$ to $\iota(b)$.

An *isomorphism* is a mapping ϕ of the vertices of one graph G onto the vertices of another graph H (we also write $G \cong H$) that preserves the edge relations, i.e. $\{u, v\} \in E(G) \Leftrightarrow \{\phi(u), \phi(v)\} \in E(H)$. Let $\mathcal{G}$ be a class of graphs. A *complete invariant* is a function $f : \mathcal{G} \to \{0, 1\}^*$ where $f(G) = f(H)$ iff $G \cong H$. A *canonical form* is a complete invariant with $f : \mathcal{G} \to \mathcal{G}$ where $f(G) \cong G$, we call $f(G)$ *canon* of G. A *canonical labeling* is a function $f : \mathcal{G} \to Sym(V(G))$ which assigns to a graph G an automorphism σ onto the canon, i.e. the function g with $g(G) = G^\sigma$ is a canonical form.

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

Complexity. A *circuit* C_n is a finite directed acyclic graph with vertices associated to n input variables or gates (e.g. Boolean functions from a given base). For an assignment of the variables we associate a Boolean value to every gate in the circuit. The value of an input is the one given by the assignment to the corresponding variable. For an internal gate, the value

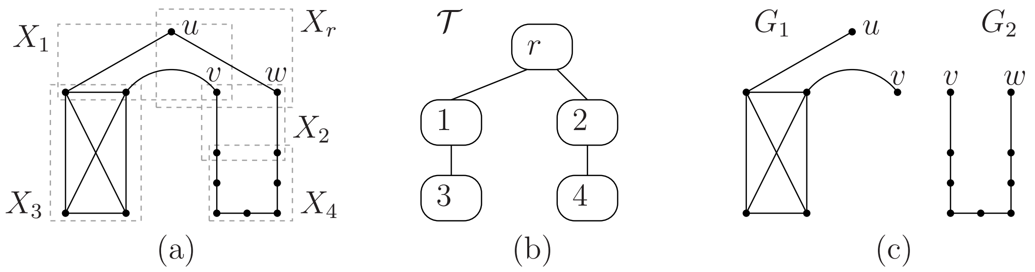

Figure 1: (a) A graph G where dashed lines indicate bags $X_r, X_1, \ldots, X_4$.
(b) The set of bags form a tree decomposition $\mathcal{T}$ of G. Let r be the root.
(c) The split components G_1 and G_2 of X_r are shown. The sets $\{u, v\}, \{v, w\} \subseteq X_r$ are the minimal separating sets for G_1 and G_2, respectively.

is the one computed by the corresponding function, from the values of the gate inputs. The circuit computes a function $f(x_1, \ldots, x_n)$. This is the value of the designated *output gate*. The indegree of the vertices is called *fanin*. A *circuit family* $\{C_1, C_2, \ldots\}$ is a collection of circuits where C_n has n inputs. We consider here DLOGTIME-*uniform* circuit families, i.e. a deterministic Turing machine on input of 1^n, integer i, and bit b, with $O(\log n)$ time bound accepts iff the i-th bit of the description of C_n is b.

We give an overview of the complexity classes mentioned in this work.

L (also denoted *logspace*) 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 logspace to a context free language. Problems in LogCFL can be computed also by uniform families of polynomial size and logarithmic depth circuits over bounded fan-in *and*-gates and unbounded fan-in *or*-gates.

The class NC^i contains the problems computable by uniform families of polynomial size and $O(\log^i n)$ depth circuits over bounded fan-in *and*-gates and bounded fan-in *or*-gates. Note, that $\mathsf{NC} = \bigcup_i \mathsf{NC}^i$. AC^i is defined as NC^i but with unbounded fan-in gates. The class TC^i contains the problems computable by uniform families of polynomial size and $O(\log^i n)$ depth circuits with threshold gates, i.e. gates that evaluate to 1 if at least half of their inputs are 1. The known relationships among these classes are:

$$\mathsf{AC}^0 \subset \mathsf{TC}^0 \subseteq \mathsf{NC}^1 \subseteq \mathsf{L} \subseteq \mathsf{LogCFL} \subseteq \mathsf{AC}^1 \subseteq \mathsf{TC}^1 \subseteq \mathsf{NC}^2 \subseteq \mathsf{AC}^2 \subseteq \mathsf{TC}^2 \subseteq \cdots \subseteq \mathsf{NC} \subseteq \mathsf{P} \subseteq \mathsf{NP}.$$

3 Canonization of Graphs of Bounded Treewidth.

To prove Theorem 1.3, we construct a circuit where we have some preprocessing steps, e.g. for the split components of bags and their size. Then in $O(\log n)$ steps, we compute for each bag X what we call a *good minimal description*, i.e. a unique tree decomposition that depends on some parameters. After the first level, we have good minimal descriptions for single bags, after the second level for bags which have leaf bags as children, and so on. After $O(\log n)$ levels, we have good minimal descriptions for tree decompositions of logarithmic depth, and by Lemma 1.2 a tree decomposition for the whole graph, if it exists. At each level, we select the smallest good minimal description. Once we find a good minimal description at some level i, then this remains unchanged in all levels $\geq i$. First, we describe some tools and then the construction of the minimal description.

3.1 Pre-ordering for Canonization and Valid Child Bags.

Canonical Representation of Bags. Comparing adjacency matrices lexicographically is a natural way to test isomorphism. In general, this is an NP-complete problem, but when comparing bags this can be done with constant effort. We define $adj(G[X], \sigma)$ to be the adjacency matrix of the induced subgraph where the vertices of X are arranged in a fixed order given by $\sigma \in Sym(X)$. We compare adjacency matrices bitwise line by line. For example, in Figure 1 we have $adj\,(G[X_r], \left(\begin{smallmatrix} u & v & w \\ u & v & w \end{smallmatrix}\right)) = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}$ and $adj\,(G[X_r], \left(\begin{smallmatrix} u & v & w \\ u & w & v \end{smallmatrix}\right)) = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$.

Child bags and their order. In the following two definitions we bring together the logarithmic depth bound of Lemma 1.2 and what we need for canonization. First, we define an order on split components that is just partially canonical, but which is the basis for our canonization algorithm. Second, we define what we call *valid child bags*, we use them to construct approximate tree decompositions. Note, although we consider graphs of treewidth at most k, we get approximate tree decompositions that allow bags of size at most $4k + 3$.

Definition 3.1 *Let G be a graph of treewidth at most k and X a bag of size $\leq 4k + 3$. Let $G_1, \ldots, G_m$ be its split components. Let $\sigma \in Sym(X)$ be an ordering on the vertices in X. Let $S_1, \ldots, S_l \subseteq X$ be a complete list of minimal separating sets of $G_1, \ldots, G_m$ in X. We define an* order *on $G_1, \ldots, G_m$ with respect to σ using two criteria lexicographically:*

1. *primarily according to σ, which induces a lexicographical order on $S_1, \ldots, S_l$, and*

2. *among them which are equal, we reorder them according to the sizes of their associated split components.*

Note, l is a constant, because $l \leq m$ and in an approximate tree decomposition $l \leq |\mathcal{P}(X)|$ (i.e. $l \leq (4k + 3)! \leq 2^{(4k+3)\log(4k+3)}$), that is the number of all possible subsets of X.

The primary order of split components is deduced from σ. We give an example: let $\sigma = \left(\begin{smallmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 1 & 4 \end{smallmatrix}\right)$ and $S_1 = \{1, 2, 4\}$ and $S_2 = \{3, 4\}$. When sorted according to σ (i.e. $2 < 3 < 1 < 4$), then we have $(2, 1, 4) < (3, 4)$ lexicographically and therefore S_1 comes before S_2. The second criterion brings together split components of equal size, this is useful for canonization and also for the complexity analysis, also see the tree canonization algorithm of Lindell [Lin92].

There are split components which are equal according to Definition 3.1. To get a default order, they can be sorted according to the label of the minimal vertex in the split components except X. To rearrange them is part of the canonization algorithm.

Definition 3.2 *Consider G, X, $G_1, \ldots, G_m$ being sorted and σ as in Definition 3.1. Let $1 \leq p \leq q \leq m$, we consider now $G_p, \ldots, G_q$. Let $S_1, \ldots, S_{l'}$ be all the minimal separating sets of $G_p, \ldots, G_q$. We define three types of* valid child bags of X with respect to split components $G_p, \ldots, G_q$ *as follows:*

(a) Take minimal separating sets of bags: If $l' > 1$ and S_j is the minimal separating set for G_p then the set S_j is a valid child bag.

(b) Take a subset of X and add vertices from G_i: If $l' = 1$ then for $V_i \subseteq X \cup V(G_i)$ with $i \in \{p, \ldots, q\}$ with $|X \cup V_i| \leq 4k + 3$ and $\emptyset \subset (X \cap V_i) \subset X$, then the set V_i is a valid child bag.

(c) The set X is a valid child bag.

3.2 Minimal Description for Graphs of Bounded Treewidth.

We define a minimal description for graphs. This is based on approximate tree decompositions which are balanced and depth bounded. The minimal description is computed with respect to some parameters.

By Lemma 1.2 we know that there exist tree decompositions of logarithmic depth, we show that we can guarantee this depth restriction, we use a so called *depth parameter*. We consider perfect trees only.

There is a task where split components are partitioned into classes, e.g. if they have the same size. By Definition 3.1 split components are ordered and hence, split components in a class can be addressed by an *interval*, e.g. $[p, q]$ for $G_p, \ldots, G_q$. Another parameter is a permutation σ, it describes a unique order of the vertices in the current root bag X.

We consider first some cases concerning the depth parameter. That is, when the total depth is exceeded and we return *no-canon*, or when a minimal description is already computed in the previous step and we just take this.
In the good case the minimal description is a word $C_0C'_0C_1C_2 \in \{0, 1, 2\}^*$.

Here, C_0 and C'_0 contain information of the root, C_1 and C_2 contain the minimal description of the children. To specify which of two minimal descriptions is the smaller one, we define an order $\prec$ on them.

Definition 3.3 *We define an* order $\prec$ *on minimal descriptions* $C = C_0C'_0C_1C_2$ *and* $D = D_0D'_0D_1D_2$. *We define* $C \prec D$ *if*

- $C_0 < D_0$ *where we compare adjacency matrices line by line and bit by bit, or*
- $C_0 = D_0$ *but* $C'_0 < D'_0$ *where we compare not vertex labels according to* σ *but their positions in the parent of* X *(example: for* $C'_0 = (b, c, d, e)$ *and the ordered vertices of* $\text{Parent}(X) = (a, b, c, d)$ *we have* $C'_0 = (2, 3, 4)$*), or*
- $C_0 = D_0$ *and* $C'_0 = D'_0$ *but* $C_1 \prec D_1$*, recursively, or*
- $C_0 = D_0$ *and* $C'_0 = D'_0$ *and* $C_1 = D_1$ *but* $C_2 \prec D_2$*, recursively.*

The construction. Let G be a graph of size n, treewidth at most k, a root bag X of size $\leq 4k + 3$, a permutation $\sigma \in Sym(X)$, an interval $[p, q]$ (with $1 \leq p \leq q \leq m$ and $G_1, \ldots, G_m$ the split components in $G \setminus X$ arranged according to Definition 3.1) and depth parameter $d \in \mathbb{Z}$. A *minimal description* $C(G, X, \sigma, (p, q), d)$ is *no-canon* or a word in $\{0, 1, 2\}^*$ defined as follows.

If $d < 0$ then $C(G, X, \sigma, (p, q), d) =$ *no-canon*, this tree decomposition exceeds the maximum depth. We call a minimal description *good* if it is different to *no-canon.*

If $d > 0$ and $C(G, X, \sigma, (p, q), d - 1) \neq$ *no-canon* then we just copy it from the previous level, that is $C(G, X, \sigma, (p, q), d) = C(G, X, \sigma, (p, q), d - 1)$.

If $d = 0$ or $d > 0$ and $C(G, X, \sigma, (p, q), d - 1) =$ *no-canon* then $C(G, X, \sigma, (p, q), d) = C_0C'_0C_1C_2$ or *no-canon* defined as follows:

(1) C_0 *encodes the adjacency matrix of the root bag* X.

$$C_0 = adj(G[X], \sigma)\{2\}^{(4k+3)^2 - |X|^2},$$

the adjacency matrix of the root, filled up with symbol 2, an extra symbol to get the total length $(4k + 3)^2$.

(2) *C'_0 encodes the vertices of X ordered by σ.* This part is important for canonization, when bringing all vertices in a unique order.

$$C'_0 = v_1 \dots v_i \{2\}^{\lceil \log(n) \rceil \cdot ((4k+3)-i)}$$

with $i = |X|$. The vertices are ordered according to σ, (i.e. $\sigma(v_j) = j$ for all $v_j \in X$). We assume, that the description of every vertex has a fixed length of exactly $\lceil \log n \rceil$ bits.

(3) *Definition of C_1 and C_2 in further subcases.* We only consider the split components with their index inside the interval $[p, q]$, these are $(q - p + 1)$ many. Let $S_1, \dots, S_l \subseteq X$ be all the minimal separating sets (in lexicographical increasing order according to σ) for split components $G_p, \dots, G_q$ which are ordered according to Definition 3.1. We partition the split components corresponding to the members in S_1: let $\Theta_1, \dots, \Theta_t$ be the classes where the corresponding split components have equal size. These *size classes* are arranged in increasing order of the sizes of the corresponding split components. To define C_1 and C_2, we consider the following cases.

(i) $p < q$ **and** $l > 1$**.** That is, we have many such minimal split components, we separate them from the others and recursively canonize them in C_1 and the others in C_2. For both, X is the child according to type (c) in Definition 3.2.

Let $G_p, \dots, G_{q_1}$ be the set of split components separated by S_1. Let ψ be obtained from σ after removing the vertices from $X \setminus S_1$, we define

$$C_1 = C(G, X, \sigma, (p, q_1), d-1) = C(G', S_1, \psi, (1, q_1 - p + 1), d-1).$$

where $G' = G[V(G_p) \cup \dots \cup V(G_{q_1})]$. Note, since the split components are ordered according to Definition 3.1, we just take a subinterval $[p, q_1]$ of $[p, q]$. If $l > 2$ then

$$C_2 = C(G, X, \sigma, (q_1 + 1, q), d-1).$$

If $l = 2$ then we define ψ' accordingly as ψ before with respect to S_2,

$$C_2 = C(G, X, \sigma, (q_1 + 1, q), d-1) = C(G', S_2, \psi', (1, q - q_1), d-1)$$

where $G' = G[V(G_{q_1+1}) \cup \dots \cup V(G_q)]$.

(ii) $p < q$ **and** $l = 1$ **and** $t > 1$**.** That is, we have a single minimum separating set S_1 but many size classes. We try to find how to partition the size classes which result in the smallest minimal description, i.e. such that we have two sets $\Theta_1, \dots, \Theta_i$ and $\Theta_{i+1}, \dots, \Theta_t$.

For each $i \in \{1, \dots, t-1\}$ let $G_p, \dots, G_{q_i}$ be the set of split components of size classes $\Theta_1, \dots, \Theta_i$. We define

$$C_{1,i} = C(G, X, \sigma, (p, q_i), d-1) \text{ and } C_{2,i} = C(G, X, \sigma, (q_i + 1, q), d-1).$$

Let (C_{min}, C'_{min}) be the lexicographically smallest pair according to $\prec$ in the set $\{(C_{1,i}, C_{2,i}), (C_{2,i}, C_{1,i}) \mid 1 \leq i \leq t-1\}$. We define $C_1 = C_{min}$ and $C_2 = C'_{min}$.

(iii) $p < q$ **and** $l = 1$ **and** $t = 1$**.** That is, we have many children from one separating set and one size class. We canonize all the children $G_p, \dots, G_q$ individually and sort their minimal descriptions in ascending order according to $\prec$. Then, we do the same rearrangements with the split components $G_p, \dots, G_q$. In C_1, we canonize the first half of them and in C_2 the second half. Note, the sorting of $q-p+1$ children is expensive, hence we reduce the depth parameter logarithmically in the length of this interval. X is the child in both cases according to type (c) in Definition 3.2.

Let $a = \lceil \log_c(q-p+1) \rceil$ where $c = 3/2$. For each $i \in \{p, \dots, q\}$ we define $C'_i = C(G, X, \sigma, (i,i), d-a)$. If one of the $C'_i =$ *no-canon* then $C_1 =$ *no-canon*. Rearrange $C'_p, \dots, C'_q$ in lexicographical increasing order according to $\prec$. Rearrange $G_p, \dots, G_q$ according to the new order of $C'_p, \dots, C'_q$. Let $i = \lceil (q-p)/2 \rceil$. We define

$$C_1 = C(G, X, \sigma, (p, p+i), d-1) \quad \text{and} \quad C_2 = C(G, X, \sigma, (p+i+1, q), d-1).$$

(iv) $p = q$**.** That is, we consider one single child. We consider all valid child bags of type (b) in Definition 3.2. The one which gives the smallest minimal description is selected for C_1, whereas C_2 is filled up with default symbols.

For each set of vertices $V_i \subseteq X \cup G_p$ that is a *valid child bag* as in Definition 3.2 with $V_i \cap G_p \neq \emptyset$, and each permutation $\psi_{i,j} \in \mathit{Sym}(V_i)$ which is obtained from σ when removing the vertices in $X \setminus V_i$ and adding to the right the vertices in $V_i \setminus X$ in an arbitrary order, we define

$$C_{1,(i,j)} = C(G_p \setminus (X \setminus V_i), V_i, \psi_{i,j}, (1, m'), d-1)$$

and $(i,j) \in I$, where m' is the number of split components in $G_p \setminus (X \cup V_i)$, i.e. the subgraph $G_p \setminus (X \setminus V_i)$ when removing V_i. We define C_1 to be the minimum of $\bigcup_{(i,j)\in I} C_{1,(i,j)}$. $C_2 = \{2\}^{f(d-1)}$ is a *default description* for a complete binary subtree. We define $f : d \mapsto (|C_0| + |C'_0|) \cdot (2^d - 1)$, i.e.

$$f(d-1) = \left[(4k+3)^2 + \lceil \log n \rceil \cdot (4k+3)\right] \cdot (2^{(d-1)} - 1).$$

(v) $m = 0$**.** That is, X is not a separating set in G, i.e. a leaf node in a tree decomposition. If no valid child bag exists, then we define $C_1 = C_2 = \{2\}^{f(d-1)}$ with f as in case (iv). That is, both have a default description. Then, a good minimal description is returned.

In general, there is one exception, namely if one of C_1 or C_2 returns *no-canon*, then the minimal description is *no-canon*.

If $C_2 \prec C_1$, then swap them.
If C_1 or C_2 is *no-canon* then $C(G, X, \sigma, (p,q), d) =$ *no-canon*.

The depth of the tree decomposition. We mention some points that have an influence on the depth of the tree decomposition. Note, the construction for $m > 2$ uses some ideas from the proof of a Theorem in a full version of [EJT10a], where a binary tree is computed by introducing *white nodes*. The main difference is here, that the whole tree is not given explicitly. Our construction prefers a balanced and binary tree-structure. The depth analysis is inspired from the following more restricted version of Theorem 3.14 in [EJT10b].

Lemma 3.4 (c.f. [EJT10b]) *For every tree T with n vertices and height h, there is a binary tree T' of height at most $O(h + \log n)$ such that T can be embedded in T'.*

By Lemma 1.2 the depth of a tree decomposition that has width $4k+3$ is at most $c \log_2 n$ for a constant c that depends on k only. Hence, when starting with $d = c' \log_2 n$ (for a constant c') we get a minimal description for $C(G, X, \sigma, (1, m), d)$.

We observe, that our definition guarantees the $O(\log n)$ depth bound. After $O(1)$ steps, the size of the split components is divided at least by 2. In Case (i) we partition the subtrees that belong to the smallest minimal separating set S_1 of X. We can do this, since the number of separating sets is a constant.
In Case (ii) we run through all possibilities to split the size classes $\Theta_1, \ldots, \Theta_t$ into two sets $\Theta_1, \ldots, \Theta_i$ and $\Theta_{i+1}, \ldots, \Theta_t$. For example, in two steps we can isolate a size class Θ_i where all the subtrees have together more than half the total size of the subgraph rooted at X: first, split off those to the left and second, those to the right of Θ_i.
In Case (iii) we sort the split components and partition them such that in C_1 at most one more split component is canonized than in C_2. This can only happen when there are more than two split components considered currently. Therefore the size of C_1 is at most $2/3$ the size of $C_0 C_0' C_1 C_2$ and this is the reason why $c = 3/2$ in $a = \lceil \log_c q - p + 1 \rceil$. Later in the complexity analysis part (see Lemma 3.9) we will show that with an inductive argument, split components of size n/i can be canonized by a sub-circuit of depth $c \log(n/i) = c \log n - c \log i$, with c a constant. Hence, to encounter case (iii) recursively is no problem.
In Case (iv) we have a single split component. Hence, we get the following.

Lemma 3.5 *There is a constant c which depends on k only such that for all graphs G of treewidth at most k, there is a root bag X with m split components in $G \setminus X$, permutation $\sigma \in Sym(X)$ and depth parameter $d = c \log_2 n$, such that the minimal description $C(G, X, \sigma, (1, m), d)$ is* good.

The next is to show that the minimal description is unique up to isomorphism.

Lemma 3.6 *For a graph G, a constant c, and two bags X, X' with permutations $\sigma \in Sym(X)$ and $\sigma' \in Sym(X')$, with m split components in $G \setminus X$ (and $G \setminus X'$), and depth parameters $d = d' = c \log n$ (for a constant c) it holds that $C(G, X, \sigma, (1, m), d) = C(G, X', \sigma', (1, m), d')$ if and only if there is an automorphism ϕ of G which maps X onto X' via $\sigma(\sigma')^{-1}$.*

Proof: The definition of tree decompositions is based on the edge relation of the graph. An automorphism moves the vertices while keeping the edge relations unchanged. Hence, if G has a tree decomposition $\mathcal{T}$ with bags $X_1, \ldots, X_n$ and ϕ is an automorphism then there is a tree decomposition with bags $\phi(X_1), \ldots, \phi(X_n)$. We partition the children according to a partial isomorphism σ and the sizes of the subtrees into classes. We sort the minimal descriptions of all children that fall into the same class, lexicographically. For finding a non-trivial child bag, we run through all possible vertex sets of size at most $4k+3$ and all arrangements of the new vertices. Hence, the case analysis does not exclude tree decompositions.

Now to the other direction. A bag is uniquely specified by its adjacency matrix and σ, i.e. how the vertices are arranged. From a bag to its child bag Y, we update the permutation σ, old vertices are removed and new vertices are added to the right to obtain ψ. Hence, we get a unique minimal description also for $G[X \cup Y]$. Because we record the vertex labels, we can recover the entire edge relation from this description. By an induction argument this generalizes to the whole tree decomposition. $\square$

To obtain a canonical invariant for graphs of treewidth at most k, run through all sets of at most $4k+3$ vertices as initial root bag X and all permutations $\sigma \in Sym(X)$. According to Lemma 3.5 there exist good minimal descriptions. We ignore the ones with *no-canon* and select the smallest of all these good minimal descriptions. Thereby, recursively in the parts C_0', we relabel the vertices according to their first occurrence in the good minimal description.

Theorem 3.7 *The smallest minimal description of all bags X and permutations σ is a canonical invariant for graphs of treewidth at most k.*

3.3 Complexity Analysis.

We prove now that graphs of bounded treewidth can be canonized in AC^1. We construct a circuit which consists of preprocessing steps and a main part, where in $O(\log n)$ levels a minimal description of the input graph is computed.
Valid child bags. We consider Definition 3.2. We show, that valid child bags can be computed in logspace. For an AC^1-circuit, in a preprocessing step we compute in parallel for each possible bag which are its split components and its valid child bags.

Lemma 3.8 *On input of a graph G a bag X with a child bag Y, and an interval $[p,q]$, there is a logspace-computable function that computes whether Y is a valid child bag of X.*

Proof: The main task is the following. Compute, whether there is a split component G_i (with $p \leq i \leq q$) such that $Y \subseteq X \cup V(G_i)$.

Computing split components and containment in split components can be done with reachability tests. Reingold proved, that reachability testing in undirected graphs is in logspace [Rei08]. □

Computing the minimum and sorting. Computing the minimum and sorting. We discuss how to compute the minimal description by an AC^1-circuit. By the recursive construction of minimal descriptions in Section 3.2, the depth of the circuit corresponds to the depth of the tree decompositions. The next lemma is essential in the proof of Theorem 1.3.

Lemma 3.9 *Let G be a graph of treewidth at most k, X a root bag, $\sigma \in Sym(X)$ a permutation, $[p,q]$ an interval, d a depth parameter, and suppose we have given minimal descriptions for all valid child bags and all split components $G_p, \ldots, G_q$ for all depth parameters $\leq d-1$.*

- *(a) For Case (iii) in the minimal description on Page 9 there is a (depth i)-bounded AC^1-computable function that computes the smallest minimal descriptions with depth parameter $d-i$ for each of the i equal sized split components, and arranges them in lexicographical increasing order.*
- *(b) For all the other cases, there is an AC^0-computable function that computes the smallest minimal description with depth parameter $d-1$ for each split component.*

On input of the minimal descriptions from (a) and (b), there is an AC^0-computable function that computes the minimal description $C(G, X, \sigma, (p,q), d)$.

Proof: We begin with part (b). In the canonical Description on Page 9 in cases (i), (ii) and (iii) we have to compute the smallest of many minimal descriptions of children. In case (iv) one is a *default description* and sorting is trivial there. To handle case (iv) and case (v), we compute f in a preprocessing step.

We consider at most n^{4k+3} children (as an upper bound for the number of possible valid child bags). We compute the smallest of n^{4k+3} minimal descriptions according to the order $\prec$ with depth parameter $\leq d-1$. Each bit of these minimal descriptions is given in the input. Hence, the main task is to compute the minimum of strings of length at most polynomial in n.

Claim 1 *There is an* AC^0*-computable function that computes on input of strings* $S_1, \ldots, S_N \in \{0,1,2\}^*$ *of length polynomial in* N *the lexicographical smallest string.*

Proof: For two strings this task is first order definable (c.f. [Imm99]) and hence in AC^0. Given a 2-ary relation $<$ and 3-ary relation R with $(a,i,x) \in R$ if the i-th symbol of string S_a is $x \in \{0,1,2\}$:

$$LESS(a,b) \equiv \exists i[(\exists x \exists y\ x < y \wedge R(a,i,x) \wedge R(b,i,y)) \wedge (\forall h \exists x\ (h < i \rightarrow R(a,h,x) \wedge R(b,h,x)))]$$

Accordingly, we can ask whether one string is less or equal:

$$LESS\text{-}OR\text{-}EQUAL(a,b) \equiv LESS(a,b) \quad \vee \quad (\forall i \exists x\ (R(a,i,x) \wedge R(b,i,x)))$$

This can be generalized to N strings. We compute first whether a string S_a is never found to be the larger of two strings:

$$IS\text{-}MIN(a) \equiv \forall b\ LESS\text{-}OR\text{-}EQUAL(a,b)$$

Whether the i-th symbol of the minimal string is x, is computed as follows:

$$MIN\text{-}SYMBOL(i,x) \equiv \exists a R(a,i,x) \wedge IS\text{-}MIN(a)$$

□

The more difficult part is (a). In case (iii) of the minimal description on Page 9 there is an exception where we have to deviate from the binary tree structure. We compare $i = q-p+1$ split components at a time that have the same size. Suppose, we run into case (iii) recursively. Since there are at most i children of size n/i, we need a circuit with depth $\log i$ to sort i strings of size polynomial in n. Therefore, we must reduce the depth parameter to $d - \log i$ to guarantee the depth bound. We show that with an inductive argument, for split components of size n/i the minimal description can be computed by a sub-circuit of depth $c \log(n/i) = c \log n - c \log i$ (for a constant c). Hence, we can do case (iii) recursively.

Claim 2 *On input of* i *strings of length* N *there is a function that arranges them in lexicographical increasing order, and which is computable by an* $O(\log i)$*-depth bounded* AC^1*-circuit.*

Proof: Let $S_1, \ldots, S_i$ be strings of length N. We use the function $LESS\text{-}OR\text{-}EQUAL$ from the proof of Claim 1 to compare two such strings. By this claim, we can do this comparison in AC^0. We describe each layer from the depth bounded AC^1-circuit.

The procedure is also known from *merge-sort*. In the first layer, we compare whether $S_j < S_{j+1}$ for each odd $j \leq i-1$ in parallel and put them together to tuples of sorted

strings. For $j > 1$, in the j-th layer, we pairwise put together two sorted tuples of strings in parallel. Let $(S_1, \ldots, S_{2^j})$ $(S_{2^j+1}, \ldots, S_{2^j+2^j})$ be two such tuples. We show now how to put them together to a tuple of size 2^{j+1}. For each string, there is a AC^0-computable function that computes at which position it comes.
S_a from the first tuple (i.e. $1 \leq a \leq 2^j$), comes in the new tuple at position $a + b$ (where $1 \leq b \leq 2^j$) if and only if $S_a \leq S_{2^j+b}$ *and* $(b = 1$ *or* $S_a > S_{2^j+b-1})$.
S_b from the second tuple (i.e. $1 \leq b \leq 2^j$), comes in the new tuple at position $a + b$ (where $1 \leq b \leq 2^j$) if and only if $S_{2^j+b} \geq S_a$ *and* $(a = 2^j$ *or* $S_{2^j+b} < S_{a+1})$.
We do these computations for all $a, b \in \{1, \ldots, 2^j\}$ in parallel.

By induction, after $O(\log i)$ layers, there is a sorted tuple of all strings. □

Length of the canon. Another important point is to know the length of the canon. In the minimal description in Section 3.2 we consider balanced binary trees and every bag is described by a word of length $(4k + 3)^2$ followed by its vertices within $(4k + 3) \cdot \lceil \log n \rceil$ bits.

Claim 3 *The size of the minimal description for a balanced binary tree of depth d is $\big[(4k + 3)^2 + (4k + 3) \cdot \lceil \log n \rceil\big] \cdot (2^d - 1)$.*

Proof: The minimal description is filled up to have total length $\big[(4k + 3)^2 + (4k + 3) \cdot \lceil \log n \rceil\big] \cdot (2^d - 1)$ and this holds recursively. The proof goes by induction, let $i = \big[(4k + 3)^2 + (4k + 3) \cdot \lceil \log n \rceil\big]$.

$$\begin{aligned}
|C(G, X, \sigma, d)| &= i \cdot (2^d - 1) \\
&= i \cdot (2 \cdot 2^{(d-1)} - 1) \\
&= i \cdot (1 + 2 \cdot (2^{(d-1)} - 1)) \\
&= i + (i \cdot 2 \cdot (2^{(d-1)} - 1)) \\
&= i + (2 \cdot i \cdot (2^{(d-1)} - 1)) \\
&= |C_0| + |C_0'| + (|C_1| + |C_2|).
\end{aligned}$$

□

Finally, we argue that it is an AC^0-computable task to concatenate the strings of fixed length that are already arranged in lexicographical increasing order. This finishes the proof of Lemma 3.9. □

We summarize, to get an AC^1-circuit that computes the minimal description $C(G, X, \sigma, (1, m), d)$ for the input graph G, we have preprocessing steps to ensure that G has treewidth at most k (by Lemma 1.1), and we have circuits to compute in parallel for pairs of bags X, Y whether Y is a valid child of X. We have $O(\log n)$ levels of small circuits where minimal descriptions for subtrees are computed, selected or sorted. The total size is polynomial and the total depth of the circuit is $O(\log n)$.

Theorem 3.10 *There is a constant c and an AC^1-computable function that on input of graph G of treewidth at most k, root bag X of size at most $4k + 3$, permutation $\sigma \in Sym(X)$, m split components in $G \setminus X$ and depth parameter $d = c \log n$ computes a good minimal description $C(G, X, \sigma, (1, m), d)$ if one exists.*

3.4 The Canonization.

The minimal description depends on some parameters: a bag X, a permutation $\sigma \in Sym(X)$ and depth parameter d. There are at most n^{4k+3} many bags and $(4k+3)!$ permutations for X. According to Lemma 3.5 we can fix $d = c \log_2 n$ (for a constant c) and still get a *good* minimal description. Hence, we set up $n^{4k+3} \cdot (4k+3)!$ many circuits in parallel and compute all possibilities of minimal descriptions. We select in AC^0 the smallest of all these minimal descriptions.

Theorem 3.11 *There is an AC^1-computable function, that computes a canonical invariant for graphs of treewidth at most k.*

Compute the canonical labeling. To obtain a *canonizing function* the algorithm is doing some extra work in parallel. While computing the minimal description we bring the vertices in a unique order. For this, in a minimal description $C = C_0C_0'C_1C_2$ the part C_0' plays a central role.

The minimal description gives an order to the bags. We list now the vertices of the bags in a fixed order and the inserted vertices of each bag with their original vertex names.

To define the canonizing function, we use a simple logspace-computable procedure which can be found e.g. in [DLN08]. The order of the occurrences of all vertices defines a fixed order. After renaming the vertices according to this order we arrange the edges in lexicographical increasing order. Note, the renaming of the vertices is an automorphism from G onto its canon, i.e. this is a *canonical labeling*. Hence, the canonization of graphs of treewidth at most k is in AC^1. This completes the proof of Theorem 1.3.

Conclusion. We improve the upper bound of the canonization problem for bounded treewidth graphs very close to the LogCFL upper bound of isomorphism testing [DTW10]. However, it is not clear how to improve the new AC^1 upper bound with known standard techniques for canonization. In [EJT10a] interesting concepts for bounded treewidth graphs are introduced, they state the question whether these can be used for isomorphism testing or canonization.

Acknowledgment. We thank Jacobo Torán and anonymous referees for comments and helpful discussions.

References

[ACP87] S. Arnborg, D. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic and Discrete Methods*, 8(2):277–284, 1987.

[ADK08] V. Arvind, Bireswar Das, and Johannes Köbler. A logspace algorithm for partial 2-tree canonization. In *Computer Science Symposium in Russia (CSR)*, pages 40–51, 2008.

[BH98] Hans L. Bodlaender and Torben Hagerup. Parallel algorithms with optimal speedup for bounded treewidth. *SIAM Journal on Computing*, 27(6):1725–1746, 1998.

[BK08] Hans L. Bodlaender and Arie M.C.A. Koster. Combinatorial optimization on graphs of bounded treewidth. *The Computer Journal*, 51(3):255–269, 2008.

[BL83] László Babai and Eugene M. Luks. Canonical labeling of graphs. In *15th Annual ACM Symposium on Theory of Computing (STOC)*, pages 171–183, 1983.

[Bod89] Hans L. Bodlaender. NC-algorithms for graphs with small treewidth. In *Proceedings of the 14th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 1–10, 1989.

[Bod90] Hans L. Bodlaender. Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees. *Journal of Algorithms*, 11:631–644, 1990.

[Bod96] Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25(6):1305–1317, 1996.

[Bod98] Hans L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209:1–45, 1998.

[CH88] N. Chandrasekharan and Stephen T. Hedetniemi. Fast parallel algorithms for tree decomposition and parsing partial k-trees. In *Proceedings of the 26th Annual Allerton Conference on Communication, Control, and Computing*, pages 283–292, 1988.

[DLN08] Samir Datta, Nutan Limaye, and Prajakta Nimbhorkar. 3-connected planar graph isomorphism is in log-space. In *Proceedings of the 28th annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 153–162, 2008.

[DLN$^+$09] Samir Datta, Nutan Limaye, Prajakta Nimbhorkar, Thomas Thierauf, and Fabian Wagner. Planar graph isomorphism is in log-space. *Annual IEEE Conference on Computational Complexity (CCC)*, pages 203–214, 2009.

[DNTW09] Samir Datta, Prajakta Nimbhorkar, Thomas Thierauf, and Fabian Wagner. Isomorphism for $K_{3,3}$-free and K_5-free graphs is in log-space. In *Proceedings of the 29th annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 145–156, 2009.

[DTW10] Bireswar Das, Jacobo Torán, and Fabian Wagner. Restricted space algorithms for isomorphism on bounded treewidth graphs. In *Proceedings of the 27th International Symposium on Theoretical Aspects of Computer Science*, pages 227–238, 2010.

[EJT10a] Michael Elberfeld, Andreas Jakoby, and Till Tantau. Logspace versions of the theorems of Bodlaender and Courcelle. In *Proceedings of the 51st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 143–152, 2010.

[EJT10b] Michael Elberfeld, Andreas Jakoby, and Till Tantau. Logspace versions of the theorems of Bodlaender and Courcelle. Technical Report TR10-062, Electronic Colloquium on Computational Complexity (ECCC), 2010.

[GV06] Martin Grohe and Oleg Verbitsky. Testing graph isomorphism in parallel by playing a game. In *Annual International Colloquium on Automata, Languages and Programming (ICALP)*, 2006.

[Imm99] Neil Immerman. *Descriptive Complexity. Graduate Texts in Computer Science.* Springer-Verlag, 1999.

[JKMT03] Birgit Jenner, Johannes Köbler, Pierre McKenzie, and Jacobo Torán. Completeness results for graph isomorphism. *Journal on Computer and System Sciences*, 66(3):549–566, 2003.

[KK09] Johannes Köbler and Sebastian Kuhnert. The isomorphism problem for k-trees is complete for logspace. In *Proceedings of the 34th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 537–548, 2009.

[Köb06] Johannes Köbler. On graph isomorphism for restricted graph classes. In *Second Conference on Computability in Europe (CiE)*, pages 241–256, 2006.

[KV08] Johannes Köbler and Oleg Verbitsky. From invariants to canonization in parallel. In *Third International Computer Science Symposium in Russia (CSR)*, pages 216–227, 2008.

[Lag90] Jens Lagergren. Efficient parallel algorithms for tree-decomposition and related problems. In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 173–182, 1990.

[Lin92] Steven Lindell. A logspace algorithm for tree canonization (extended abstract). In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing (STOC)*, pages 400–404. ACM, 1992.

[Luk93] Eugene M. Luks. Permutation groups and polynomial-time computation. *DIMACS series in Discrete Mathematics and Theoretical Computer Science*, 11:139–175, 1993.

[Ree92] Bruce A. Reed. Finding approximate separators and computing tree width quickly. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing (STOC)*, pages 221–228, 1992.

[Rei08] Omer Reingold. Undirected connectivity in log-space. *Journal of the ACM (JACM)*, 55(4):1–24, 2008.

[RS86] Robertson and Seymour. Graph minors. II. algorithmic aspects of tree-width. *Journal of Algorithms (ALGORITHMS)*, 7(3):309–322, 1986.

[Wan94] Egon Wanke. Bounded tree-width and LOGCFL. *Journal of Algorithms*, 16, 1994.

ECCC ISSN 1433-8092

http://eccc.hpi-web.de