# The Space Complexity of $k$-Tree Isomorphism

V. Arvind[1], Bireswar Das[1], and Johannes Köbler[2]

[1] The Institute of Mathematical Sciences, Chennai 600 113, India
{arvind,bireswar}@imsc.res.in
[2] Institut für Informatik, Humboldt Universität zu Berlin, Germany
koebler@informatik.hu-berlin.de

**Abstract.** We show that isomorphism testing of $k$-trees is in the class StUSPACE($\log n$) (strongly unambiguous logspace). This bound follows from a deterministic logspace algorithm that accesses a strongly unambiguous logspace oracle for canonizing $k$-trees. Further we give a logspace canonization algorithm for $k$-paths.

## 1 Introduction

It often turns out that NP-hard graph problems when restricted to the class of *partial $k$-trees* for constant $k$ have efficient polynomial-time algorithms [Bod88, SS87]. Partial $k$-trees are also known as the class of graphs of treewidth $k$. For constant $k$, in general, they are known as *bounded treewidth graphs* (formal definitions are given in Section 3).

*Graph Isomorphism* is the problem of deciding whether two given graphs are isomorphic. I.e. the problem is to test whether there is a bijective function that maps vertices of the first graph to vertices of the second graph and preserves the edge relation. Graph Isomorphism has attracted much algorithmic research. It is one of the few problems in NP that is neither known to be computable in polynomial time nor to be NP-complete. Polynomial time algorithms have been designed for the problem for several interesting restricted graph classes [Luk82, Mil83, Bab86], including the class of partial $k$-trees [Bod90]. Bodlaender [Bod90] gave the first polynomial-time algorithm for testing the isomorphism of partial $k$-trees. Bodlaender's algorithm, based on dynamic programming, runs in time $O(n^{k+4.5})$.

Our interest is in a *complexity-theoretic* characterization of Graph Isomorphism for partial $k$-trees using space bounded complexity classes. We explain our motivation for studying the space complexity of $k$-tree isomorphism. On the one hand, we have Lindell's result [Lin92, JKMT03] that tree canonization is complete for deterministic logspace,[1] which tightly characterizes the complexity of both isomorphism and canonization of trees. What about partial $k$-tree isomorphism? The recent $\mathrm{TC}^1$ upper bound for isomorphism of partial $k$-trees by Grohe and Verbitsky [GV06] raises the question about a tight complexity-theoretic classification of the problem. It is tempting to conjecture that partial

[1] Provided that the tree is given in the pointer notation; using the parenthesis notation the problem is $\mathrm{NC}^1$-complete [Bus97, JKMT03]

$k$-tree isomorphism should also be complete for deterministic logspace, just like ordinary tree isomorphism. However, the best known complexity bound for even recognizing partial $k$-trees is LOGCFL (the class of decision problems that are logspace many-one reducible to CFLs) [Wan94].

The $\mathrm{TC}^1$ bound of [GV06] suggests that we can put the problem in a natural complexity class contained in $\mathrm{TC}^1$ like LOGCFL or DET, or perhaps somewhere in the logspace counting hierarchy. The logspace counting classes, introduced in the seminal paper [AO96], contain many natural problems sitting in $\mathrm{NC}^2$ and have been used to characterize most natural problems in $\mathrm{NC}^2$ satisfactorily from a complexity-theoretic viewpoint. A comprehensive study can be found in Allender's survey article [All04].

In this paper we show that *full* $k$-tree canonization is in $\mathrm{FL}^{\mathrm{NL}}$. Recall that the *canonization problem* for graphs is to produce a *canonical form* $canon(G)$ for a given graph $G$ such that $canon(G)$ is isomorphic to $G$ and $canon(G_1) = canon(G_2)$ for any pair of isomorphic graphs $G_1$ and $G_2$. Canonization is clearly at least as hard as Graph Isomorphism. In fact, it is easy to see that Graph Isomorphism is even $\mathrm{AC}^0$ reducible to Graph Canonization. However, in general it is not known if the two problems are even polynomial-time equivalent.

Interestingly, the NL oracle required for $k$-tree canonization is a language computed by an NL machine $M$ that is *strongly unambiguous*: for any two configurations $x$ and $y$ of machine $M$ there is at most one computation path from $x$ to $y$. The class of languages accepted by such NL machines is denoted StUSPACE($\log n$) (StUL for short) by Allender and Lange [AL89]. As shown in [AL89], StUL is in fact contained in DSPACE($\log^2 n/\log\log n$), improving the DSPACE($\log^2 n$) bound given by Savitch's theorem. Furthermore, the complexity class StUL is closed under complementation and even closed under logspace Turing reductions [BJLR91, Corollary 15]. Thus, it follows that $k$-tree isomorphism is in StUL. The class StUL is not known to be contained in L. In fact, it contains the class ULIN of unambiguous linear languages [BJLR91] which is not known to be in L. To the best of our knowledge, no explicit example of a language in StUL is known that is not already in L. Thus, $k$-tree isomorphism is the first natural problem in StUL which is not known to be in L.

We note that parallel algorithms are known for $k$-tree isomorphism. For example, in [GSS02] a processor efficient $\mathrm{AC}^2$ algorithm was given for $k$-tree isomorphism. Since $\mathrm{StUL} \subseteq \mathrm{UL} \subseteq \mathrm{NL} \subseteq \mathrm{AC}^1$, our upper bound is tighter than previously known bounds from a complexity-theoretic perspective. We also look into the problem of canonizing $k$-paths, a special case of $k$-trees, and give a logspace canonization algorithm for $k$-paths.

## 2 Preliminaries

By graphs we mean finite simple graphs. For basic graph theoretic definitions we refer the reader to [Die97]. For a graph $G$, let $V(G)$ denote its vertex set and $E(G)$ denote its edge set. The set $\{w \in V(G) \mid \{v,w\} \in E(G)\}$ of all *neighbors* of

$v \in V(G)$ is denoted by $N(v)$. For a subset $U \subseteq V(G)$, we use $G[U]$ to denote the *induced subgraph* of $G$, where $V(G[U]) = U$ and $E(G[U]) = \{e \in E(G) \mid e \subseteq U\}$.

Two graphs $G$ and $H$ are isomorphic if there is an edge-preserving bijection $\tau : V(G) \to V(H)$, i.e., for all $u, v \in V(G)$, $\{u, v\} \in E(G)$ if and only if $\{\tau(u), \tau(v)\} \in E(H)$. In case the vertices of $G$ and $H$ are labeled, then the isomorphism $\tau$ must also preserve the labels.

Next we recall some complexity classes defined by circuits and some space bounded classes.

A language $A$ is in the complexity class $\mathrm{NC}^i$ (resp. $\mathrm{AC}^i$) if there is a uniform family of circuits $\{C_n\}_n$ of depth $O(\log^i n)$ and size $poly(n)$ with internal *AND*, *OR* and *NOT* gates with bounded (resp. unbounded) fan-in that accepts $A$. $\mathrm{TC}^i$ is the extension of $\mathrm{AC}^i$ where we additionally allow unbounded *MAJORITY* gates.

The complexity class L consists of all languages $A$ accepted by a deterministic $O(\log n)$ space bounded Turing machine. NL is defined in the same way by using nondeterministic machines. FL is the class of all functions computable by a deterministic $O(\log n)$ space bounded Turing machine.

A nondeterministic Turing machine $M$ is called *unambiguous*, if for any input $x$, it has at most one accepting computation path.

$M$ is said to be *reach-unambiguous* if it is unambiguous and for any input $x$, there is at most one computation path from the starting configuration to any other configuration.

$M$ is said to be *strongly unambiguous* if it is unambiguous and for any pair of configurations $u$ and $v$ of $M$ there is at most one computation path from $u$ to $v$.

A *mangrove* is a directed acyclic graph such that there is at most one directed path from $i$ to $j$ for any two nodes $i$ and $j$ in the graph. In other words, a directed graph is a mangrove if and only if for any node $u$ the subgraph induced by $u$ and all nodes reachable from $u$ is a rooted directed tree.

Note that an unambiguous machine $M$ is strongly unambiguous if and only if its configuration graph is a mangrove.

A language $A$ is in the class UL (RUL, StUL) if there is an $O(\log n)$ space bounded unambiguous (reach-unambiguous, strongly unambiguous, respectively) Turing machine accepting $A$. It is well known that

$$\mathrm{NC}^1 \subseteq \mathrm{L} \subseteq \mathrm{StUL} \subseteq \mathrm{RUL} \subseteq \mathrm{UL} \subseteq \mathrm{NL} \subseteq \mathrm{AC}^1 \subseteq \mathrm{TC}^1 \subseteq \mathrm{NC}^2.$$

The following result of Allender and Lange [AL89] shows that Savitch's $\log^2 n$ space deterministic simulation of NL can be improved for StUL and RUL.

**Theorem 1.** [AL89] $\mathrm{StUL} \subseteq \mathrm{RUL} \subseteq \mathrm{DSPACE}(\log^2 n / \log\log n)$.

## 3 *k*-Tree Canonization

Let $\mathcal{G}$ be a class of (encodings of) graphs. We say that $f$ computes a *complete invariant* for $\mathcal{G}$, if

$$\forall G, H \in \mathcal{G} : G \cong H \Leftrightarrow f(G) = f(H).$$

A complete invariant $f$ for $\mathcal{G}$ that computes for any graph $G \in \mathcal{G}$ a graph $f(G)$ that is isomorphic to $G$ is called a *canonization* for $\mathcal{G}$. We call $f(G)$ the *canon* of $G$ (w.r.t. $f$).

Notice that if there is a polynomial time computable canonization for $\mathcal{G}$ then the graph isomorphism problem restricted to $\mathcal{G}$ can also be solved in polynomial time. As shown by Gurevich, canonization of general graphs is polynomial-time equivalent to computing a complete invariant [Gur97].

Certain complete invariants are known to be intractable. For example, it is NP-hard to compute the lexicographically least graph (w.r.t. some specific representation) isomorphic to the given graph [BL83]. However, the approach of computing complete invariants has been successful for solving the graph isomorphism problem efficiently for some graph classes [BL83, Spi96].

The classes of $k$-trees and partial $k$-trees were introduced by Arnborg and Proskurowski (see e.g. [AP89]).

**Definition 2.** *The class of* $k$-trees *is inductively defined as follows.*

- *A clique with $k$ vertices ($k$-clique for short) is a $k$-tree.*
- *Given a $k$-tree $G'$ with $n$ vertices, a $k$-tree $G$ with $n+1$ vertices can be constructed by introducing a new vertex $v$ and picking a $k$-clique $C$ in $G'$ and then joining $v$ to each vertex $u$ in $C$. Thus, $V(G) = V(G') \cup \{v\}$, $E(G) = E(G') \cup \{\{u, v\} \mid u \in C\}$.*

*A* partial $k$-tree *is a subgraph of a $k$-tree.*

Before we go into the $k$-tree canonization we notice that the following characterization of $k$-trees gives a logspace algorithm for recognizing $k$-trees.

**Definition 3.** [Klo94] *Let $G = (V, E)$ be a graph. A subset $S$ of $V$ is called a* vertex separator *for two nonadjacent vertices $u, v \in V$, if in the subgraph of $G$ induced by the vertex set $V - S$ the two vertices $u, v$ are in different connected components. A vertex separator $S$ for $u, v$ is called* minimal, *if no proper subset of $S$ is a vertex separator for $u$ and $v$. A subset $S \subseteq V$ is a* minimal vertex separator *if $S$ is a minimal vertex separator for some pair of vertices $u, v \in V$.*

**Lemma 4.** [CI88] *A graph $G$ with $n > k$ vertices is a $k$-tree if and only if*

- *every pair of nonadjacent vertices $u$ and $v$ has a $k$-clique as a minimal vertex separator and*
- *$E(G)$ contains exactly $\binom{k}{2} + k(n-k)$ edges.*

It is easy to see that the two conditions of Lemma 4 can be checked in logspace. Hence, from now on we can assume that the input graph $G$ is a $k$-tree. Further we assume that $V(G) = \{1, \ldots, n\}$.

Our algorithm for $k$-tree canonization works by reducing the problem to the problem of canonizing certain labeled trees that encode essential information about $k$-trees. Our initial goal is to define this labeled tree. For this we use the concept of the layer decomposition of a $k$-tree with respect to a base $B$. This concept was introduced in [KCP82] for testing isomorphism in hookup classes. Subsequently, it was used in [Cha90, GSS02] for the design of efficient $k$-tree isomorphism algorithms.

**Definition 5.** (cf. [KCP82, GSS02]) *Let $G = (V, E)$ be a $k$-tree and let $B \subseteq V$ be a $k$-clique in $G$. Then the $B$-*decomposition *of $G$ is the sequence of sets $B(0), \ldots, B(p)$ such that $B(0) = B$ and $p = \max\{i \geq 0 \mid B(i) \neq \emptyset\}$, where $B(i+1)$ is inductively defined by*

$$B(i+1) = \{v \in V - B[i] \mid N(v) \cap B[i] \text{ is a } k\text{-clique}\}.$$

*Here, $B[i]$ denotes the union $B[i] = B(0) \cup \cdots \cup B(i)$.*

The set $B(i)$ is called the $i$th layer of the $B$-decomposition of $G$. Intuitively, the layers of the $B$-decomposition indicate the order in which vertices could be added to $G$ when we choose $B$ as the initial $k$-clique. More precisely, starting with the $k$-tree $G_0 = G[B]$, $G_{i+1} = G[B[i+1]]$ can be constructed from $G_i = G[B[i]]$ by adding the vertices in $B(i+1)$ to $G_i$. Recall that $v$ can be added to $G_i$ if and only if the set $N(v) \cap B[i]$ of $v$'s neighbors in $B[i]$, henceforth denoted by $N_i(v)$, induces a $k$-clique in $G_i$. In [KCP82], this set $N_i(v)$ is called the *support* of $v \in B(i)$).

If this process is successful, i.e., if each vertices of $G$ is covered by some layer $B(i)$, then $B$ is called a *base* of $G$ (cf. [KCP82]).

We first show that any $k$-clique $B$ in $G$ can be used as a base for constructing $G$ (see Lemma 9).

**Definition 6.** (cf. [GSS02]) *A vertex $v$ of a $k$-tree $G$ is called* simplicial*, if $N(v)$ induces a $k$-clique in $G$.*

**Claim 7.** *Any $k$-tree $G$ with $n \geq k+2$ vertices has at least two nonadjacent simplicial vertices.*

*Proof.* The proof is by induction on $n$. If $n = k+2$, then $G$ is obtained from a $(k+1)$-clique $G'$ by choosing a $k$-clique $C$ in $G'$ and introducing a new node $v$ which is joined to each vertex in $C$. Let $u$ be the unique vertex in $G'$ not covered by $C$. Then $u$ and $v$ are two nonadjacent simplicial vertices in $G$. For the inductive step assume that $G$ has $n > k+2$ vertices. Then $G$ has been obtained from a $k$-tree $G'$ by introducing a new node $v$ and joining it to each vertex in a $k$-clique $C$ of $G'$. Clearly, $v$ is simplicial in $G$. Further, by the induction hypothesis, $G'$ has two nonadjacent simplicial vertices $u_1$ and $u_2$. Since $u_1$ and $u_2$ are nonadjacent, at least one of them does not belong to $C$ and therefore it is also simplicial in $G$. □

**Claim 8.** *Let $B$ be a $k$-clique of a $k$-tree $G$ with $n \geq k+1$ vertices. Then $G$ has a simplicial vertex $v \notin B$.*

*Proof.* If $n = k+1$, then the only vertex not in $B$ is simplicial. If $n > k+1$, then Claim 7 guarantees the existence of two nonadjacent simplicial vertices that cannot be both in $B$. □

**Lemma 9.** *For any $k$-tree $G = (V, E)$ and any $k$-clique $B$, the $B$-decomposition forms a partition of $V$.*

*Proof.* The proof is by induction on $n$. The base case $n = k$ is clear. For the inductive step assume that $n \geq k + 1$ and let $B(0), \ldots, B(p)$ be the $B$-decomposition of $G$. By Claim 8, $G$ has a simplicial vertex $v$ not in $B$. It is easy to prove that $G - v$ is a $k$-tree and hence, by the induction hypothesis, the $B$-decomposition $B'(0), \ldots, B'(p')$ of $G-v$ forms a partition of $V - \{v\}$. Now let $i \geq 0$ be the minimum integer such that $N(v) \subseteq B'[i]$. Then it follows that $B(i+1) = B'(i+1) \cup \{v\}$ and $B(j) = B'(j)$ for all $j \neq i + 1$, implying that $V = B[p]$. □

The following properties of the $B$-decomposition have been proved in [KCP82].

**Proposition 10.** *If $B$ is a base for a $k$-tree $G = (V, E)$, then the $B$-decomposition $B(0), \ldots, B(p)$ has the following properties.*

1. *Any two vertices in $B(i)$, $i \geq 1$, are nonadjacent. Hence, $N_{i-1}(v) = N_i(v)$ for any vertex $v \in B(i)$.*
2. *Any vertex $v \in B(i)$, $i \geq 2$, has a unique neighbor $f(v) \in B(i - 1)$, called the* father *of $v$ w.r.t. $B$.*

In order to efficiently compute information on the $B$-decomposition of a $k$-tree $G$ we use a directed graph $D(G, B)$ which is defined as follows (whenever $G$ and $B$ are clear from the context we simply write $D$ instead of $D(G, B)$). $D$ has the vertex set

$$V(D) = \{B\} \cup \{(C, v) \mid v \notin C \text{ and } C \cup \{v\} \text{ is a } (k + 1)\text{-clique in } G\}$$

and the vertices of $D$ are joined by the directed edges in the set

$$\begin{aligned} E(D) = \{&(B, (B, v)) \mid (B, v) \in V(D)\} \cup \\ &\{((C, v), (C', v')) \mid v \in C', v' \notin C, \|C \cap C'\| = k - 1\}. \end{aligned}$$

This means that in $D$ we provide a transition from $(C, v)$ to $(C', v')$ if $C'$ can be obtained from $C$ by replacing some vertex $u \in C$ by $v$, i.e., $C' = (C - \{u\}) \cup \{v\}$. Our next aim is to show that $D$ is a mangrove (see Lemma 13).

**Claim 11.** *For any vertex $v \in B(i)$, $i \geq 1$, $D$ has a directed path of length $i$ from $B$ to $(N_{i-1}(v), v)$.*

*Proof.* We prove the claim by induction on $i$. The base case $i = 1$ is clear. For the inductive step assume that $v \in B(i)$, $i \geq 2$ and let $f(v) \in B(i - 1)$ be the father of $v$. By the induction hypothesis it follows that $D$ has a directed path of length $i - 1$ from $B$ to $(N_{i-2}(f(v)), f(v))$. Clearly, $f(v) \in N_{i-1}(v)$ and $v \notin N_{i-2}(f(v))$. Further, since $f(v)$ is the only vertex in $N_{i-1}(v)$ belonging to $B(i - 1)$, the remaining $k - 1$ vertices belong to $B[i - 2]$ and, as they are also neighbors of $f(v)$, they belong to the support $N_{i-2}(f(v))$ of $f(v)$. This shows that $D$ has an edge from $(N_{i-2}(f(v)), f(v))$ to $(N_{i-1}(v), v)$. □

**Claim 12.** *If $D$ has a directed path $B, (C_1, v_1), \ldots, (C_{i-1}, v_{i-1}), (C, v)$ of length $i \geq 1$ from $B$ to some vertex $(C, v)$, then $v \in B(i)$ and $C = N_{i-1}(v) \subseteq B \cup \{v_1, \ldots, v_{i-1}\}$.*

*Proof.* Again the proof is by induction on $i$. If $E(D)$ contains the edge $(B, (B, v))$, then clearly $v \in B(1)$ via the support $N_0(v) = B$.

For the inductive step let us assume that $B, (C_1, v_1), \ldots, (C_{i-1}, v_{i-1}), (C, v)$ is a directed path of length $i \geq 2$ from $B$ to $(C, v)$. Then by the induction hypothesis it follows that $v_{i-1} \in B(i-1)$ via the support $C_{i-1} = N_{i-2}(v_{i-1}) \subseteq B \cup \{v_1, \ldots, v_{i-2}\}$. As $N_{i-2}(v_{i-1}) = N_{i-1}(v_{i-1})$ by part 1 of Proposition 10, this implies that $C_{i-1}$ contains all neighbors of $v_{i-1}$ in $B[i-1]$. Since $v$ is a neighbor of $v_{i-1}$ that does not belong to $C_{i-1}$, $v$ cannot be in $B[i-1]$. As $((C_{i-1}, v_{i-1}), (C, v)) \in E(D)$, it follows that $C$ is obtained from $C_{i-1}$ by replacing some vertex $u$ in $C_{i-1}$ by $v_{i-1}$, i.e., $C = (C_{i-1} - \{u\}) \cup \{v_{i-1}\} \subseteq B \cup \{v_1, \ldots, v_{i-1}\}$. Hence, all vertices in $C$ belong to $B[i-1]$ and are adjacent to $v$, implying that $v \in B(i)$ via the support $N_{i-1}(v) = C$ (notice that $C \subsetneq N_{i-1}(v)$ would imply $v \notin B[p]$). □

**Lemma 13.** *For any $k$-clique $B$ in a $k$-tree $G$, the graph $D(G, B)$ is a mangrove.*

*Proof.* We first show that $D = D(G, B)$ does not have different paths from $B$ to the same node $(C, v)$.

By Claim 12, all paths from $B$ to $(C, v)$ have the same length. In order to derive a contradiction let $i$ be minimal such that there are two different paths $B, (C_1, v_1), \ldots, (C_{i-1}, v_{i-1}), (C, v)$ and $B, (C_1', v_1'), \ldots, (C_{i-1}', v_{i-1}'), (C, v)$ of length $i$ from $B$ to some node $(C, v)$. Then $v_{i-1}$ and $v_{i-1}'$ must be different, since otherwise Claim 11 implies that $C_{i-1} = N_{i-2}(v_{i-1}) = N_{i-2}(v_{i-1}') = C_{i-1}'$, contradicting the minimality of the path length $i$. But now Claim 12 implies that $v_{i-1}$ and $v_{i-1}'$ both belong to $B(i-1)$ as well as to the support $C$ of $v$, contradicting part 2 of Proposition 10.

To complete the proof suppose there are different directed paths between two nodes $(C, v)$ and $(C', v')$ in $D(G, B)$. Then we would also have different directed paths between the two nodes $C$ and $(C', v')$ in $D(G, C)$, contradicting the argument above. □

Now let $T = T(G, B)$ be the subgraph of $D(G, B)$ induced by the vertices reachable from $B$. Then Lemma 13 implies that $T$ is a directed rooted tree with root $B$.

In fact, from Claims 11 and 12 it is immediate that by projecting the first component out from the nodes $(C, v) \in V(T)$ we get exactly the tree $T(G)$ defined in [KCP82]. There, the following labeling with respect to a bijection $\theta : B \to \{1, 2, \ldots, k\}$ has been defined.

Let $(C, v)$ be a node in $T$. W.l.o.g. suppose that $C = \{v_1, \ldots, v_k\}$ where $C \cap B = \{v_1, \ldots, v_m\}$ for some $m \leq k$. Notice that by part 1 of Proposition 10, the $k - m$ vertices in $C - B$ belong to $k - m$ different layers $B(i_1), \ldots, B(i_{k-m})$. Then vertex $(C, v)$ is labeled by the set $\{\theta(v_1), \ldots, \theta(v_m), k + i_1, \ldots, k + i_{k-m}\}$.

We denote the tree $T$ together with this labeling by $T(G, B, \theta)$. The following theorem is due to [KCP82].

**Theorem 14.** *Let $G$ and $G'$ be two $k$-trees, let $B$ be a base for $G$ and let $\theta : B \to \{1, \dots, k\}$ be a bijection. Then $G$ and $G'$ are isomorphic if and only if there exists a base $B'$ for $G'$ and a bijection $\theta' : B' \to \{1, \dots, k\}$ such that the two labeled trees $T(G, B, \theta)$ and $T(G', B', \theta')$ are isomorphic.*

The proof of Theorem 14 crucially hinges on the fact that each isomorphic copy $T'$ of the labeled tree $T(G, B, \theta)$ provides enough information to reconstruct $G$ from $T'$ up to isomorphism. To see why, for $i \geq 1$ let $B_i$ be the set of vertices of $T'$ that have distance $i$ from the root of $T'$ and let $p$ be the maximum distance of any vertex in $T'$ from the root. Then starting with a $k$-clique $G_0$ we can successively add in parallel all the vertices $v \in B_i$ to $G_{i-1}$ for $i = 1, \dots, p$. The crucial observation is that the labeling $\{\theta(v_1), \dots, \theta(v_m), k + i_1, \dots, k + i_{k-m}\}$ of the node $v$ in $T'$ tells us to which vertices in $G_{i-1}$ vertex $v$ should be connected (recall that Claim 12 guarantees that all vertices in the support of a node either belong to the base or lie on the path from the root to that vertex in the corresponding tree).

To canonize $k$-trees we use Lindell's [Lin92] deterministic logspace canonization algorithm for trees which can be made to work for any labeled tree by constructing gadgets for labels. More precisely, consider the algorithm $A$ that on input a $k$-tree $G$ computes the canon of all labeled trees $T(G, B, \theta)$ for all $k$-cliques $B$ in $G$ and all bijections $\theta : B \to \{1, \dots, k\}$ and picks the lexicographically least among them. Then Theorem 14 implies that

- if two $k$-trees $G$ and $H$ are isomorphic then any tree of the form $T(G, B, \theta)$ is isomorphic to some tree of the form $T(H, B', \theta')$ and
- if $G$ and $H$ are non-isomorphic then no tree of the form $T(G, B, \theta)$ is isomorphic to some tree of the form $T(H, B', \theta')$.

Hence, $A$ outputs the same tree $T$ for both $k$-trees $G$ and $H$ if and only if $G$ and $H$ are isomorphic, implying that $A$ computes a complete invariant for $k$-trees.

Furthermore, as explained above, the output tree $T$ of $A$ on input $G$ provides enough information to reconstruct $G$ from $T$ in logspace up to isomorphism. The combination of $A$ with this reconstruction procedure thus yields the desired canonization algorithm $A'$ for $k$-trees. It remains to show that $A$ can be implemented in $\mathrm{FL}^{\mathrm{StUL}}$. In the next lemma we show that the labeled trees $T(G, B, \theta)$ can be computed in logspace relative to some oracle in StUL. The following claim provides this oracle.

**Claim 15.** *The problem of deciding whether a vertex $(C, v)$ of $D$ has distance $i$ from $B$ is in* StUL*.*

*Proof.* The algorithm tries to guess a path of length $i$ from $B$ to $(C, v)$ in the tree $T = T(G, B)$. For that, starting with vertex $B$, it iteratively guesses a next node $(C', v')$ and checks if $T$ provides an edge from the actual node to that node. If after $i$ steps the algorithm reaches $(C, v)$ then it accepts, otherwise it rejects.

Clearly, the algorithm runs in logspace since it has to store only two nodes of $T$ and some counters. Since $D(G, B)$ is a mangrove by Claim 13, it is easy to see that the configuration graph is also a mangrove. □

**Lemma 16.** *On input a $k$-tree $G$, a $k$-clique $B$ and a bijection $\theta : B \to \{1, \dots, k\}$, the labeled tree $T(G, B, \theta)$ can be computed in logspace relative to some oracle in* StUL*.*

*Proof.* The algorithm for generating $T = T(G, B, \theta)$ first outputs $V(T)$ by checking for each node $(C, v)$ in $V(D)$ whether it is reachable from $B$ by using the StUL oracle of Claim 15. If so, it computes the label of $(C, v)$ by recomputing the layer numbers of all the vertices in $C$ (again using the StUL oracle). Finally, for each distinct pair of nodes in $V(T)$ it checks whether $D$ provides a directed edge between them. □

This shows that the algorithm $A$ described above can indeed be implemented in logspace relative to some oracle in StUL. Hence, we can state our main result.

**Theorem 17.** *For each fixed $k$ there is a canonizing algorithm for $k$-trees that runs in* $\mathrm{FL}^{\mathrm{StUL}}$*.*

As StUL is closed under logspace Turing reductions [BJLR91, Corollary 15], we immediately get the following complexity upper bound for testing isomorphism for $k$-trees.

**Corollary 18.** *The isomorphism problem for $k$-trees is in* StUL*.*

## 4 *k*-Path Canonization

A $k$-path is a special type of $k$-tree. The subgraphs of $k$-paths are called partial $k$-paths. They coincide with the graphs of *pathwidth* at most $k$ [Pro89]. In [GNPR05] a polynomial time algorithm for subgraph isomorphism for bounded pathwidth graphs was given. Here we look at the space complexity of the canonization problem for $k$-paths.

**Definition 19.** *An* interval graph *is a graph whose vertices can be put in one to one correspondence with a set of open intervals on the real line such that two vertices are adjacent if and only if the corresponding intervals have a nonempty intersection.*

**Definition 20.** [Klo94] *A $k$-*path *is a $k$-tree which is an interval graph.*

An alternative constructive definition of $k$-paths is given in [GNPR05]. The idea is to restrict the choice of the $k$-clique used as support for adding a new vertex depending on the support of the previously added vertex. The restriction can be best described by maintaining the notion of *current clique*.

Initially the starting clique is the current clique. When a new vertex is added it is joined to each vertex in the current clique. After adding the new vertex the current clique may remain the same (in that case the new vertex added becomes simplicial) or it may change by dropping a vertex and adding the new vertex in the current clique. Clearly, when a vertex is dropped it cannot come back in the current clique.

The difference between the definition of $k$-tree and the constructive definition of $k$-path is that for $k$-trees a new vertex can be joined to any $k$-clique when expanding a $k$-tree, whereas for $k$-paths a new vertex can only be added to the current clique of a $k$-path.

From this constructive definition of $k$-paths the following characterization of $k$-paths in the terminology of Section 3 can be obtained. Recall that a *caterpillar* is a rooted tree in which each node has at most one child that is not a leaf.

**Lemma 21.** *A $k$-tree $G$ is a* $k$-path *if and only if for some base $B$ of $G$, the tree $T(G, B)$ is a* caterpillar*.*

*Proof (sketch).* Assume that $G = (V, E)$ is a $k$-path and let $C_i$, $i = k, \ldots, n-1$, be the current $k$-clique that has been used as support for adding vertex $v_{i+1}$ to $G_i = G[\{v_1, \ldots, v_i\}]$, where $C_k = \{v_1, \ldots, v_k\}$ is the initial $k$-clique. Notice that $C_i \neq C_{i+1}$ implies $C_j \neq C_i$ for all $j > i$. Now it is easy to verify that $T = T(G, C_1)$ is a caterpillar with vertices $C_k, (C_k, v_{k+1}), \ldots, (C_{n-1}, v_n)$ containing for each $j \geq k$ with $C_j = C_k$ the edge $(C_k, (C_k, v_{j+1}))$ and for each pair $i, j$ with $C_i \neq C_{i+1} = C_j$ the edge $((C_i, v_{i+1}), (C_j, v_{j+1}))$.

For the backward direction assume that $T = T(G, B)$ is a caterpillar and let $B(0), \ldots, B(p)$ be the $B$-decomposition of $G$. We call $v \in V - B$ a leaf node if $(N_{i-1}(v), v)$ is a leaf in $T$. Now we can order the vertices of $G$ in such a way that all the vertices in $B(i)$ precede the vertices in $B(i+1)$ and within each layer $B(i)$, $i > 0$, the leaf nodes come first. Let $v_1, \ldots, v_n$ be such an ordering. Then it is easy to verify that we can construct $G$ from the initial $k$-tree $G_k = G[B] = G[\{v_1, \ldots, v_k\}]$ by successively adding the vertices $v_{i+1}$ to $G_i = G[\{v_1, \ldots, v_i\}]$ using $N_i(v_{i+1})$ as the current clique. □

To canonize a given $k$-path $G$ we use a similar approach as the one that we used in Section 3 for $k$-trees. In fact, the only difference is that now our algorithm $A$ additionally checks for each base $B$ whether $T(G, B)$ is a caterpillar. Notice that this can easily be done in logspace as follows.

Starting with the root $B$ as the current node, the algorithm verifies that the current node has at most one child $(C', v')$ in $T(G, B)$ that is not a leaf and then proceeds with $(C', v')$ as the next current node (if the current node has two or more non leaf children, the algorithm detects that $T(G, B)$ is not a caterpillar).

As soon as the algorithm reaches a node that has only leaves as children it decides that $T(G, B)$ is a caterpillar and starts to compute the canons of the labeled trees $T(G, B, \theta)$ for all bijections $\theta : B \longrightarrow \{1, \ldots, k\}$ as explained in Section 3.

Since for a caterpillar $T(G, B)$ the oracle described in Claim 15 is clearly decidable in logspace, we have proved the following result.

**Theorem 22.** *For each fixed $k$ there is a logspace canonizing algorithm for $k$-paths. Hence, the isomorphism problem for $k$-paths is in* L.

## References

[AL89] Allender, E., Lange, K.-J.: RUSPACE(log n) is contained in DSPACE($\log^2$ n/loglog n). Theory of Computing Systems 31, 539–550 (1989)

[All04] Allender, E.: Arithmetic circuits and counting complexity classes. In: Krajíček, J. (ed.) Complexity of Computations and Proofs, Seconda Universita di Napoli. Quaderni di Matematica, vol. 13, pp. 33–72 (2004)

[AO96] Allender, E., Ogihara, M.: Relationships among PL, #L and the determinant. R.A.I.R.O. Informatique Théorique et Applications 30(1), 1–21 (1996)

[AP89] Arnborg, S., Proskurowski, A.: Linear time algorithms for NP-hard problems restricted to partial $k$-trees. Discrete Applied Mathematics 23(2), 11–24 (1989)

[Bab86] Babai, L.: A Las Vegas-NC algorithm for isomorphism of graphs with bounded multiplicity of eigenvalues. In: Proc. 27th IEEE Symposium on the Foundations of Computer Science, pp. 303–312. IEEE Computer Society Press, Los Alamitos (1986)

[BJLR91] Buntrock, G., Jenner, B., Lange, K.-J., Rossmanith, P.: Unambiguity and fewness for logarithmic space. In: Budach, L. (ed.) FCT 1991. LNCS, vol. 529, pp. 168–179. Springer, Heidelberg (1991)

[BL83] Babai, L., Luks, E.: Canonical labeling of graphs. In: Proc. 15th ACM Symposium on Theory of Computing, pp. 171–183 (1983)

[Bod88] Bodlaender, H.: Dynamic programming on graphs with bounded treewidth. In: Lepistö, T., Salomaa, A. (eds.) Automata, Languages and Programming. LNCS, vol. 317, pp. 105–118. Springer, Heidelberg (1988)

[Bod90] Bodlaender, H.: Polynomial algorithm for graph isomorphism and chromatic index on partial $k$-trees. Journal of Algorithms 11(4), 631–643 (1990)

[Bus97] Buss, S.: Alogtime algorithms for tree isomorphism, comparison, and canonization. In: Gottlob, G., Leitsch, A., Mundici, D. (eds.) KGC 1997. LNCS, vol. 1289, pp. 18–33. Springer, Heidelberg (1997)

[Cha90] Chandrasekharan, N.: Isomorphism testing of $k$-trees is in NC. Information Processing Letters 34(6), 283–287 (1990)

[CI88] Chandrasekharan, N., Iyengar, S.S.: NC algorithms for recognizing chordal graphs and $k$ trees. IEEE Transactions on Computers 37(10), 1178–1183 (1988)

[Die97] Diestel, R.: Graph Theory. Graduate Texts in Mathematics, vol. 173. Springer, Heidelberg (1997)

[GNPR05] Gupta, A., Nishimura, N., Proskurowski, A., Ragde, P.: Embeddings of $k$-connected graphs of pathwidth $k$. Discrete Applied Mathematics 145(2), 242–265 (2005)

[GSS02] Del Greco, J.G., Sekharan, C.N., Sridhar, R.: Fast parallel reordering and isomorphism testing of $k$-trees. Algorithmica 32(1), 61–72 (2002)

[Gur97] Gurevich, Y.: From invariants to canonization. Bulletin of the European Association of Theoretical Computer Science (BEATCS) 63, 115–119 (1997)

[GV06] Grohe, M., Verbitsky, O.: Testing graph isomorphism in parallel by playing a game. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4051, pp. 3–14. Springer, Heidelberg (2006)

[JKMT03] Jenner, B., Köbler, J., McKenzie, P., Torán, J.: Completeness results for graph isomorphism. Journal of Computer and System Sciences 66, 549–566 (2003)

[KCP82] Klawe, M.M., Corneil, D.G., Proskurowski, A.: Isomorphism testing in hookup classes. SIAM Journal of Algebraic Discrete Methods 3(2), 260–274 (1982)

[Klo94] Kloks, T. (ed.): Treewidth. LNCS, vol. 842. Springer, Heidelberg (1994)

[Lin92] Lindell, S.: A logspace algorithm for tree canonization. In: Proc. 24th ACM Symposium on Theory of Computing, pp. 400–404. ACM Press, New York (1992)

[Luk82] Luks, E.: Isomorphism of bounded valence can be tested in polynomial time. Journal of Computer and System Sciences 25, 42–65 (1982)

[Mil83] Miller, G.L.: Isomorphism of $k$-contractible graphs. Information and Computation 56(1/2), 1–20 (1983)

[Pro89] Proskurowski, A.: Maximal graphs of path-width k or searching a partial k-caterpillar. Technical Report UO-CIS-TR-89-17, University of Oregon (1989)

[Spi96] Spielman, D.A.: Faster isomorphism testing of strongly regular graphs. In: Proc. 28th ACM Symposium on Theory of Computing, pp. 576–584. ACM Press, New York (1996)

[SS87] Scheffler, P., Seese, D.: A combinatorial and logical approach to linear-time computability. In: Davenport, J.H. (ed.) ISSAC 1987 and EUROCAL 1987. LNCS, vol. 378, pp. 379–380. Springer, Heidelberg (1989)

[Wan94] Wanke, E.: Bounded tree-width and LOGCFL. Journal of Algorithms 16(3), 470–491 (1994)