The Isomorphism Problem for k-Trees is Complete for Logspace

Johannes Köbler and Sebastian Kuhnert

Institut für Informatik, Humboldt Universität zu Berlin, Germany,
{koebler,kuhnert}@informatik.hu-berlin.de

Abstract. We show that k-tree isomorphism can be decided in logarithmic space by giving a logspace canonical labeling algorithm. This improves over the previous StUL upper bound and matches the lower bound. As a consequence, the isomorphism, the automorphism, as well as the canonization problem for k-trees are all complete for deterministic logspace. We also show that even simple structural properties of k-trees are complete for logspace.
Keywords: graph isomorphism, graph canonization, k-trees, space complexity, logspace completeness.

1 Introduction

Two graphs G and H are called *isomorphic* if there is a bijective mapping ϕ between the vertices of G and the vertices of H that preserves the adjacency relation, i.e., ϕ relates edges to edges and non-edges to non-edges. *Graph Isomorphism* (GI) is the problem of deciding whether two given graphs are isomorphic. The problem has received considerable attention since it is one of the few natural problems in NP that are neither known to be NP-complete nor known to be solvable in polynomial time.

It is known that GI is contained in coAM [GS86,Sch88] and in SPP [AK06] providing strong evidence that GI is not NP-complete. On the other hand, the strongest known hardness result due to Torán [Tor04] says that GI is hard for the class DET (cf. [Coo85]). DET is a subclass of NC^2 (even of TC^1) and contains NL as well as all logspace counting classes [AJ93,BDH$^+$92].

For some restricted graph classes the known upper and lower complexity bounds for the isomorphism problem match. For example, a linear time algorithm for tree isomorphism was already known in 1974 to Aho, Hopcroft and Ullman [AHU74]. In 1991, an NC algorithm was developed by Miller and Reif [MR91], and one year later, Lindell [Lin92] obtained an L upper bound. On the other hand, in [JKM$^+$03] it is shown that tree isomorphism is L-hard (provided that the trees are given in pointer notation). In [ADK08], Lindell's log-space upper bound has been extended to the class of partial 2-trees, a class of planar graphs also known as generalized series-parallel graphs. Very recently, it has been shown that even the isomorphism problem for all planar graphs is in logspace [DLN$^+$08]. Much of the recent progress on logspace algorithms for

graphs has only become possible through Reingold's result that connectivity in undirected graphs can be decided in deterministic logspace [Rei05]. Our result does not depend on this, yielding a comparatively simple algorithm.

In this paper we show that the isomorphism problem for k-trees is in logspace for each fixed $k \in \mathbb{N}^+$. This improves the previously known upper bound of StUL [ADK07] and matches the lower bound. In fact, we prove the formally stronger result that a canonical labeling for a given k-tree is computable in logspace. 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. Clearly, graph isomorphism reduces to graph canonization. A *canonical labeling* for G is any isomorphism between G and $canon(G)$. It is not hard to see that even the search version of GI (i.e., computing an isomorphism between two given graphs in case it exists) as well as the automorphism group problem (i.e., computing a generating set of the automorphism group of a given graph) are both logspace reducible to the canonical labeling problem.

The parallel complexity of k-tree isomorphism has been previously investigated by Del Greco, Sekharan, and Sridhar [GSS02] who introduced the concept of the *kernel* of a k-tree in order to restrict the search for an isomorphism between two given k-trees. We show that the kernel of a k-tree can be computed in logspace and exploit this fact to restrict the search for a canonical labeling of a given k-tree G. To be more precise, we first transform G into an undirected tree $T(G)$ whose nodes are formed by the k-cliques and $(k+1)$-cliques of G. Then we compute the center node of $T(G)$ which coincides with the kernel of G and try all labelings of the vertices in $\ker(G)$. In order to extend a labeling of the kernel vertices of G to the other vertices of G in a canonical way, we color the nodes of the tree $T(G)$ to encode additional structural information about G. Finally, we apply a variant of Lindell's algorithm to compute canonical labelings for the colored versions of $T(G)$ and derive from them a canonical labeling for the k-tree G.

Our tree representation $T(G)$ is similar to the construction used in [ADK07]. The main advantage of our construction lies in the fact that the tree $T(G)$ can be directly constructed in logspace from G, whereas the tree representation of [ADK07] is obtained as a reachable subgraph of a mangrove[1] based on G and hence can only be derived from G with the help of an StUL oracle.

2 Preliminaries

As usual, **L** is the class of all languages decidable by Turing machines with read-only input tape and an $\mathcal{O}(\log n)$ bound on the space used on the working tapes. **FL** is the class of all functions computable by Turing machines that additionally have a write-only output tape.

Given a graph G, we use $\boldsymbol{V(G)}$ and $\boldsymbol{E(G)}$ to denote its vertex and edge sets, respectively. We define the following notations for subgraphs of G. For

[1] A mangrove is a digraph with at most one directed path between each pair of nodes.

$M \subseteq V(G)$, $\boldsymbol{G[M]}$ denotes the subgraph of G induced by M and we use $\boldsymbol{G - M}$ as a shorthand for $G[V(G) \setminus M]$.

Given a graph G and two vertices $u, v \in V(G)$, the distance $\boldsymbol{d_G(u, v)}$ is the length of the shortest path from u to v. The **eccentricity** of a vertex $v \in V(G)$ is the longest distance to another vertex, i.e., $\boldsymbol{ecc_G(v)} = \max\{d_G(u, v) | u \in V(G)\}$. The **center** of G consists of all vertices with minimal eccentricity.

Given two graphs G and H, an **isomorphism** from G to H is a bijection $\phi\colon V(G) \to V(H)$ with $\{u, v\} \in E(G) \Leftrightarrow \{\phi(u), \phi(v)\} \in E(H)$. On colored graphs, an isomorphism must additionally preserve colors. G and H are called **isomorphic**, in symbols $G \cong H$, if there is an isomorphism from G to H. Given a graph class $\mathcal{G}$, a function f defined on $\mathcal{G}$ computes an **invariant** for $\mathcal{G}$ if

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

If the reverse implication also holds, f is a **complete invariant** for $\mathcal{G}$. If additionally $f(G) \cong G$ for all $G \in \mathcal{G}$, f computes **canonical forms** for $\mathcal{G}$. Given a function f that computes canonical forms, an isomorphism ψ_G from G to its canonical form $f(G)$ is called a **canonical labeling**.

The isomorphisms from a graph G to itself are called **automorphisms** and they form a group, which we denote by $\mathbf{Aut(G)}$. An automorphism is called **non-trivial** if it is not the identity. The **graph automorphism problem (GA)** is to decide if a graph has non-trivial automorphisms. A graph without non-trivial automorphisms is called **rigid**.

In the next section, we present an FL algorithm that, given a k-tree G, computes a canonical labeling ψ_G.

3 Canonizing k-trees

Fix any $k \in \mathbb{N}^+$. The class of ***k*-trees** is inductively defined as follows. Any k-clique is a k-tree. Further, given a k-tree G and a k-clique C in G, one can construct another k-tree by adding a new vertex v and connecting v to every vertex in C. The initial k-clique is called **base** of G, and the k-clique C the new vertex v is connected to is called **support** of v. Note that each k-clique of a k-tree G can be used as base for constructing G – but once the base is fixed, the support of each vertex is uniquely determined.

An interesting special case of k-trees are ***k*-paths**, where the support C_i of any new vertex v_i (except the first vertex added to G) must either contain the vertex v_{i-1} added in the previous step or be equal to the support C_{i-1} of the latter. Fig. 1 shows a 2-tree that is a 2-path as well.

We note that k-trees can be recognized in logspace [ADK07], so we can safely assume that the input is indeed a k-tree.

We first define a tree representation $T(G)$ for k-trees G.

Definition 1. *For a k-tree G, its **tree representation $\boldsymbol{T(G)}$** is defined by*

$$V(T(G)) = \left\{M \subseteq V(G) \;\middle|\; M \text{ is a } k\text{-clique or a } (k+1)\text{-clique}\right\}$$
$$E(T(G)) = \left\{\{M_1, M_2\} \subseteq V(T(G)) \;\middle|\; M_1 \subsetneq M_2\right\}\,.$$

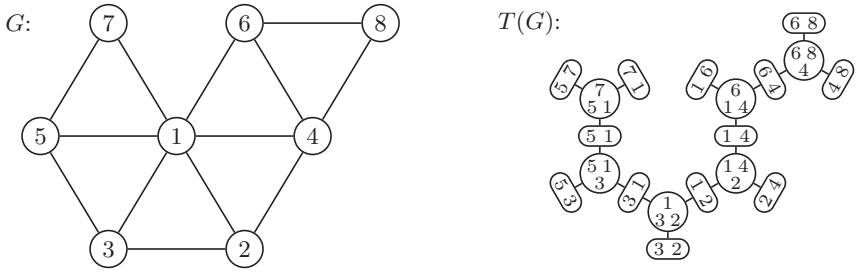

Fig. 1. A 2-tree G and its tree representation $T(G)$

Note that $T(G)$ reflects the iterative construction of G: The base of G is a k-clique and thus a node in $T(G)$. Each time a new vertex u is added to G, it is connected to all vertices of its support P_u (a k-clique), forming a new $(k+1)$-clique C_u that is a superset of P_u. In $T(G)$, the addition of u results in a new node C_u being added and connected to P_u. Additionally, the k many k-cliques in C_u that contain the new vertex u are added as new nodes to $T(G)$ and connected to C_u. From these observations it is clear that $T(G)$ is indeed a tree.

We continue by proving some basic properties of our tree representation $T(G)$.

Lemma 2. *For any k-tree G and any vertex $v \in V(G)$, the nodes of $T(G)$ that contain v form a subtree of $T(G)$.*

Proof. We prove by induction over the construction of G that any node M added to $T(G)$ with $v \in M$ either is the unique node first introducing v or is hooked up to a previously added node that contains v. If M is a k-clique in G this is immediately clear as it is either the base node in $T(G)$ or it is a subset of a $(k+1)$-clique node and hence does not introduce any new vertices. So assume that M is a $(k+1)$-clique C_u, which was added to $T(G)$ upon the addition of some vertex u to G. If $u = v$ then M is the single node of $T(G)$ introducing v. If $u \neq v$ we have $v \in C_u \setminus \{u\} = P_u$ and thus there is an edge to a previously added k-clique node P_u that contains v. □

Lemma 3. *For any k-tree G, the center of $T(G)$ is a single node.*

Proof. Suppose not. Then the center consists of two adjacent nodes, one a k-clique and one a $(k+1)$-clique. This leads to a contradiction because k-clique nodes have even eccentricity while that of $(k+1)$-clique nodes is odd: All leaves are k-clique nodes, and k-cliques and $(k+1)$-cliques alternate on every path. □

Definition 4. *The clique corresponding to the center node of $T(G)$ is called* ***kernel*** *of G and denoted* $\mathbf{ker}(\boldsymbol{G})$.

Note that $\ker(G)$ can be either a k-clique or a $(k+1)$-clique, depending on the structure of G. The concept of the kernel of a k-tree was introduced in [GSS02]. The definition there is slightly different but the equivalence can be easily verified.

We continue by recalling some basic facts concerning undirected trees.

Fact 5. *Given an undirected tree T and two nodes $u, v \in V(T)$, the distance $d_T(u, v)$ can be computed in* FL.

Proof. Think of T as rooted in u and all edges directed away from u. The direction of an edge e can be determined in logspace by computing the lexicographically-first Euler tour starting at u that visits each edge once per direction (cf. [AM04]). Then the unique path from v to u can be found by always choosing the unique incoming edge as next step. Only the current node and the number of steps have to be remembered. Upon reaching u, output the number of steps taken. □

Fact 6. *The center of an undirected tree T can be computed in* FL.

Proof. We first show that the eccentricity $ecc_T(u)$ of each node $u \in V(T)$ is computable in logspace. This can be done by iterating over all $v \in V(T)$, each time calculating $d_T(u, v)$ (this is possible in logspace by Fact 5). Only the maximum distance to u has to be remembered, the result being $ecc_T(u)$.

Observe now that also the maximum eccentricity $ecc_{\max}$ of all nodes $u \in V(T)$ is computable in logspace by iterating over all $u \in V(T)$. Then compute again the eccentricity of all nodes u, this time outputting u if $ecc_T(u) = ecc_{\max}$. □

Our goal is to canonize G by using Lindell's algorithm [Lin92] to canonize $T(G)$. To achieve this, we declare the kernel K of G as the root of $T(G)$. As a consequence, we can identify each $(k+1)$-clique $M \in V(T(G)) \setminus \{K\}$ with the unique vertex $v \in M$ that is not present in the k-clique M' that lies next to M on the path from K to M in $T(G)$. For later use, we denote this vertex by $v(M)$ and for each $v \in V(G) \setminus K$, we use M_v to denote the unique $(k+1)$-clique $M \in V(T(G)) \setminus \{K\}$ with $v(M) = v$.

It is clear that $T(G)$ does not provide complete structural information about G, since the vertices in the kernel K are indistinguishable in $T(G)$ and further, only one out of the k edges between each added vertex u and its support can be recovered from $T(G)$. To add the missing information, we give individual colors to the kernel vertices and color the nodes of $T(G)$ as well. Since the kernel K of a given k-tree G can be determined in logspace, we can simplify the notation by assuming that K consists of the vertices $1, \ldots, k'$, where $k' = \|K\| \in \{k, k+1\}$ equals the size of K.

Definition 7. *Let G be a k-tree with vertex set $V(G) = \{1, \ldots, n\}$ and kernel $K = \{1, \ldots, k'\}$. For each vertex v of G, we denote by*

$$\boldsymbol{l_G(v)} = \min \left\{ d_{T(G)}(K, M) \;\middle|\; M \in V(T(G)), v \in M \right\}$$

the ***level*** *of v in G. Further, for any permutation $\pi \in S_{k'}$, let $\boldsymbol{T(G, \pi)}$ denote the directed colored tree obtained from $T(G)$ by choosing K as the root and coloring each node $M \in V(T(G))$ by the set $c(M) = \{c(v) \mid v \in M\}$, where*

$$\boldsymbol{c(v)} = \begin{cases} \pi(v) & \text{if } v \in \ker(G), \\ l_G(v) + k' & \text{otherwise.} \end{cases}$$

The definition of $T(G,\pi)$ is similar to the construction of the colored tree $T(G,B,\theta)$ in [ADK07]. The main advantage of our construction lies in the fact that $T(G,\pi)$ can be directly constructed from G in logspace, whereas the tree representation used in [ADK07] (which in turn is related to the decomposition defined in [KCP82]) is defined as the reachable subgraph of a mangrove derived from G. This allows us to decide st-reachability in the tree $T(G,\pi)$ in logspace, an essential step to achieve our upper bound.

Another advantage of our construction comes with the usage of the kernel K as a canonical base. This makes it superfluous to cycle through all k-cliques of G (as in [ADK07]), leaving only the permutations of the vertices within K to enumerate.

Lemma 8. *For a k-tree G and a permutation π on the kernel K of G, $T(G,\pi)$ can be computed in* FL.

Proof. It is clear that the nodes and edges of $T(G)$ can be determined in logspace: First iterate over all subsets M of $V(G)$ of size k (this requires space $k \log n$) and output M as a node if M is a k-clique in G. Likewise, find and output all $(k+1)$-cliques M, each time adding edges to all $k+1$ many k-cliques contained in M. The (intermediate) result $T(G)$ cannot be stored due to space limitations, but it is possible to recompute it as needed (as long as only a constant number of operations is chained).

Next determine the kernel K of G (Fact 6) and think of all edges in $T(G)$ directed away from K. As described in Fact 5, the direction can be determined in logspace. It remains to compute the color $c(M)$ of each node $M \in V(T(G))$.

For each $v \in M$ calculate $c(v)$ by examining the unique path from M to K in $T(G)$ (the path can be found by following the unique incoming edge at each node). Store the length ℓ of the path and the position p_v where v was last found (this can be done in parallel for all $v \in M$). If $p_v = \ell$ (i. e. $v \in K$), then add the number $c(v) = \pi(v)$ to the color $c(M)$ of M. If $p_v < \ell$, add the number $c(v) = \ell - p_v + \|K\|$ to $c(M)$. The latter is correct, because by Lemma 2 the nodes containing v form a subtree of $T(G)$ and thus the node that is closest to K and contains v is on the path from K to M. □

We will need to compute a canonical labeling of $T(G,\pi)$. We observe the following generalization of the logspace tree canonization algorithm.

Lemma 9. *Lindell's algorithm [Lin92] can be extended to colored trees and to output not only a canonical form, but also a canonical labeling. This modification preserves the logarithmic space bound.*

Proof sketch. Colors can be handled by extending the *tree isomorphism order* defined in [Lin92] by using $color(s) < color(t)$ as additional condition (where s and t are the roots of the trees to compare). The canonical labeling can be computed by using a counter i initialized to 0: Instead of printing (the first letter of) the canon of a node v, increment i and print $v \mapsto i$. □

Next we show that the colored tree representations of isomorphic k-trees are also isomorphic, provided that the kernels are labeled accordingly.

Lemma 10. *Let $\phi \in S_n$ be an isomorphism between two k-trees G and H with $V(G) = V(H) = \{1, \ldots, n\}$ and $\ker(G) = \ker(H) = K$. Then ϕ (viewed as a mapping from $V(T(G))$ to $V(T(H))$) is an isomorphism between $T(G, \pi_1)$ and $T(H, \pi_2)$, provided that $\pi_1(u) = \pi_2(\phi(u))$ for all $u \in K$.*

Proof. It can be easily checked that any isomorphism between G and H is also an isomorphism between $T(G)$ and $T(H)$ that maps the kernel K of G to the kernel of H, which equals K by assumption. In order to show that the color of a node $M \in V(T(G, \pi_1))$ coincides with the color of $\phi(M) \in V(T(H, \pi_2))$, we prove the stronger claim that $c(v) = c(\phi(v))$ for all $v \in V(G)$. For $v \in K$, we have $c(\phi(v)) = \pi_2(\phi(v)) = \pi_1(v) = c(v)$ by assumption. Since ϕ must preserve the level of the vertices, it follows further for $v \in V(G) \setminus K$ that

$$c(\phi(v)) = l_H(\phi(v)) + \|K\| = l_G(v) + \|K\| = c(v)\,.$$

This completes the proof of the lemma. □

Conversely, the next lemma shows that from any isomorphic copy T of $T(G, \pi_1)$ we can easily derive an isomorphic copy G' of G. Moreover, any isomorphism ϕ between $T(G, \pi)$ and T can be efficiently converted into an isomorphism between G and G'.

Lemma 11. *Let G be a k-tree and let π be a permutation on the kernel K of G. Then from any colored tree T that is isomorphic to $T(G, \pi)$, an isomorphic copy G' of G can be computed in logspace. Further, it is possible to compute in logspace an isomorphism between G and G' from any given isomorphism between $T(G, \pi)$ and T.*

Proof sketch. Construct G' as follows. Let $V(G') = \{1, \ldots, n\}$, where n is k plus the number of $(k+1)$-clique nodes in T (we call $m \in V(T)$ a ***l*-clique node**, if $l = \|c(m)\|$). This is correct due to the one-to-one correspondence between the vertices $v \in V(G) \setminus K$ and the $(k+1)$-clique nodes $M_v \in T(G, \pi) \setminus \{K\}$. Next determine the center node z of T (see Lemma 6) and make $\{1, \ldots, k'\}$ a clique in G', where $k' = \|c(z)\|$. Further, for any non-center $(k+1)$-clique node $m \in V(T) \setminus \{z\}$, let $v(m)$ denote the corresponding vertex in $V(G')$ (to make this mapping unique, let $v(m)$ preserve the order of $(k+1)$-clique nodes in $V(T)$). Based on the color $c(m) = \{c_1, \ldots, c_{k+1}\}$ of m add the following edges to $E(G')$: For each $c_i \leq k'$ add an edge $\{c_i, v(m)\}$ and for each $c_i > k'$ with $c_i < c_{\max} = \max\{c_i \mid c_i \in c(m)\}$ add an edge $\{v(m), v(m')\}$, where m' is the $(c_i - k')$-th node on the path from z to m. This completes the construction of G'.

Now let ϕ be an isomorphism from $T(G, \pi)$ to T. Construct an isomorphism ϕ' from G to G' as follows. For $v \in K$, let $\phi'(v) = \pi(v)$, and for $v \notin K$, let $\phi'(v) = v(\phi(M_v))$. By induction on the level of v in G, it can be proven that this is indeed an isomorphism. Both constructions can easily be seen to be in logspace. □

Now we are ready to prove our main result.

Theorem 12. *Given a k-tree G with vertex set $V(G) = \{1, \ldots, n\}$ and kernel $K = \{1, \ldots, k'\}$, a canonical labeling $\psi_G \in S_n$ can be computed in* FL.

Proof. In order to compute ψ_G we iterate over all permutations $\pi \in S_{k'}$ and compute a canonical labeling $\psi_{T(G,\pi)}$ for the colored tree $T(G, \pi)$ using the algorithm from Lemma 9. Let π_1 be one of the permutations that give rise to the lexicographically smallest colored tree $\psi_{T(G,\pi_1)}(T(G, \pi_1))$. By applying Lemma 11, we can reconstruct from this tree an isomorphic copy $canon(G)$ of G together with an isomorphism ψ_G between G and $canon(G)$. By Lemmas 8 and 9, it is clear that ψ_G is computable in logspace.

It remains to show that the canonical labelings of any two isomorphic k-trees G and H map these graphs to the same canon $\psi_G(G) = \psi_H(H)$. To see this, let $\pi_1, \pi_2 \in S_{k'}$ be two permutations that give rise to the lexicographically smallest trees $\psi_{T(G,\pi_1)}(T(G, \pi_1))$ and $\psi_{T(H,\pi_2)}(T(H, \pi_2))$, respectively. Since by Lemma 10 for any tree $T(G, \pi_1)$ that can be derived from G via some permutation π_1 there is an isomorphic tree $T(H, \pi_2)$ that can be derived from H via some permutation π_2 (and vice versa), it follows that $\psi_{T(G,\pi_1)}(T(G, \pi_1))$ and $\psi_{T(H,\pi_2)}(T(H, \pi_2))$ are equal, implying that $canon(G) = canon(H)$. □

We note that the above construction can be extended to colored k-trees as follows. Let $\zeta\colon V(G) \to C$ be a vertex coloring of G. Modify the coloring of $T(G, \pi)$ (cf. Definition 7) by replacing $c(v)$ with the pair $c'(v) = (c(v), \zeta(v))$.

Theorem 12 immediately yields the following corollaries.

Corollary 13. *For any fixed k, k-tree canonization is in* FL.

Corollary 14. *For any fixed k, k-tree isomorphism is* L*-complete.*

The L-hardness can be seen by a reduction from the isomorphism problem for trees in pointer notation, which is known to be L-hard [JKM$^+$03]. The reduction transforms a tree T into a k-tree $\boldsymbol{E_k(T)}$ by adding a $(k-1)$-clique C and connecting C to all nodes in $V(T)$ (cf. Fig. 2). It can easily be seen that $E_k(T)$ is a k-tree and that $T_1 \cong T_2 \Leftrightarrow E_k(T_1) \cong E_k(T_2)$.

We note that fixing k is essential, as the isomorphism problem for the class of all k-trees, $k \in \mathbb{N}^+$, is isomorphism complete [KCP82] and thereby unlikely to be decidable in polynomial time.

Furthermore, there is a standard Turing reduction of the automorphism group problem (i.e., computing a generating set of the automorphism group of a given graph) to the search version of GI for colored graphs (cf. [Hof82,KST93]). It is not hard to see that this reduction can be performed in logspace.

Corollary 15. *For any fixed k, computing a generating set of the automorphism group of a given k-tree, and hence computing a canonical labeling coset for a given k-tree is in* FL.

Corollary 16. *For any fixed k, the k-tree automorphism problem (i. e., deciding whether a given k-tree has a non-trivial automorphism) is* L*-complete.*

Observe that the mapping $T \mapsto E_k(T)$ does not provide a correct reduction of tree automorphism (which is L-complete [JKM+03]) to k-tree automorphism, as the vertices within the newly added clique can always be permuted without changing the graph. To sidestep this difficulty, we use a transformation E'_k that preserves rigidity. Let T be a rooted tree with $n = \|V(T)\|$ and root $r \in V(T)$. Then $\boldsymbol{E'_k(T, r)}$ is defined as follows (cf. Fig. 2):

$$\begin{aligned} V(E'_k(T,r)) &= V(T) \cup \big\{u_i \;\big|\; 1 \le i \le k+n\big\} \\ E(E'_k(T,r)) &= E(T) \cup \big\{\{v, u_i\} \;\big|\; v \in V(T), 1 \le i \le k-1\big\} \\ &\quad \cup \big\{\{r, u_k\}\big\} \cup \big\{\{u_i, u_j\} \;\big|\; 1 \le i < j \le k+n, j-i \le k\big\} \end{aligned}$$

It is easy to see that $E'_k(T, r)$ is a k-tree and that any non-trivial automorphism of (T, r) induces a non-trivial automorphism of $E'_k(T, r)$. To see that $E'_k(T, r)$ is rigid whenever (T, r) is rigid, assume $n > k$ (all smaller trees can be hard-coded in the reduction). Any automorphism of $E'_k(T, r)$ must fix all newly added vertices u_i: Each of the vertices u_i, $1 \le i \le k-1$, is uniquely determined by its degree $n + k - 2 + i$ (unless r is connected to all vertices of T, but then T is a star and not rigid anyway). The vertices u_i, $k+1 \le i \le k+n$, are the only ones not adjacent to u_1 and uniquely identified by the structure of $E'_k(T, r)$ (examine the tree representation $T(E'_k(T, r))$ to see this). Finally, the vertex u_k is unique among the remaining ones by the shortest distance to u_{k+n}.

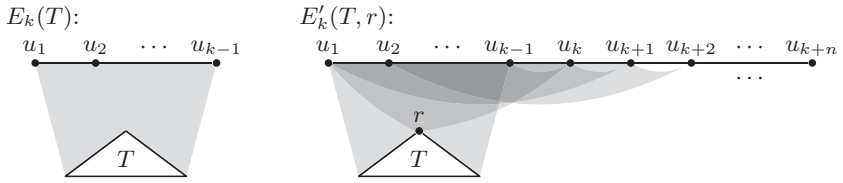

Fig. 2. The transformations $E_k(T)$ and $E'_k(T, r)$

4 Complete problems for logspace

In this section we prove some additional completeness results for logspace that are related to our main result. The hardness is under DLOGTIME-uniform AC^0-reductions. We first recall that ORD is L-complete, where **ORD** is the problem of deciding for a directed line graph P and two vertices $s, t \in V(P)$ if there is a path from s to t [Ete97].

In Lemma 6 we have seen that the center of an undirected tree can be computed in FL. We now show that the decision variant is hard for L even when restricted to paths.

Theorem 17. *Given an undirected path P and a vertex $c \in V(P)$, it is* L*-hard to decide if c belongs to the center of P.*

This implies the L-hardness of the following problem: Given a k-tree (or k-path) G and a vertex $c \in V(G)$, decide whether c belongs to the kernel of G. The reduction for this is $(P, c) \mapsto (E_k(P), c)$, where E_k is as defined above.

Proof. We reduce from ORD using $(P, s, t) \mapsto (P', n)$ as reduction, where

$$\begin{aligned} V(P') &= V(P) \cup \big\{ i' \bigm| i \in V(P) \big\} \cup \{s''\} \\ E(P') &= \big\{ \{i, j\} \bigm| (i, j) \in E(P) \wedge j \neq t \big\} \cup \big\{ \{n, n'\} \big\} \\ &\quad \cup \big\{ \{i', j'\} \bigm| (i, j) \in E(P) \wedge j \notin \{s, t\} \big\} \\ &\quad \cup \big\{ \{i', s''\} \bigm| (i, s) \in E(P) \big\} \cup \{\{s'', s'\}\} \end{aligned}$$

and n is the vertex without successor in P. P' is the undirected path that consists of two copies of P that are twisted before t, connected at their ends and have the second copy of s duplicated (cf. Fig. 3). If s precedes t in P (left side), then n is the center of P', but if t precedes s then n' is the center of P' (right side). □

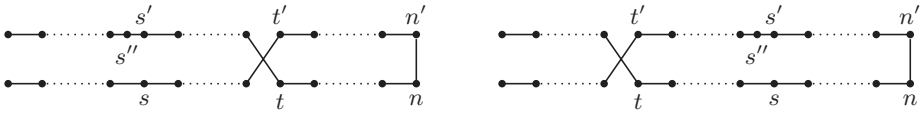

Fig. 3. The reduction of ORD to verifying the center

Finally, we examine two problems related to the structure of k-trees. Let G be a graph. A vertex $v \in V(G)$ is called **simplicial** in G, if its neighborhood induces a clique. A bijective mapping $\sigma\colon \{1, \ldots, \|V(G)\|\} \to V(G)$ of the vertices of G is called **perfect elimination order (PEO)**, if for all i, $\sigma(i)$ is simplicial in $G - \bigcup_{j<i}\{\sigma(j)\}$. Note that a graph can have several perfect elimination orders, so finding a PEO is not a functional but a search problem. It is well-known that a graph has a PEO if and only if it is chordal. As k-trees are a subclass of chordal graphs, each k-tree has a PEO.

A related problem is the **fast reordering problem (FRP)** which is defined in [GSS02] as a preprocessing step for parallel algorithms. It consists of finding a sequence of sets $R_0, \ldots, R_k \subseteq V(G)$, such that each R_i is a maximal independent set of simplicial vertices of $G - \bigcup_{j<i} R_j$ and that $G - \bigcup_{0 \le j \le k} R_j$ is a clique. For general chordal graphs there can be several such sequences, but for k-trees this sequence is unique and the remaining clique is the kernel. In [GSS02] it was shown that if the input graphs are restricted to k-trees, the FRP can be solved in NC. We improve this and show logspace completeness for both problems:

Theorem 18. *For k-trees (k fixed), it is logspace complete to find a perfect elimination order and to solve the fast reordering problem.*

Proof. We first show FRP $\in$ FL: Let G be a k-tree. We compute the level $l_G(v)$ for each $v \in V(G)$ (cf. Definition 7). As observed in Lemma 8, this is possible in

logspace. Let $l_{\max} = \max\{l_G(v) \mid v \in V(G)\}$. Output $R_i := \{v \in V(G) \mid l_G(v) = l_{\max} - 2i\}$ for $i = 0, \ldots, \lceil l_{\max}/2 \rceil - 1$. The correctness follows from the structure of $T(G)$.

Next, we note that a perfect elimination order can be efficiently computed when a solution to the FRP is known (i. e. finding a PEO reduces to solving the FRP): Take the members of the R_i in ascending order (first those from R_0, then those from R_1 and so on up to R_k) and finally those from $\ker(G) = V(G) \setminus \bigcup_i R_i$. No matter which order is chosen within the R_i and the kernel, the result is a PEO, as each R_i is independent and $\ker(G)$ is a clique.

Finally we show that finding a perfect elimination order is hard for logspace even for paths. The result for k-trees (and k-paths) can again be obtained using the construction of E_k given above. We solve an ORD instance (P, s, t) in DLOGTIME-uniform AC^0 with a single oracle gate for computing a PEO of the path P' given by

$$\begin{aligned} V(P') &= V(P) \cup \left\{ i' \;\middle|\; i \in V(P) \setminus \{n\} \right\} \\ E(P') &= \left\{ \{i, j\} \;\middle|\; (i, j) \in E(P) \right\} \\ &\quad \cup \left\{ \{i', j'\} \;\middle|\; (i, j) \in E(P), j \neq n \right\} \cup \left\{ \{i', n\} \;\middle|\; (i, n) \in E(P) \right\} \end{aligned}$$

where n is the vertex in P without successor. We claim that for any PEO σ of P' (where p_i is a shorthand for the position $\sigma^{-1}(i)$ of a vertex in σ):

$$(P, s, t) \in \text{ORD} \Leftrightarrow p_s \leq p_t \leq p_n \vee p_{s'} \leq p_{t'} \leq p_n$$

If $(P, s, t) \notin \text{ORD}$, then s is between t and n, and s' is between t' and n in P' (right side in Fig. 4). Thus σ cannot satisfy both $p_s \leq p_t$ and $p_{s'} \leq p_{t'}$. If $(P, s, t) \in \text{ORD}$ (left side of Fig. 4), n does not become simplicial until at least one copy of P is completely removed. Similarly, if the first copy is completely removed before n, t does not become simplicial before s is removed; and if the second copy is completely removed before n, t' does not become simplicial before s' is removed. □

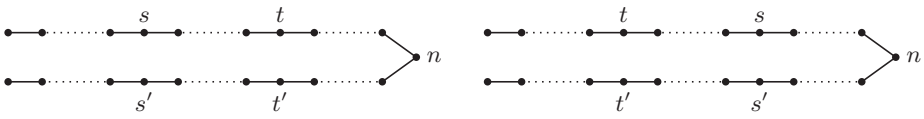

Fig. 4. The reduction of ORD to finding a PEO

References

[ADK07] Arvind, V., Das, B., Köbler, J.: The space complexity of k-tree isomorphism. In: Algorithms and Computation. Proceedings of 18th ISAAC. LNCS 4853, Springer (2007) 822–833

[ADK08] Arvind, V., Das, B., Köbler, J.: A logspace algorithm for partial 2-tree canonization. In: Proceedings of the 3rd International Computer Science Symposium in Russia (CSR). LNCS 5010, Springer (2008) 40–51

[AHU74] Aho, A., Hopcroft, J., Ullman, J.: The design and analysis of computer algorithms. Addison-Wesley (1974)

[AJ93] Àlvarez, C., Jenner, B.: A very hard log-space counting class. Theoretical Computer Science **107**(1) (1993) 3–30

[AK06] Arvind, V., Kurur, P.P.: Graph isomorphism is in SPP. Information and Computation **204**(5) (2006) 835–852

[AM04] Allender, E., Mahajan, M.: The complexity of planarity testing. Information and Computation **139**(1) (February 2004)

[BDH+92] Buntrock, G., Damm, C., Hertrampf, U., Meinel, C.: Structure and importance of logspace-MOD classes. Mathematical Systems Theory **25** (1992) 223–237

[Coo85] Cook, S.A.: A taxonomy of problems with fast parallel algorithms. Information and Control **64** (1985) 2–22

[DLN+08] Datta, S., Limaye, N., Nimbhorkar, P., Thierauf, T., Wagner, F.: A log-space algorithm for canonization of planar graphs. CoRR (2008) http://arxiv.org/abs/0809.2319

[Ete97] Etessami, K.: Counting quantifiers, successor relations, and logarithmic space. Journal of Computer and System Sciences **54**(3) (1997) 400–411

[GS86] Goldwasser, S., Sipser, M.: Private coins versus public coins in interactive proof systems. In: Randomness and Computation. Volume 5 of Advances in Computing Research. JAI Press (1989) 73–90

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

[Hof82] Hoffmann, C.: Group-Theoretic Algorithms and Graph Isomorphism. LNCS 136. Springer (1982)

[JKM+03] Jenner, B., Köbler, J., McKenzie, P., Torán, J.: Completeness results for graph isomorphism. Journal of Computer and System Sciences **66** (2003) 549–566

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

[KST93] Köbler, J., Schöning, U., Torán, J.: The Graph Isomorphism Problem: Its Structural Complexity. Progress in Theoretical Computer Science. Birkhäuser, Boston (1993)

[Lin92] Lindell, S.: A logspace algorithm for tree canonization. extended abstract. In: Proceedings of the 24th STOC, New York, ACM (1992) 400–404

[MR91] Miller, G., Reif, J.: Parallel tree contraction part 2: further applications. SIAM Journal on Computing **20** (1991) 1128–1147

[Rei05] Reingold, O.: Undirected st-connectivity in log-space. In: Proceedings of the 37th STOC, New York, ACM (2005) 376–385

[Sch88] Schöning, U.: Graph isomorphism is in the low hierarchy. Journal of Computer and System Sciences **37** (1988) 312–323

[Tor04] Torán, J.: On the hardness of graph isomorphism. SIAM Journal on Computing **33**(5) (2004) 1093–1108