Electronic Colloquium on Computational Complexity, Revision 1 of Report No. 53 (2009)

The Isomorphism Problem for k-Trees is Complete for Logspace

V. Arvind[1], Bireswar Das[1], Johannes Köbler[2], and Sebastian Kuhnert[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, kuhnert}@informatik.hu-berlin.de

Abstract. We show that, for k constant, k-tree isomorphism can be decided in logarithmic space by giving a logspace canonical labeling algorithm. The algorithm computes a unique tree decomposition, uses colors to fully encode the structure of the original graph in the decomposition tree and invokes Lindell's tree canonization algorithm. As a consequence, the isomorphism, the automorphism, as well as the canonization problem for k-trees are all complete for deterministic logspace. Completeness for logspace holds even for simple structural properties of k-trees. We also show that isomorphism of all k-trees, $k \geq 1$, is fixed parameter tractable with respect to k by giving an algorithm running in time $\mathcal{O}((k+2)! \cdot m)$, where m is the number of edges in the input graph.
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

ISSN 1433-8092

(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. Recently, it has been shown that even the isomorphism problem for all planar graphs is in logspace [DLN$^+$09] (in fact, excluding one of K_5 or $K_{3,3}$ as minor is sufficient [DNT$^+$09]). 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.

Our motivation for studying the isomorphism of k-trees is that graphs of tree width k coincide with partial k-trees, i. e. subgraphs of k-trees. Isomorphism of partial k-trees was known to be in TC^1 [GV06] and canonization in TC^2 [KV08]. We hope that our result can be generalized from k-trees to partial k-trees in the future. Das, Torán and Wagner already used some of our techniques to put isomorphism of tree distance width k graphs in L [DTW10], a subclass of tree width k graphs that is incomparable to k-trees. They also reduce isomorphism of decomposed bounded tree width graphs to isomorphism of bounded tree distance width graphs. The remaining obstacle to put isomorphism of partial k-trees in L is thus the computation of compatible tree decompositions in logspace, compatible meaning that if two partial k-trees are isomorphic then there should be an isomorphism that maps one decomposition to the other. Recently, Elberfeld, Jakoby and Tantau showed how to compute a tree decomposition in logspace [EJT10] (previously, LogCFL was known [Wan94]), using and improving the mangrove technique we employed in our intermediate StUL result for k-tree isomorphism [ADK07]. However, the constructed tree decomposition strongly depends on the order of the input vertices and computing compatible tree decompositions seems to require new ideas.

In this article we show that the isomorphism problem for k-trees is in logspace for each fixed $k \in \mathbb{N}^+$, matching the lower bound. This combines two conference papers that improved the previously known upper bound of TC^1 [GV06] first to StUL [ADK07] and then to L [KK09]. 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 a canonical labeling for the colored $T(G)$ and derive a canonical labeling for the k-tree G.

This main result is presented in Section 3. In Section 4 we show that several simple structural properties of k-trees that can be computed using our tree representation are also hard for logspace. In Section 5 we consider a variant of our algorithm to prove that isomorphism of k-trees is fixed parameter tractable with respect to k.

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 $V(G)$ and $E(G)$ to denote its vertex and edge sets, respectively. We define the following notations for subgraphs of G. For $M \subseteq V(G)$, $G[M]$ denotes the subgraph of G induced by M and we use $G - M$ as a shorthand for $G[V(G) \setminus M]$.

Given a graph G and two vertices $u, v \in V(G)$, the *distance* $d_G(u, v)$ 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., $ecc_G(v) = \max\{d_G(u, v) \mid u \in V(G)\}$. 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*; they form a group which we denote by $\mathrm{Aut}(G)$. 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.

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

Definition 3.1. [Klo94] *Let $G = (V, E)$ be a graph. A subset $S \subset V$ is a* vertex separator *for two nonadjacent vertices $u, v \in V$, if u and v are in different connected components of $G - S$. A vertex separator S for u and v is called* minimal*, if no proper subset of S is a vertex separator for u and v. A subset $S \subset V$ is a* minimal vertex separator *if S is a minimal vertex separator for some pair of vertices $u, v \in V$.*

Lemma 3.2. [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 3.2 can be checked in logspace. Hence, from now on we can assume that the input graph G is a k-tree.

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

Definition 3.3. *For a k-tree G, its* tree representation $T(G)$ *is defined by*

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

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

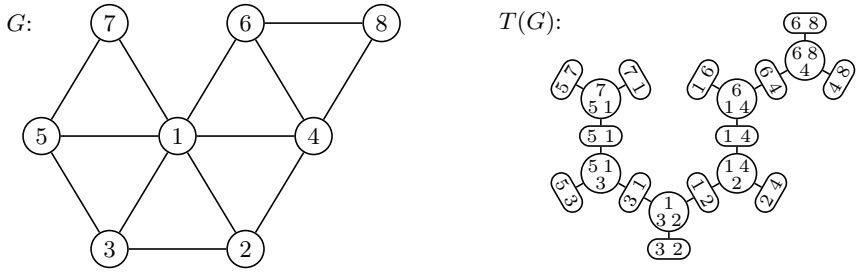

Fig. 1. A 2-tree G and its tree representation $T(G)$

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 3.4. *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 3.5. *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 3.6. *The clique corresponding to the center node of $T(G)$ is called* kernel *of G and denoted* $\ker(G)$.

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 3.7. *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 3.8. *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 3.7). 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 3.9. *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*

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

the level *of v in G. Further, for any permutation $\pi \in S_{k'}$, let $T(G, \pi)$ 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*

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

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 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 3.10. *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 3.8) and think of all edges in $T(G)$ directed away from K. As described in the proof of Fact 3.7, 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 3.4 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 3.11. *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 3.12. *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 3.13. *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 $m \in V(T)$ an *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 3.8) 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 proved 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 3.14. *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 3.11. 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 3.13, 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 3.10 and 3.11, 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 3.12 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 3.9) by replacing $c(v)$ with the pair $c'(v) = (c(v), \zeta(v))$.

Theorem 3.14 immediately yields the following corollaries.

Corollary 3.15. *For any fixed k, k-tree canonization is in* FL.

Corollary 3.16. *For any fixed k, k-tree (and k-path) isomorphism is* L*-complete.*

We delay the hardness part to Proposition 4.3.

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; a similar reduction exists for counting the number of automorphisms (cf. [Hof82,KST93]). It is not hard to see that these reductions can be performed in logspace.

Corollary 3.17. *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 3.18. *For any fixed k, computing the number of automorphism of a given k-tree is in* FL.

Corollary 3.19. *For any fixed k, the k-tree (and k-path) automorphism problem (i. e., deciding whether a given k-tree has a non-trivial automorphism) is* L*-complete.*

We postpone the proof of the hardness to Proposition 4.2.

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 path P and two vertices $s, t \in V(P)$ if there is a path from s to t [Ete97].

In Lemma 3.8 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 4.1. *Given an undirected path P and a vertex $c \in V(P)$, it is L-complete to decide if c belongs to the center of P. The hardness also holds if P is required to have odd length.*

We will call this problem PATHCENTER. This theorem 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 transforms a path P (resp. tree T) into a k-path $E_k(P)$ (resp. a k-tree $E_k(T)$) by adding a $(k-1)$-clique C and connecting C to all nodes in $V(P)$ (resp. $V(T)$) (cf. Fig. 2).

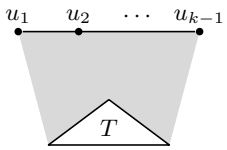

Fig. 2. The transformation $E_k(T)$

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

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

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). □

Using the hardness of PATHCENTER, we now prove the hardness of the automorphism and isomorphism problems for k-paths.

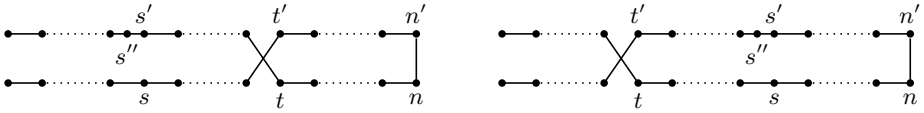

Fig. 3. The reduction of ORD to PATHCENTER

Proposition 4.2. *For any fixed k, the automorphism problem for k-paths (and thereby k-trees) is* L*-hard.*

Proof. We reduce from PATHCENTER using the function $(P, c) \mapsto P'$ where the neighbors of c are n_1 and n_2 and

$$\begin{aligned} V(P') &= V(P) \cup \{c_1, c_2\} \\ E(P') &= \big\{\{u, v\} \;\big|\; 1 \le d_P(u, v) \le k \text{ for } u, v \in V(P)\big\} \\ &\quad \cup \big\{\{u, c_i\} \;\big|\; 0 \le d_{P-\{n_i\}}(u, c) < k \text{ for } u \in V(P), i \in \{1, 2\}\big\} \end{aligned}$$

We assume that c has distance more than k to both ends, as otherwise the problem is trivial. It is easy to see that P' is a k-path: The first k vertices on the path are the base. As the following vertices are added, the support is always a set of k consecutive nodes on the path, with the additional vertices c_i being added after c enters the support and before c leaves the support, respectively. It is obvious that $P' - \{c_1, c_2\}$ has the nontrivial automorphism that maps the ith vertex on the path to the $(n-i)$-th one, but is otherwise rigid (all other pairs of vertices have different eccentricity). This automorphism can be extended to all of P' by exchanging c_1 and c_2 if and only if c is the center of P. □

Proposition 4.3. *For any fixed k, the isomorphism problem for k-paths (and thereby k-trees) is* L*-hard.*

Proof. Again we reduce from PATHCENTER. Modifying the previous construction, we reduce $(P, c) \mapsto (P_1, P_2)$ where the neighbors of c are n_1 and n_2, the ends of P are v_1 and v_2 and

$$\begin{aligned} V(P_j) &= V(P) \cup \{c_1, c_2, e_1, e_2\} \\ E(P_j) &= \big\{\{u, v\} \;\big|\; 1 \le d_P(u, v) \le k \text{ for } u, v \in V(P)\big\} \\ &\quad \cup \big\{\{u, c_i\} \;\big|\; 0 \le d_{P-\{n_i\}}(u, c) < k \text{ for } u \in V(P), i \in \{1, 2\}\big\} \\ &\quad \cup \big\{\{u, e_i\} \;\big|\; 0 \le d_P(u, v_j) < k \text{ for } u \in V(P), i \in \{1, 2\}\big\} \end{aligned}$$

We assume that c has distance more than $k+1$ to both ends. If c is the center of P, the function that maps the ith vertex of P to the $(n-i)$-th, exchanges c_1 and c_2 and maps e_1, e_2 to themselves is an isomorphism from P_1 to P_2. Conversely, if there is such an isomorphism then the e_i force the original path vertices to be mirrored and the c_i ensure that c is 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 $\sigma\colon \{1,\dots,\|V(G)\|\} \to V(G)$ 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 $R_0,\dots,R_k \subseteq V(G)$, 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 4.4. *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 $\in \mathsf{FL}$: Let G be a k-tree. We compute the level $l_G(v)$ for each $v \in V(G)$ (cf. Definition 3.9). As observed in Lemma 3.10, 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 $i = 0,\dots,\lceil l_{\max}/2\rceil - 1$. 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 be obtained using the construction of E_k given above. We solve an Ord instance (P,s,t) in $\mathsf{DLOGTIME}$-uniform AC^0 with a single oracle gate for computing a PEO of the path P' given by

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

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):

$$(P,s,t) \in \textsc{Ord} \Leftrightarrow p_s \le p_t \le p_n \vee p_{s'} \le p_{t'} \le p_n$$

If $(P,s,t) \notin \textsc{Ord}$, 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 $p_s \le p_t$ and $p_{s'} \le p_{t'}$. If

$(P, s, t) \in$ 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. □

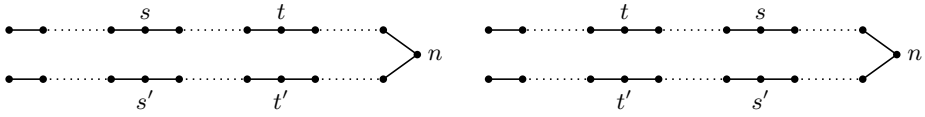

Fig. 4. The reduction of ORD to finding a PEO

5 Fixed parameter tractability

A problem is called *fixed parameter tractable* with respect to some parameter k, if it is solved by an algorithm in time $f(k)n^{\mathcal{O}(1)}$, where $f(k)$ can be an arbitrary function not depending on the input size n.

Theorem 5.1. *The isomorphism problem of the class of all k-trees, $k \in \mathbb{N}^+$, is fixed parameter tractable with respect to k: There is an algorithm that runs in time $\mathcal{O}((k+2)! \cdot n)$, where n is the number of nodes in the input graph.*

As noted before, this class is isomorphism complete [KCP82]. The fixed parameter tractability is no immediate consequence of the logspace algorithm for fixed k, as there are already n^k choices for k-cliques that are searched by brute force.

Proof. First note that any k-tree has less than $k \cdot n$ edges, and that k can easily be obtained from the input graph, as all inclusion-maximal cliques have size $k+1$. The tree representation $T(G)$ of the input graph G can be computed in time $\mathcal{O}(k \cdot n)$ by iteratively removing simplicial vertices [KCP82]. In this process it can also be checked that G is indeed a k-tree. Additionally, for each of the $k'! \leq (k+1)!$ permutations $\pi \in S_{k'}$, the colored tree $T(G, \pi)$ can be constructed in linear time by first finding the center of $T(G)$, computing all distances from the center to the other tree nodes using breadth first search and finally outputting the color sets. Finally, tree isomorphism can be decided in linear time [AHU74, p. 84]. □

Previously, Toda has proved that isomorphism of chordal graphs whose so-called s-components are of size k is fixed parameter tractable [Tod06]. This class is a generalization of chordal graphs of bounded clique size k and thus includes $(k-1)$-trees. However, the time bound obtained there is far from our $\mathcal{O}((k+2)! \cdot n)$. Relatedly, Yamazaki, Bodlaender, de Fluiter, and Thilikos showed how to check isomorphism for graphs of rooted tree distance width k in time $\mathcal{O}(k!^2 k^2 n^2)$ [YBF+00]. This graph class is incomparable to k-trees but like those a subclass of tree width k graphs.

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

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

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

[DTW10] Das, B., Torán, J., Wagner, F.: Restricted space algorithms for isomorphism on bounded treewidth graphs. In *STACS* (2010) 227–238

[DLN$^+$09] Datta, S., Limaye, N., Nimbhorkar, P., Thierauf, T., Wagner, F.: Planar graph isomorphism is in log-space. In *CCC* (2009) 203–214

[DNT$^+$09] Datta, S., Nimbhorkar, P., Thierauf, T., and Wagner, F.: Graph isomorphism for $K_{3,3}$-free and K_5-free graphs is in log-space. In *FSTTCS* (2009) 145–156

[EJT10] Elberfeld, M., Jakoby, A., Tantau, T.: Logspace versions of the theorems of Bodlaender and Courcelle. ECCC TR10-062 (2010) http://eccc.uni-trier.de/report/2010/062/

[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

[GV06] Grohe, M., Verbitsky, O.: Testing graph isomorphism in parallel by playing a game. In: Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Proceedings. LNCS 4051. Springer (2006) 3–14

[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

[KK09] Köbler, J., Kuhnert, S.: The isomorphism problem for k-trees is complete for logspace. In: Proceedings of 34th International Symposium Mathematical Foundations of Computer Science (MFCS). LNCS 5734. Springer (2009) 537–548.

[Klo94] Kloks, T.: Treewidth. Computations and approximations. LNCS 842. Springer (1994)

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

[KV08] Köbler, J., Verbitsky, O.: From invariants to canonization in parallel. In: Proceedings of 3rd Computer Science in Russia (CSR'08). LNCS 5010, Springer (2008) 216–227

[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

[Tod06] Toda, S.: Computing automorphism groups of chordal graphs whose simplicial components are of small size. IEICE Transactions on Information and Systems **E89-D**(8) (2006) 2388–2401

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

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

[YBF+00] Yamazaki, K., Bodlaender, H.L., de Fluiter, B., and Thilikos, D.M.: Isomorphism for graphs of bounded distance width. Algorithmica **24**(2) (1999) 105–127

ECCC ISSN 1433-8092

http://eccc.hpi-web.de