Information and Computation 217 (2012) 1–11

Contents lists available at SciVerse ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

The isomorphism problem for k-trees is complete for logspace

V. Arvind [a], Bireswar Das [b], Johannes Köbler [c], Sebastian Kuhnert [c,*,1]

[a] The Institute of Mathematical Sciences, Chennai 600 113, India
[b] Indian Institute of Technology Gandhinagar, Ahmedabad 382 424, India
[c] Humboldt-Universität, Institut für Informatik, 10099 Berlin, Germany

ARTICLE INFO

Article history:
Received 16 July 2010
Revised 13 February 2012
Available online 27 April 2012

Keywords:
Graph isomorphism
Graph canonization
k-Trees
Space complexity
Logspace completeness

ABSTRACT

We show that, for k constant, k-tree isomorphism can be decided in logarithmic space by giving an $\mathcal{O}(k \log n)$ space 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 a variant of our canonical labeling algorithm runs in time $\mathcal{O}((k+1)!\, n)$, where n is the number of vertices, yielding the fastest known FPT algorithm for k-tree isomorphism.

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 [17,31] and in SPP [7], providing strong evidence that GI is not NP-complete. On the other hand, the strongest known hardness result due to Torán [33] says that GI is hard for the class DET (cf. [10]). DET is a subclass of NC^2 (even of TC^1) and contains NL as well as all logspace counting classes [3,9].

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 [1]. In 1991, an NC algorithm was developed by Miller and Reif [27], and one year later, Lindell [26] obtained an L upper bound. On the other hand, in [20] it is shown that tree isomorphism is L-hard (provided that the trees are given in pointer notation). In [6], Lindell's logspace 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 [12] (in fact, excluding one of K_5 or $K_{3,3}$ as minor is sufficient [13]). 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 [29]. Our result does not depend on this, yielding a comparatively simple algorithm.

* Corresponding author.
E-mail addresses: arvind@imsc.res.in (V. Arvind), bireswar@iitgn.ac.in (B. Das), koebler@informatik.hu-berlin.de (J. Köbler), kuhnert@informatik.hu-berlin.de (S. Kuhnert).

[1] Partially supported by DFG grant KO 1053/7-1.

0890-5401/$ – see front matter © 2012 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.ic.2012.04.002

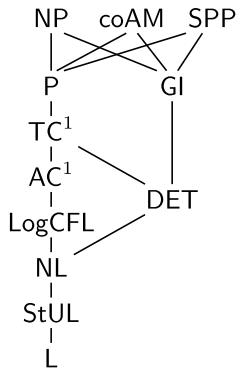

Fig. 1. Known inclusions between the mentioned complexity classes.

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 [18] first to StUL [5] and then to L [23]. 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 [14] 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 all $(k+1)$-cliques and some k-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 kernel vertices. In order to extend such a labeling 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. (Note that this differs from *valid colorings* using few colors.) Finally, we apply a variant of Lindell's algorithm [26] to compute a canonical labeling for the colored $T(G)$ and derive a canonical labeling for the k-tree G.

Fig. 1 shows the known inclusions between the mentioned complexity classes, where GI is the class of all problems that can be reduced to graph isomorphism in polynomial time.

As the k-tree isomorphism problem, for unbounded k, is graph isomorphism complete [21], it makes sense to study the fixed parameter tractability of this problem. A problem is called *fixed parameter tractable* (*FPT*) 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. We show that our k-tree canonization algorithm can also be implemented in $\mathcal{O}((k+1)!\,n)$ time, yielding the fastest known FPT isomorphism algorithm for this class. Previously, Klawe et al. gave an FPT isomorphism algorithm for general chordal graphs with maximum clique size $k+1$, which runs in $\mathcal{O}((k+1)!^2 n^3)$ time [21]. Nagoya improved this to $\mathcal{O}((k+1)!\,n^3)$ [28]. Toda gave an isomorphism algorithm that is FPT in the maximum size s of simplicial components, using $\mathcal{O}((s!\,n)^{\mathcal{O}(1)})$ time [32]. While the exact running time of this algorithm is hard to analyze, the parameter s can be smaller than $k+1$ by a factor of up to $\Theta(n)$ (and is never larger). We remark that the kernel of a k-tree is always a simplicial component, implying that $k \leqslant s \leqslant k+1$ for all k-trees. Relatedly, Yamazaki et al. showed how to check isomorphism for graphs of rooted tree distance width k in time $\mathcal{O}(k!^2 k^2 n^2)$ [36]. This graph class contains non-chordal graphs and is incomparable to k-trees, but like k-trees it is a subclass of treewidth k graphs.

In Section 3, we describe our algorithm and prove its correctness; in Sections 4 and 5 we give implementations in logspace and FPT, respectively. In Section 6, we show that several simple structural properties of k-trees that can be computed using our tree representation are also hard for logspace.

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, where N is the input size. FL is the class of all functions computable by Turing machines that additionally have a write-only output tape. Note that FL is closed under composition: To compute $f(g(x))$ for $f, g \in \mathsf{FL}$, simulate the Turing machine for f and keep track of the position of its input head. Every time this simulation needs a character from f's input tape, simulate the Turing machine for g on input x until it outputs the required character. Note also that g can first output a copy of its input x, so we can assume that f has access to both x and $g(x)$. This