A Logspace Algorithm for Partial 2-Tree Canonization

Vikraman 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 partial 2-tree canonization, and hence isomorphism testing for partial 2-trees, is in deterministic logspace. Our algorithm involves two steps: (a) We exploit the "tree of cycles" property of biconnected partial 2-trees to canonize them in logspace. (b) We analyze Lindell's tree canonization algorithm and show that canonizing general partial 2-trees is also in logspace, using the algorithm to canonize biconnected partial 2-trees.

1 Introduction

Computing canonical forms for graphs is a fundamental algorithmic problem. The problem is closely related to the graph isomorphism problem GI. Let $\mathcal{G}$ be a class of (encodings of) graphs closed under isomorphism. 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 the graph $f(G)$ the *canonical form* of G (w.r.t. f) and denote it by *canon*(G). E.g. we could define $f(G)$ as the lexicographically least graph isomorphic to G. This canonizing function is computable in $\mathsf{FP}^{\mathsf{NP}}$ by prefix search, but it is NP-hard [8,18]. Whether there is *some* canonizing function for graphs that is polynomial-time computable is a long-standing open question. No better bound than $\mathsf{FP}^{\mathsf{NP}}$ is known for general graphs (for any canonizing function). Clearly, GI is polynomial-time reducible to graph canonization. It is an open question if the converse reduction holds.

The seminal paper of Babai and Luks [8] takes a general group-theoretic approach to graph canonization. However, combinatorial methods have worked well in various special cases. For example, for random graphs the Weisfeiler-Lehman method [7,6] produces a canonical form with high probability. From a complexity-theoretical viewpoint, a key result [17,14] we recall is that tree canonization is complete for deterministic logspace.[1] Indeed, Lindell's logspace upper bound for tree canonization is our main motivation for the present paper

[1] Provided that the tree is given in the pointer notation; using the parenthesis notation the problem is NC^1-complete (see [14]).

and, more generally, our motivation for studying the space bounded complexity of Graph Isomorphism for partial k-trees.

The recent TC^1 upper bound for isomorphism of partial k-trees by Grohe and Verbitsky [10] raises the question about a tight complexity-theoretic classification of the problem. In the present paper, we give a deterministic logspace algorithm for canonizing partial 2-trees. The algorithm is based on ideas from Lindell's tree canonization algorithm and uses the combinatorial characterizations of partial 2-trees. This tightly classifies partial 2-tree isomorphism. The class of partial 2-trees coincides with the class of series-parallel graphs and contains all outerplanar graphs. Thus, we obtain logspace canonization algorithms for these graph classes. Furthermore, partial 2-trees are planar graphs. However, we do not know if planar graph isomorphism is in logspace (or in NL). In that direction, recently Thierauf and Wagner gave a $\mathsf{UL} \cap \mathsf{co\text{-}UL}$ upper bound for planar *3-connected* graph isomorphism [20]. They also provide an NL algorithm for oriented graphs.

We note that Arnborg and Proskurowski gave a linear time and quadratic space[2] canonization algorithm for both partial 2-trees and partial 3-trees [5]. Their algorithm is based on a graph reduction technique and uses labels to canonically record the reductions. However, this technique does not appear useful for showing a logspace upper bound.

For partial k-trees in general, we don't know of any better bound than TC^1. We do not know of any hardness result that would indicate that partial k-tree isomorphism is not in deterministic logspace. However, the TC^1 upper bound suggests that we can put perhaps the problem in a natural complexity class contained in TC^1 like LOGCFL or DET, or in the logspace counting hierarchy [3,2]. Recently, in [1] we have shown for *full* k-trees that isomorphism testing is in the strongly unambiguous logspace class $\mathsf{StUL} \subseteq \mathsf{UL}$. The bound follows from an $\mathsf{FL}^{\mathsf{StUL}}$ canonizing algorithm for full k-trees.

Due to lack of space, some proofs are omitted from this extended abstract.

1.1 Preliminaries

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

By graphs we mean finite simple graphs. For a graph $G = (V, E)$, let $V(G)$ denote its vertex set V and $E(G)$ denote its edge set E. For a vertex $v \in V(G)$, the set $\{w \in V(G) \mid \{v, w\} \in E(G)\}$ of all *neighbors* of v 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 $E(G[U]) = \{e \in E(G) \mid e \subseteq U\}$. For the graph $G[V - U]$ we also use the notation $G - U$. We say that U *separates* two vertices v and w, if $v \neq w$ and there is no path in $G - U$ from v to w.

Two graphs G and H are *isomorphic* (in symbols $G \cong H$) if there is a bijection $\tau : V(G) \to V(H)$, such that for all $u, v \in V(G)$, $\{u, v\} \in E(G)$ if and

[2] They also have an $O(n \log n)$ time algorithm which they refer to as log-linear time and space in their paper.

only if $\{\tau(u), \tau(v)\} \in E(H)$. In case the vertices of G and H are labeled (or colored), then the isomorphism τ must also preserve the labels (resp. colors). An *automorphism* of a (possibly vertex/edge colored) graph $G = (V, E)$ is a bijection $\xi : V \longrightarrow V$ that is an isomorphism from G to itself. The set $Aut(G)$ of all automorphisms of a graph G is a group under permutation composition. The Graph Isomorphism problem, denoted GI, is to decide if two input graphs G and H are isomorphic. Further, we consider the following problems:

– AUT (automorphism group): On input a graph G, compute a generating set for $Aut(G)$.
– PGA (partially specified graph automorphism): Given a graph G and a list of pairs of vertices $(u_1, v_1), \ldots, (u_l, v_l)$, does G have an automorphism ξ such that $\xi(u_i) = v_i$ for $i = 1, \ldots, l$? Let SGA denote the search version of PGA.

An isomorphism $\phi : V(G) \longrightarrow V(canon(G))$ from a graph G to its canonical form $canon(G)$ is called a *canonical labeling*. We assume $V(canon(G)) = \{1, 2, \ldots, |V(G)|\}$. Hence, the canonical labeling actually gives an ordering of the vertices of G. For a canonical form, the set of isomorphisms from G to $canon(G)$ is the *canonical labeling coset* $CL(G)$. Clearly, $CL(G) = Aut(G)\xi$, for some isomorphism ξ from G to $canon(G)$. Thus, $CL(G)$ can be represented by some $\xi \in CL(G)$ together with a generating set for $Aut(G)$. As shown by Gurevich, canonization of general graphs is polynomial-time equivalent to computing a complete invariant [9]. We define two closely related problems on canonization.

– CL-COSET: Given a graph G compute the canonical labeling coset $CL(G)$ of G.
– COLOR-CL: Given a colored graph G, compute some isomorphism from G to $canon(G)$, i.e., compute a member of the canonical labeling coset $CL(G)$ of the colored graph G.

Notice that COLOR-CL is a search problem (i.e. there might exist more than one solution for a given graph G). We assume that an oracle for COLOR-CL is actually a *functional* oracle that returns for any query G some canonical labeling in $CL(G)$.

2 Relative Complexity of Computing Canonical Forms, Canonical Labelings, and Labeling Cosets

It is well-known (see, e.g., [11,16]) that the problems AUT, PGA, and SGA are all polynomial-time equivalent to Graph Isomorphism. Similarly, the problem of computing canonical forms is easily seen to be polynomial-time equivalent to the problem CL-COSET of computing the corresponding canonical labeling coset as well as to COLOR-CL [8]. However, it is not clear whether these reductions can also be performed in logspace. In the following we compare the different problems w.r.t. logspace Turing reductions (denoted by $\leq_T^L$) and show that all these problems reduce to the canonical labeling problem for colored graphs.

Lemma 1. pGA $\leq_T^L$ Aut $\leq_T^L$ sGA $\leq_T^L$ Color-CL.

As Aut $\leq_T^L$ Color-CL we easily get the following consequence.

Corollary 2. CL-Coset $\leq_T^L$ Color-CL.

In order to canonize a given partial 2-tree G, we decompose G into its biconnected components $G_1, \ldots, G_r$. Since G is a tree T of its biconnected components, we will essentially canonize T using the biconnected component canonization as a subroutine. Now, for this entire procedure to work in logspace, we design a deterministic logspace base machine that makes calls to subroutines that canonize the biconnected partial 2-trees $G_1, \ldots, G_r$ and T. As suggested by the reductions in this section, it suffices to solve the Color-CL problem for $G_1, \ldots, G_r$ and T in order to be able to compute a canonical labeling of G.

3 Canonizing Biconnected Partial 2-Trees

First we recall the definition of (partial) k-trees (cf. [12]).

Definition 3. *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.*

In this section we give a logspace algorithm for biconnected partial 2-tree canonization. But first we state a useful characterization of partial 2-trees [15].

Definition 4. *Let $G = (V, E)$ be a graph. A vertex $v \in V$ is an* articulation point *if $G - v$ has more connected components than G. G is* biconnected *if it has no articulation point. A* cell *of G is a set of edges in a chordless cycle of G. The* cell-completion *of $G = (V, E)$ is the graph $\overline{G} = (V, E')$ where E' is obtained from E by adding all edges $\{x, y\} \subseteq V$ for which $G - \{x, y\}$ has at least three connected components.*

Definition 5. *A* tree of cycles *is a member of the class $\mathcal{TC}$ of graphs defined inductively as follows:*

- *Every chordless cycle is an element of $\mathcal{TC}$.*
- *Given a graph G in $\mathcal{TC}$ and a chordless cycle C, the graph obtained by identifying an edge and its two end vertices of C with an edge and its two end-vertices of G is also in $\mathcal{TC}$.*

Theorem 6. [15] *A biconnected graph G is a partial 2-tree if and only if its cell-completion $\overline{G}$ is a tree of cycles.*

We first consider COLOR-CL for colored biconnected partial 2-trees G. For this, we exploit the special structure of G as explained in Definition 5 and Theorem 6. More precisely, we now give logspace algorithms for computing the tree completion $\overline{G}$ of G and its decomposition into the "tree of cycles". This will allow us to again use ideas from Lindell's tree canonization algorithm to solve the problem.

A logspace algorithm can check for each pair $\{x, y\} \subseteq V$ if $G - \{x, y\}$ has three connected components. If this is the case and if $\{x, y\}$ is not an edge in G then the algorithm outputs $\{x, y\}$ as a "red" edge in the cell-completion $\overline{G}$. We color $\{x, y\}$ red to indicate that the edge is not present in G.

Lemma 7. *Let $G = (V, E)$ be a tree of cycles. Two distinct edges $e_1 = \{x, y\}$ and $e_2 = \{a, b\}$ are in the same cell if and only if either of the following conditions holds true:*

1. *The set $\{x, b\}$ separates y from a and the set $\{y, a\}$ separates x from b but $\{y, b\}$ does not separate x from a and $\{x, a\}$ does not separate y from b.*
2. *The set $\{x, a\}$ separates y from b and the set $\{y, b\}$ separates x from a but $\{y, a\}$ does not separate x from b and $\{x, b\}$ does not separate y from a.*

Notice that the conditions stated in Lemma 7 can be verified by querying an oracle for s-t connectivity and hence is in logspace [19]. As a consequence, for any tree of cycles G we can compute the cells $C_1, C_2, \ldots, C_m$ of G in logspace.

Definition 8. *Let $G = (V, E)$ be a tree of cycles with cells $C_1, \ldots, C_m$. The* skeleton *of G is the tree $S(G) = (V', E')$ with vertex set $V' = \{C_1, \ldots, C_m\} \cup E$ and $\{C_i, e\} \in E'$ if and only if $e \in C_i$.*

Notice that the bipartite graph $S(G) = (V', E')$ is a tree when G is a tree of cycles. For two cells C_i and C_j in a tree of cycles G we have $C_i \cap C_j = \{e\}$ precisely when $\{C_i, e\}$ and $\{C_j, e\}$ are edges in the skeleton $S(G)$. Since we can find the cells $C_1, C_2, \ldots, C_m$ of G in deterministic logspace, it follows that $S(G)$ can be computed in deterministic logspace.

Definition 9. *Let $e = \{a, b\}$ be an edge of a tree of cycles G. Suppose we orient e as the ordered pair (a, b). This orientation naturally induces an orientation of every edge on any cycle C containing e, by walking along C in the (a, b) direction. Applying this step repeatedly yields a unique orientation of all edges. We call this the* orientation *of G induced by (a, b).*

The following symmetry property clearly holds for orientations: Let $e = \{a, b\}$ and $e' = \{a', b'\}$ be two edges in a tree of cycles G. Then the orientation (a, b) induces the orientation (a', b') if and only if the orientation (a', b') induces the orientation (a, b).

Lemma 10. *Let G be a tree of cycles and let $e_0 = \{a, b\}$ be an edge in G. Then the orientation of any edge $e' = \{x, y\}$ of G induced by (a, b) can be computed in deterministic logspace.*

Proof. Let $C_1, \ldots, C_m$ be the cells of G and suppose that $e_0 \in C$ and $e' \in C'$. Let $2d$ be the distance of C' from C in the skeleton $S(G)$ of G and let $C = C_{i_0}, e_1, C_{i_1}, e_2, C_{i_2}, \ldots, e_d, C_{i_d} = C'$ be the path form C to C' in $S(G)$, where $C_{i_{t-1}} \cap C_{i_t} = \{e_t\}$ for $t = 1, \ldots, d$. If $d = 0$, then the orientation of e' induced by (a, b) is computable in logspace by traversing the nodes in the cycle $C = C'$. If $d \geq 1$, the algorithm determines the orientations of $e_1, \ldots, e_d$ one after another. Once it knows the orientation of e_d, the orientation of e' can be determined in the same way as in the case $C = C'$. Inductively suppose the algorithm knows C_{i_t} (i.e., the index i_t), e_t and the orientation of e_t induced by (a, b). In order to find $C_{i_{t+1}}$, e_{t+1} and the orientation of e_{t+1} induced by (a, b) it finds (in logspace) a vertex C'' in the skeleton $S(G)$ such that for some edge e'' of G, $C_{i_t} \cap C'' = \{e''\}$ and the unique path in $S(G)$ from C_{i_t} to C' passes through C''. Since $S(G)$ is a tree, it is clear that the unique choice for C'' is $C_{i_{t+1}}$ and that $e'' = e_{t+1}$. Also, the orientation of e_{t+1} induced by (a, b) can be determined from the orientation of e_t by traversing the nodes in the cycle C_{i_t}. □

3.1 The Tree Representation for the Tree of Cycles

Let G be a colored tree of cycles, C be a given cell in G and let $e = (a, b)$ be an oriented edge in C. Let $C_1, \ldots, C_m$ be the cells of G and let $e_1, \ldots, e_l$ be the oriented edges of G induced by (a, b). The *tree representation* of G with respect to the cell C and the oriented edge (a, b) is a colored ordered tree $T(G, C, a, b) = (V, E)$ with root C, where $V = \{C_1, \ldots C_m\} \cup \{e_1, \ldots, e_l\}$ and $E = \{\{e_i, C_j\} \mid e_i \in C_j\}$. We call the nodes $C_1, \ldots, C_m$ "cell-nodes" and the nodes $e_1, \ldots, e_l$ are referred to as "edge-nodes". Thus, we have cell-nodes and edge-nodes alternating along any root to leaf path, starting with the cell-node C as root. The coloring of $T(G, C, a, b)$ and the ordering of the children of each cell-node C_i will be described below (the children of the edge-nodes e_i are unordered). Our goal is to describe an appropriate canonical labeling (computable in logspace) of $T(G, C, a, b)$ from which we can extract (in logspace) a canonical labeling of the colored tree of cycles G (for a given cell C and oriented edge (a, b)). Once we do this, we can determine the overall canonical labeling as the one yielding the lexicographically smallest tree of cycles over all root cells C as well as all oriented edges (a, b) on C.

First, we notice certain restrictions on this tree representation enforced by the cell structure and the orientation of edges. We claim that as soon as a root cell C and an oriented edge (a, b) on C are fixed, the children of any cell-node C_i can be totally ordered in a canonical way. To see how, we start with the root C. Its children are the edge-nodes corresponding to the edges in the cycle C. We designate (a, b) as the first child of C and the remaining children are ordered by walking along C in the (a, b) direction. For any other cell-node C_i let (a_i, b_i) be its parent edge-node in $T(G, C, a, b)$. The children of C_i are the edge-nodes corresponding to the remaining edges in C_i and they are ordered by walking from node a_i along C in the (a_i, b_i) direction.

Since some edges and vertices of G may be colored, we need to add this information to the nodes of $T(G, C, a, b)$. If $\{x, y\}$ is an edge of G that is colored

c, and (x, y) is the edge-node of $T(G, C, a, b)$ corresponding to $\{x, y\}$ then we include the color c to the set of colors for edge-node (x, y). Further, for each vertex v of G having color c, the color $(1, c)$ is included in the color set of each edge-node in $T(G, C, a, b)$ of the form (v, u) and the color $(c, 1)$ is added to the color set of each edge-node of the form (u, v). This completes the description of the tree representation $T(G, C, a, b)$ of a tree of cycles G.

It is easy to see that any tree T isomorphic to $T(G, C, a, b)$ contains enough information to reconstruct the original tree of cycles G (up to isomorphism) from T. In fact, as we will show next, G is even computable in logspace from T. We first list a set of conditions that are necessary and sufficient for the existence of a tree of cycles G having T as its tree representation. For a node v in T whose children are totally ordered as $u_1, \ldots, u_{l-1}$, the *orientation-order* of the neighbors of v is defined as follows. If v is the root node r of T, then the orientation-order is $(u_1, \ldots, u_{l-1})$. If $v \neq r$, then the orientation-order is $(u_0, u_1, \ldots, u_{l-1})$, where u_0 is the parent of v. In either case we say that the neighbors of v are *color-consistent*, if u_i has a color $(c, 1)$ in its color set if and only if $succ(u_i)$ has the color $(1, c)$ in its color set, where $succ(u_i)$ denotes the cyclic successor to u_i in the orientation-order. Now it is not hard to prove the following lemma.

Lemma 11. *Let G be a tree of cycles, C be a cell of G and let (a, b) be an oriented edge on C. Then the tree representation $T = T(G, C, a, b)$ fulfills the following properties:*

1. *Every node at even distance from the root r of T has degree at least three.*
2. *The children of each node v at even distance from r are totally ordered and the neighbors of v are color-consistent with respect to their orientation-order.*
3. *The nodes at odd distance from r are colored by a set of colors of the form c, $(1, c)$ or $(c, 1)$ containing at most one color of the form c.*

Further, for each tree T fulfilling these properties, a tree of cycles G', a cell C in G' and an oriented edge (a, b) on C fulfilling $T(G', C, a, b) \cong T$ is computable in FL.

3.2 Isomorphism Ordering

We use colored tree canonization to canonize the tree representation T of a tree of cycles. For that we encode the orientation-order using special colors $c_1, c_2, \ldots$ where we assume that $c_i < c_{i+1}$. Let $(u_0, u_1, \ldots, u_{l-1})$ be the orientation-order of the neighbors of a node v in T. If v is the root r of T then we color u_i with color c_{i+1} for $i = 0, \ldots, l-1$. If $v \neq r$, then we color u_i with color c_i for $i = 1, \ldots, l-1$. Notice that in the latter case the parent u_0 of v does not get any color due to the orientation-order of the neighbors of v (though u_0 may get some color due to the orientation-order of the neighbors of some other node). We denote the (unordered) tree obtained from T by adding these special colors by $\hat{T}$.

For an ordered set C of colors let $\mathcal{T}_C$ denote the class of all rooted trees whose nodes are colored with colors from C. We denote the color of a node v by $col(v)$

and the number of its children by $\#v$. The number of nodes in a graph G is denoted as $|G|$. First we inductively define an ordering $\preceq$ on $\mathcal{T}_C$.

Definition 12. *Let $T, T' \in \mathcal{T}_C$ with roots r and r', respectively. We say that $T \preceq T'$ if either of the following conditions is fulfilled:*

1. *$col(r) < col(r')$.*
2. *$col(r) = col(r')$ and $|T| < |T'|$.*
3. *$col(r) = col(r')$, $|T| = |T'|$ and $\#r < \#r'$.*
4. *$col(r) = col(r')$, $|T| = |T'|$, $\#r = \#r'$ and $(T_{i_1}, \ldots, T_{i_k}) \preceq (T'_{j_1}, \ldots, T'_{j_k})$ in the lexicographic sense, where $T_1, \ldots, T_k$ are the subtrees rooted at the children of r, $T'_1, \ldots, T'_k$ are the subtrees rooted at the children of r', and inductively $T_{i_1} \preceq \cdots \preceq T_{i_k}$ and $T'_{j_1} \preceq \cdots \preceq T'_{j_k}$.*

For any two trees $T, T' \in \mathcal{T}_C$ at least one of $T \preceq T'$ and $T' \preceq T$ holds. Further, T' and T are isomorphic if and only if both $T \preceq T'$ and $T' \preceq T$ hold. The ordering defined in Definition 12 is similar to the one defined in [17] except that we give highest priority to colors of nodes to distinguish the subtrees rooted at them. This isomorphism ordering of colored trees gives an isomorphism ordering of the tree representations of the trees of cycles when we encode the orientation-orders by colors. We can easily modify Lindell's algorithm to take into account the priority due to coloring and get a logspace algorithm for computing the isomorphism order of colored trees in $\mathcal{T}_C$. By applying this algorithm we get a deterministic logspace algorithm for computing the isomorphism order of the tree representations of the trees of cycles.

3.3 Canonical Labeling of Biconnected Partial 2-Trees

We first explain how a colored tree $T \in \mathcal{T}_C$ is traversed using the isomorphism order defined above. The traversal starts at the root of T. Suppose we arrive at a node v. Then the algorithm tries to move to the first child in the isomorphism order, i.e., the child with minimal isomorphism order. Ties are broken using the input order. If it fails to move to the first child (if v is a leaf node), it tries to go back to the parent of v. When it comes back to the parent from a child it tries to move to the next sibling in the isomorphism order. Again ties are broken using the input order. If it fails to move to the next sibling (if it comes back from the last child) then it tries to move to the parent (if the node has no parent, the traversal stops).

To determine the canonical labeling of a tree $T \in \mathcal{T}_C$, the algorithm traverses T and when it discovers a *new* node it outputs the node. Note that a node is *new* if it comes through a successful first-child or next-sibling move. The list of nodes in the traversal order serves as the canonical labeling of T. To find the canonical labeling of the tree representation T of a tree of cycles G we first encode the orientation-order using colors and then find the canonical labeling of the colored tree as just described.

Now we can compute the canonical labeling of a tree of cycles G as follows. The algorithm computes for any cell C in G and each oriented edge (a, b) of C

the tree representation $T(G, C, a, b)$. Then it determines the canonical labeling for $\hat{T}(G, C, a, b)$ and reconstructs from the relabeled tree representation in deterministic logspace a tree of cycles G' as stated in Lemma 11. G' is our canonical form $canon(G, C, a, b)$ of G with respect to C and (a, b). The canonical form of G is the lexicographically least such canon.Notice that also the canonical labeling can be recovered by keeping track of the original vertices from the canonical labeling of $\hat{T}(G, C, a, b)$.

Clearly, the canonical labeling of the cell-completion $\overline{G}$ of a biconnected partial 2-tree G can serve as a canonical labeling of G because in $\overline{G}$ the edges that are not present in G are colored "red".

Theorem 13. *For colored biconnected partial* 2*-trees a canonical labeling can be computed in logspace.*

4 Canonizing Partial 2-Trees

In this section we show that the problem Color-CL for partial 2-trees is in logspace. We show this by designing a logspace canonical labeling algorithm for partial 2-trees that is a combination of Lindell's tree canonization algorithm with the logspace canonization algorithm of Section 3 for the class of biconnected partial 2-trees.

Theorem 14. *The problem* Color-CL *for partial* 2*-trees is in logspace. I.e. colored partial* 2*-trees are canonizable in logspace.*

Proof (sketch). Let $G = (V, E)$ be a colored input partial 2-tree for our canonization algorithm. Using Reingold's logspace *s*-*t* connectivity algorithm for undirected graphs [19] we can compute its biconnected components $G_1, \ldots, G_r$ and the set A of its articulation points. Consider the tree $T(G) = (V', E')$ of biconnected components and articulation points defined as follows: $V' = \{v_1, \ldots, v_r\} \cup A$, where v_i corresponds to G_i for each i and $E' = \{\{a, v_i\} \mid a \text{ is an articulation point in } G_i\}$. We can assume that $G, G_1, \ldots, G_r$ and $T(G)$ are given as input.

By distinguishing some articulation point $a \in A$ as the root, the tree $T(G)$ becomes a rooted tree $T(G, a)$ with root a, defined as follows. Based on Lindell's canonization method [17], given G with a vertex $v \in V$ specially marked as root, in the following steps we describe an inductive definition of an ordering on such rooted trees $T(G)$. Using this inductive definition we will be able to compute a canonical labeling (and hence a canonical form) of G. We use $\#r(v)$ to denote the number of children of the root $r(v)$ in $T(G, v)$. Further, for a node $v' \in V'$ we denote the subgraph of G corresponding to the subtree of $T(G, v)$ originating from node v' by $G(v')$. In case $v' = v_i$ corresponds to some biconnected component G_i, we assume that the parent $a_i \in A$ of v_i in $T(G, v)$ (if it exists) is colored with the special color "red" in $G(v_i)$.

Let $G, G' \in \mathcal{G}$ with vertices v and v' marked as root, respectively. We define $G \preceq G'$ if one of the following conditions holds:

1. $size(G) < size(G')$.
2. $size(G) = size(G')$ and $\#r(v) < \#r(v')$.
3. $size(G) = size(G')$, $\#r(v) = \#r(v')$ and v is an articulation point in G but v' is not an articulation point in G'.
4. $size(G) = size(G')$, $\#r(v) = \#r(v')$, either both or neither of v and v' are articulation points in G and G', respectively, and $(G(v_1), \ldots, G(v_s)) \prec (G'(v'_1), \ldots, G'(v'_s))$ in the lexicographic sense ($G \prec G'$ means that $G \preceq G'$ holds but not $G' \preceq G$), where $v_1, \ldots, v_s$ and $v'_1, \ldots, v'_s$ are the children of $r(v)$ and $r(v')$ in $T(G, v)$ and $T(G', v')$, respectively, and the corresponding graphs are inductively ordered as $G(v_1) \preceq \cdots \preceq G(v_s)$ and $G'(v'_1) \preceq \cdots \preceq G'(v'_s)$.
5. If in item 4 we have $G(v_i) \approx G'(v'_i)$ for $i = 1, \ldots, s$ (where $G \approx G'$ means that both $G \preceq G'$ and $G' \preceq G$ hold) and if $r(v)$ and $r(v')$ correspond to biconnected components G_i and G'_j of G and G', respectively, then let $a_1, \ldots, a_s$ and $a'_1, \ldots, a'_s$ be the children of $r(v)$ and $r(v')$ in $T(G, v)$ and $T(G', v')$, respectively. Inductively find the indices $0 = i_0 < i_1 < \cdots < i_{k-1} < i_k = s$ such that $G(a_1) \approx \cdots \approx G(a_{i_1}) \prec G(a_{i_1+1}) \approx \cdots \approx G(a_{i_2}) \prec G(a_{i_2+1}) \approx \cdots \approx G(a_{i_k}) \prec G(a_{i_k+1}) \approx \cdots \approx G(a_s)$. In the biconnected component G_i color v with the special color "red" and color the articulation point a_i with color r if and only if $i_{r-1} + 1 \leq i \leq i_r$. Similarly color the vertices $v', a'_1, \ldots, a'_s$ in G'_j. Now canonize the colored biconnected graphs G_i and G'_j (using the oracle for canonizing graphs in $\mathcal{B}(\mathcal{G})$) and if $G_i \preceq G'_j$ then order $G \preceq G'$.

We claim that this inductive ordering defines a canonical form for the graph G. It remains to argue that the canonical form can be computed in logspace, basically using Lindell's algorithm. As shown in Section 3, we can compute a tree representation for biconnected partial 2-trees. If G is a partial 2-tree with more than one biconnected component, then in the rooted tree defined above each biconnected component B of G will have a fixed red node r (which is an articulation point of G). Using this property, we can give a modified tree representation $T(B, r)$ for B in which r is the root, its children are the cells of B containing r, for each such cell C its children will be an ordered list of *vertices* (where the ordering is either forward or backward determined by orienting a fixed edge incident on r), and so on. The vertices of B that are other articulation points of G will be initially colored blue. In the overall tree structure, these blue nodes will have as children the red nodes which are roots of the tree representation of other biconnected components. An overview of the algorithm is now as follows: We start Lindell's algorithm on the tree $T(G, a)$ rooted at articulation point a. Every time we need to compare two trees rooted at two biconnected components as in Step 5 of the inductive definition, we will essentially have to compare two trees of the form $T(B, r)$ and $T(B', r')$. We will again use Lindell's method here. We classify the blue nodes into blocks, using the level numbers and the sizes of the subgraphs rooted at them. We first recusively compare nodes that are identical blocks of size 1 (crucially, we do not need any storage to do this recursion similar to Lindell) and order $T(B, r)$ and $T(B', r')$ as per this outcome. If all single blocks turn out to have same isomorphism order, as recursively computed, we proceed to run Lindell on the trees $T(B, r)$ and $T(B', r')$, where we need to orient an edge on roots r and r' and remember the orientation (in all 2 bits of space). The rest of the computation proceeds exactly as in Lindell, where blocks of larger size are recursively compared and the space is managed to be logarithmic. Finally, notice that to canonize G we can run the above procedure to canonize $T(G, a)$ for each articulation point a and choose the lexicographically smallest amongst them. Details of the algorithm and correctness proof will be given in the full version. □

5 Recognizing Partial 2-Trees

Jakoby and Liskiewicz [13] proved that recognition of partial 2-trees is in SL. Hence, it is in L by Reingold's result [19]. Here we give a simple logspace algorithm for recognizing partial 2-trees is based on Theorem 6 and Lemma 7. By Theorem 6 it suffices to check that the cell-completion of each biconnected component B of G is a tree of cycles. Clearly, the cell-completion $\overline{B}$ of B can be computed in logspace as described in Section 3. In order to check that $\overline{B} = (V, E)$ is a tree of cycles, the algorithm computes for each edge $e = \{x, y\}$ in $\overline{B}$ a set C_e consisting of all edges $e' = \{a, b\}$ for which either of the two conditions stated in Lemma 7 is satisfied. Next it removes all duplicate occurrences of the sets C_e. Let $C_1, \ldots, C_m$ be the remaining sets. If $|C_i \cap C_j| > 1$ for some $1 \leq i < j \leq m$, then we know that G cannot be a tree of cycles. Otherwise the algorithm checks for $i = 1, 2, \ldots, m$ that the graph $G[V_i]$ induced by the vertex set V_i of C_i is actually a cycle by verifying that each node has degree 2 and the graph $G[V_i]$ is connected. Finally, the algorithm checks that the bipartite graph $S(\overline{B}) = (V', E')$ is a tree, where $V' = \{C_1, \ldots, C_m\} \cup E$ and $E' = \{\{C_i, e\} \mid e \in C_i\}$. Recall from Definition 8 that $S(\overline{B})$ is just the *skeleton* of $\overline{B}$ in case $\overline{B}$ is a tree of cycles. Now it is easy to see that G is a partial 2-tree if and only if the cell-completion $\overline{B}$ of each biconnected component B of G passes all the tests described above.

Theorem 15. *Given a graph G there is a deterministic logspace algorithm to decide if G is a partial 2-tree.*

References

1. Arvind, V., Das, B., Köbler, J.: The space complexity of k-tree isomorphism. In: Proc. 18th International Symposium on Algorithms and Computation. LNCS, pp. 822–833. Springer, Heidelberg (2007)
2. 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)
3. 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)

4. Arnborg, S., Proskurowski, A.: Linear time algorithms for NP-hard problems restricted to partial k-trees. Discrete Applied Mathematics 23(2), 11–24 (1989)
5. Arnborg, S., Proskurowski, A.: Canonical representations of partial 2- and 3-trees. BIT 32(2), 197–214 (1992)
6. Babai, L., Erdős, P., Selkow, S.M.: Random graph isomorphism. SIAM Journal on Computing 9(3), 628–635 (1980)
7. Babai, L., Kučera, L.: Canonical labelling of graphs in linear average time. In: Proc. 20th IEEE Symposium on the Foundations of Computer Science, pp. 39–46 (1979)
8. Babai, L., Luks, E.: Canonical labeling of graphs. In: Proc. 15th ACM Symposium on Theory of Computing, pp. 171–183 (1983)
9. Gurevich, Y.: From invariants to canonization. Bulletin of the European Association of Theoretical Computer Science (BEATCS) 63, 115–119 (1997)
10. 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)
11. Hoffmann, C.M.: Group-Theoretic Algorithms and Graph Isomorphism. LNCS, vol. 136. Springer, Heidelberg (1982)
12. Harary, F., Palmer, E.M.: On acyclic simplicial complexes. Mathematica 15, 115–122 (1968)
13. Jakoby, A., Liskiewicz, M.: Paths Problems in Symmetric Logarithmic Space. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 269–280. Springer, Heidelberg (2002)
14. Jenner, B., Köbler, J., McKenzie, P., Torán, J.: Completeness results for graph isomorphism. Journal of Computer and System Sciences 66, 549–566 (2003)
15. Kloks, T.: Treewidth: Computation and Approximation. LNCS, vol. 842. Springer, Heidelberg (1994)
16. Köbler, J., Schöning, U., Torán, J.: The Graph Isomorphism Problem: Its Structural Complexity. In: Progress in Theoretical Computer Science. Birkhäuser, Boston (1993)
17. 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)
18. Luks, E.M.: Permutation groups and polynomial time computations. In: Finkelstein, L., Kantor, W.M. (eds.) Groups and Computation. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 11, pp. 139–175. American Mathematical Society (1993)
19. Reingold, O.: Undirected ST-connectivity in log-space. In: Proceedings of the 37th Annual ACM Symposium on Theory of Computing, pp. 376–385 (2005)
20. Thierauf, T., Wagner, F.: The isomorphism problem for planar 3-connected graphs is in unambiguous logspace. Technical Report TR07-068, Electronic Colloquium on Computational Complexity (ECCC) (2007)