2009 24th Annual IEEE Conference on Computational Complexity

Planar Graph Isomorphism is in Log-Space

Samir Datta*, Nutan Limaye†, Prajakta Nimbhorkar†, Thomas Thierauf‡ , Fabian Wagner§
*Chennai Mathematical Institute
Email: sdatta@cmi.ac.in
†The Institute of Mathematical Sciences, Chennai
Email: {nutan,prajakta}@imsc.res.in
‡Fakultät für Elektronik und Informatik, HTW Aalen
Email: thomas.thierauf@uni-ulm.de
§Institut für Theoretische Informatik, Universität Ulm
Email: fabian.wagner@uni-ulm.de

Abstract

Graph Isomorphism is the prime example of a computational problem with a wide difference between the best known lower and upper bounds on its complexity. There is a significant gap between extant lower and upper bounds for planar graphs as well. We bridge the gap for this natural and important special case by presenting an upper bound that matches the known log-space hardness [JKMT03]. In fact, we show the formally stronger result that planar graph canonization is in log-space. This improves the previously known upper bound of AC^1 *[MR91].*

Our algorithm first constructs the biconnected component tree of a connected planar graph and then refines each biconnected component into a triconnected component tree. The next step is to log-space reduce the biconnected planar graph isomorphism and canonization problems to those for 3-connected planar graphs, which are known to be in log-space by [DLN08]. This is achieved by using the above decomposition, and by making significant modifications to Lindell's algorithm for tree canonization, along with changes in the space complexity analysis.

The reduction from the connected case to the biconnected case requires further new ideas, including a non-trivial case analysis and a group theoretic lemma to bound the number of automorphisms of a colored 3-connected planar graph. This lemma is crucial for the reduction to work in log-space.

1. Introduction

The graph isomorphism problem GI consists of deciding whether there is a bijection between the vertices of two graphs, which preserves the adjacency relations. The wide gap between the known lower and upper bounds has kept alive the research interest in GI.

The problem is clearly in NP and by a group theoretic proof also in SPP [AK06]. This is the current frontier of our knowledge as far as upper bounds go. The inability to give efficient algorithms for the problem would lead one to believe that the problem is provably hard. NP-hardness is precluded by a result that states if GI is NP-hard then the polynomial time hierarchy collapses to the second level [BHZ87], [Sch88]. What is more surprising is that not even P-hardness is known for the problem. The best we know is that GI is hard for DET [Tor04], the class of problems NC^1-reducible to the determinant, defined by Cook [Coo85].

While this enormous gap has motivated a study of isomorphism in *general* graphs, it has also induced research in isomorphism restricted to special cases of graphs where this gap can be reduced. Tournaments are an example of directed graphs where the DET lower bound is preserved [Wag07], while there is a quasi-polynomial time upper bound [BL83].

Trees are an example of graphs where the lower and upper bounds match and are L [Lin92]. Note that for trees, the problem's complexity crucially depends on the input encoding: if the trees are presented as strings then the lower and upper bound are NC^1 [MJT98], [Bus97]). Lindell's log-space result has been extended to partial 2-trees, also known as generalized series-parallel graphs [ADK08]. Trees and partial 2-trees are special cases of planar graphs.

In this paper we consider planar graph isomorphism and settle its complexity by significantly improving the known upper bound of AC^1. The result is particularly satisfying, because Planar Graph Isomorphism turns out to be complete for a well-known and natural complexity class, namely log-space: L.

Planar Graph Isomorphism has been studied in its own right since the early days of computer science. Weinberg [Wei66] presented an $O(n^2)$ algorithm for testing isomorphism of 3-connected planar graphs. Hopcroft and Tarjan [HT74] extended this to general planar graphs, improving the time complexity to $O(n \log n)$. Hopcroft and Wong [HW74] further improved it to $O(n)$. Recently Kukluk, Holder, and Cook [KHC04] gave an $O(n^2)$ algorithm

‡Supported by DFG grants Scho 302/7-2.
§Supported by DFG grants TO 200/2-2.

978-0-7695-3717-7/09 $25.00 © 2009 IEEE
DOI 10.1109/CCC.2009.16

IEEE computer society

for planar graph isomorphism, which is suitable for practical applications. The parallel complexity of Planar Graph Isomorphism was first considered by Miller and Reif [MR91] and Ramachandran and Reif [RR94]. They showed that the upper bound is AC^1, see also [Ver07].

Recent work has dealt with a further special case, namely 3-connected planar graphs. Thierauf and Wagner [TW08] presented a new upper bound of $\mathsf{UL} \cap \mathsf{coUL}$, making use of the machinery developed for the reachability problem [RA97] and specifically for planar reachability [ADR05], [BTV07]. They also show that the problem is L-hard. Further progress, in the form of a log-space algorithm is made by Datta, Limaye and Nimbhorkar [DLN08] where the 3-connected planar case is settled, by building on ideas from [TW08] and using Reingold's construction of universal exploration sequences [Rei05].

The known results for planar graphs and their restrictions can be summarized as follows:

Graph class	Lower bound	Upper bound
Trees	L [MJT98]	L [Lin92]
Partial 2-trees	L	L [ADK08]
3-connected planar graphs	L [TW08]	L [DLN08]
Planar graphs	L	AC^1 [RR94]

The current work is a natural culmination of this series where we settle the complexity question for planar graph isomorphism by presenting the first log-space algorithm for the problem. In fact, we give a log-space algorithm for the *graph canonization problem*, to which graph isomorphism reduces. The canonization involves assigning to each graph an isomorphism invariant, polynomial length string. In this paper we establish the following theorem:

Theorem 1.1 *Planar graph isomorphism and canonization are in log-space.*

We consider planar undirected graphs without parallel edges and loops, also called *simple* graphs. For planar graphs that are directed or not simple, there are log-space many-one reductions to simple undirected planar graphs (cf. [KST93]). Our log-space algorithm consists of the following steps.

1) Decompose the planar graph into its biconnected components and construct a *biconnected component tree* in log-space ([ADK08], cf. [TW09] and Section 4).
2) Decompose biconnected planar components into their triconnected components to obtain a *triconnected component tree* in log-space. This is essentially a parallel implementation of the sequential algorithm of [HT73] (see Section 3.1).
3) Invoke the algorithm of [DLN08] to canonize the triconnected components of the graph.
4) Canonize biconnected planar graphs using their triconnected component trees. Lindell's algorithm [Lin92] for tree canonization and its complexity analysis had to be modified in a non-trivial way for this step to work in log-space (see Section 3), i.e. from [Lin92] to their triconnected component trees.
5) Canonize planar graphs using their biconnected component trees, by substituting the biconnected components with their triconnected component trees (see Section 4).

Notice, that in Step 4, pairwise isomorphism of two triconnected component trees labelled with the canons of their components does not imply isomorphism of the corresponding graphs. Figure 1 illustrates this fact. So, a naïve combination of [Lin92] and [DLN08] does not work. We need to introduce the concept of *orientations of separating pairs* (see Section 3.2 for details) to ensure the extendibility of isomorphism of individual 3-connected components to the entire biconnected planar graph.

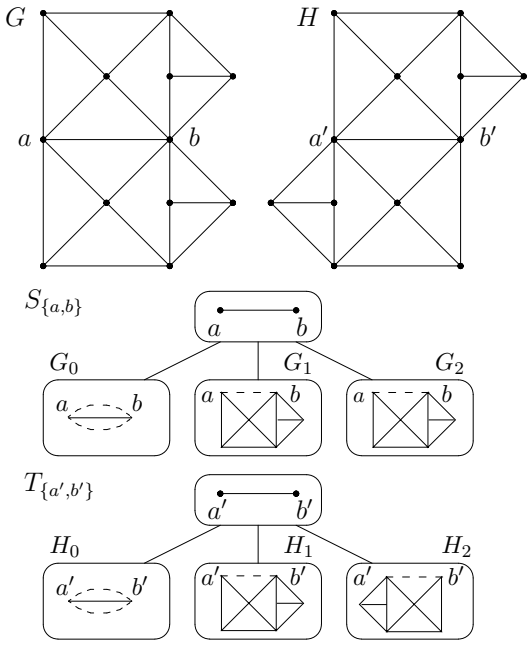

Figure 1: *The graphs G and H have the same triconnected component trees but are not isomorphic.*

Triconnected components have at most two embeddings on the sphere (cf. [Whi33]). This property helps to perform Step 4 in L. Biconnected components do not have this property. Hence, the naïve approach for Step 5 would need to keep track of exponentially many cases. We solve this problem by resorting to an intricate case analysis and a group theoretic lemma (Lemma 4.3) to bound the number of automorphisms of a coloured 3-connected planar graph.

We emphasize that Step 5 does not use Step 4 as a black-box but as a co-routine. In fact, we maintain two (logarithmically bounded) work-tapes for these two steps.

2. Definitions and Notations

We recall some basic graph theoretic notions.

A graph $G = (V, E)$ is *connected* if there is a path between any two vertices in G. For $U \subseteq V$ let $G(U)$ be the *induced subgraph* of G on U. A vertex $v \in V$ is an *articulation point* if $G(V \setminus \{v\})$ is not connected. A pair of vertices $u, v \in V$ is a *separating pair* if $G(V \setminus \{u, v\})$ is not connected. A *biconnected graph* contains no articulation points. A *3-connected graph* contains no separating pairs. A *triconnected graph* is either a 3-connected graph or a cycle or a 3-bond. A k-*bond* is a graph consisting of two vertices joined by k edges. A pair of vertices $\{a, b\}$ is said to be *3-connected* if there are three or more vertex-disjoint paths between them.

For a node v let $d(v)$ be the maximal distance that v has to any of the other nodes of G. Let C be the set of nodes v of G that have minimal value $d(v)$. The set C is called the *center of* G. In other words, vertices in the center minimize the maximal distance from other vertices in the graph. Note that if G is a tree such that every path from a leaf to a leaf has even length, then the center consists of only one node, namely the midpoint of a longest path in the tree.

Let E_v be the set of edges incident to v. A permuatation ρ_v on E_v that has only one cycle is called a *rotation*. A *rotation scheme* for a graph G is a set ρ of rotations,

$$\rho = \{\rho_v \mid v \in V \text{ and } \rho_v \text{ is a rotation on } E_v\}.$$

Let ρ^{-1} be the set of inverse rotations, $\rho^{-1} = \{\rho_v^{-1} \mid v \in V\}$. A rotation scheme ρ describes an embedding of graph G in the plane. If the embedding is planar, we call ρ a *planar rotation scheme*. Note that in this case ρ^{-1} is a planar rotation scheme as well. Allender and Mahajan [AM00] showed that a planar rotation scheme for a planar graph can be computed in log-space.

Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are said to be *isomorphic* ($G_1 \cong G_2$) if there is a bijection $\phi : V_1 \to V_2$ such that $\{u, v\} \in E_1$ if and only if $\{\phi(u), \phi(v)\} \in E_2$. *Graph isomorphism* (GI) is the problem of deciding whether two given graphs are isomorphic.

A planar graph G, along with its planar embedding (given by ρ) is called a *plane graph* $\widehat{G} = (G, \rho)$. A plane graph divides the plane into regions. Each such region is called a *face*. Let Planar-GI be the special case of GI when the given graphs are planar. The biconnected (respectively, 3-connected) planar GI is a special case of Planar-GI when the graphs are biconnected (3-connected) planar graphs.

Let $\mathcal{G}$ be a class of graphs. Let $f : \mathcal{G} \to \{0, 1\}^*$ be a function such that for all $G, H \in \mathcal{G}$ we have $G \cong H \Leftrightarrow f(G) = f(H)$. Then f computes a *complete invariant* for $\mathcal{G}$. If f computes for G a graph $f(G)$ such that $G \cong f(G)$ then we call $f(G)$ the *canon* for G.

By L we denote the languages computable by a log-space bounded Turing machine.

3. Isomorphism Order of Biconnected Planar Graphs

For tree-canonization, Lindell described an isomorphism ordering procedure which arranges subtrees in a canonical order. We define a tree structure for biconnected planar graphs and describe an isomorphism ordering procedure for this tree structure. This leads to an algorithm for canonization.

3.1. The Triconnected Component Trees

Hopcroft and Tarjan [HT73] presented a sequential algorithm for the decomposition of a biconnected planar graph into its triconnected components. Their algorithm recursively removes separating pairs from the graph and puts a copy of the separating pair in each of the components so formed. The nodes in the separating pair are connected by a virtual edge. This procedure gives a tree, called the *triconnected component tree*, on separating pairs and triconnected components. The triconnected components may be 3-connected planar graphs, cycles or bonds. See for example Figure 2. This may not be unique, but a small modification (combining simple cycles which are split at any intermediate steps) gives a decomposition which is unique [Mac37].

This algorithm is inherently sequential. If the removal of one separating pair splits another separating pair then those two cannot be simultaneously removed. Thus, the procedure requires that the removed separating pairs are remembered and may require linear space.

To get around this problem, we crucially use the fact that it is sufficient to remove only 3-connected separating pairs to get the same decomposition as in [HT73]. A separating pair $\{u, v\}$ is said to be *3-connected* if there are three vertex-disjoint paths between u and v. We also observe that these separating pairs can be removed simultaneously without any problems. Also, as both the vertices of each separating pair (in particular 3-connected separating pair) lie on a face, two of the three vertex disjoint paths are simply the face boundaries. Thus, locating such a separating pair boils down to (having removed the face boundary) a reachability question and can be done in log-space. Once these separating pairs are identified, constructing the triconnected components and the triconnected component tree can also be done in log-space. Thus, we get the following theorem:

Theorem 3.1 *The decomposition of biconnected planar graphs into triconnected components is in log-space.*

A proof for this theorem can be found in the full version of this paper. In the following we describe isomorphism ordering of triconnected component trees which is used to obtain the canon of a given biconnected planar graph. The isomorphism order and the canonization of triconnected

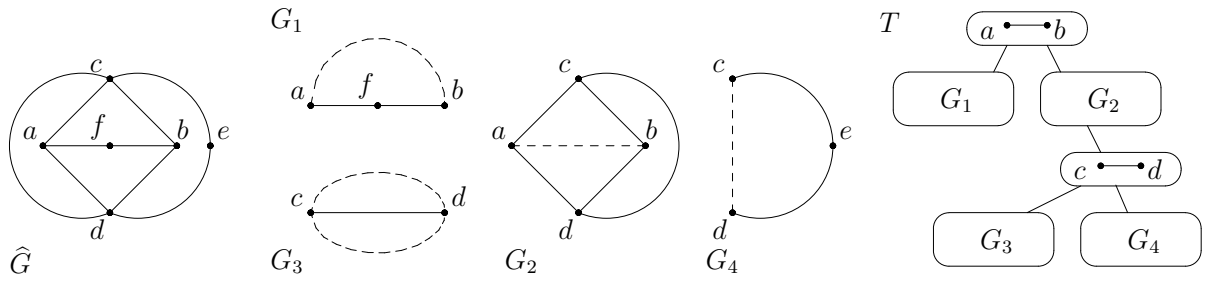

Figure 2: *The decomposition of a biconnected planar graph $\widehat{G}$. Its triconnected components are $G_1, \ldots, G_4$ and the corresponding triconnected component tree is T. In $\widehat{G}$, the pairs $\{a, b\}$ and $\{c, d\}$ are 3-connected separating pairs. The triconnected components are the induced graphs G_1 on $\{a, b, f\}$, G_2 on $\{a, b, c, d\}$, and G_4 on $\{c, d, e\}$. Since the 3-connected separating pair $\{c, d\}$ is connected by an edge in $\widehat{G}$, we also get $\{c, d\}$ as triple-bond G_3. The virtual edges corresponding to the 3-connected separating pairs are drawn with dashed lines.*

component trees are based on Lindell's tree canonization algorithm [Lin92]. We assume familiarity with the canonization algorithm of Lindell.

3.2. Isomorphism Order of Triconnected Component Trees

We describe an isomorphism order procedure for two triconnected component trees S and T, corresponding to two biconnected planar graphs G and H, respectively.

Our isomorphism ordering procedure is more complex than Lindell's algorithm, because each node of the tree is a separating pair or a triconnected component. Thus, unlike in the case of Lindell's algorithm, two leaves in a triconnected component tree are not always isomorphic. In [DLN08], a log-space canonization algorithm for 3-connected planar graphs is described. Note, an obvious way to canonize a triconnected component tree would be to invoke the algorithm of [DLN08] along with Lindell's algorithm. However, this approach does not work. See for example, Figure 1.

We root S and T at separating pair nodes $s = \{a, b\}$ and $t = \{a', b'\}$, respectively, which are chosen arbitrarily, because an isomorphism test can easily run through all the possibilities of choosing these roots. The rooted trees are denoted $S_{\{a,b\}}$ and $T_{\{a',b'\}}$ as shown in Figure 3. The isomorphism order is defined to be $S_{\{a,b\}} <_{\mathrm{T}} T_{\{a',b'\}}$ if:

1) $|S_{\{a,b\}}| < |T_{\{a',b'\}}|$ or
2) $|S_{\{a,b\}}| = |T_{\{a',b'\}}|$ but $\#s < \#t$ where $\#s$ and $\#t$ are the number of children of s and t, respectively, or
3) $|S_{\{a,b\}}| = |T_{\{a',b'\}}|$, $\#s = \#t = k$, but $(S_{G_1}, \ldots, S_{G_k}) <_{\mathrm{T}} (T_{H_1}, \ldots, T_{H_k})$ lexicographically, where we assume that $S_{G_1} \leq_{\mathrm{T}} \ldots \leq_{\mathrm{T}} S_{G_k}$ and $T_{H_1} \leq_{\mathrm{T}} \ldots \leq_{\mathrm{T}} T_{H_k}$ are the ordered subtrees of $S_{\{a,b\}}$ and $T_{\{a',b'\}}$, respectively. To compute the order between the subtrees S_{G_i} and T_{H_i} we compare lexicographically the canons of G_i and H_i and *recursively* the subtrees rooted at the children of G_i and H_i. Note that these children are again separating pair nodes.
4) $|S_{\{a,b\}}| = |T_{\{a',b'\}}|$, $\#s = \#t = k$, $(S_{G_1} \leq_{\mathrm{T}} \ldots \leq_{\mathrm{T}} S_{G_k}) =_{\mathrm{T}} (T_{H_1} \leq_{\mathrm{T}} \ldots \leq_{\mathrm{T}} T_{H_k})$, but $(O_1, \ldots, O_p) < (O'_1, \ldots, O'_p)$ lexicographically, where O_j and O'_j are the orientation counters of the j^{th} isomorphism classes I_j and I'_j of all the S_{G_i}'s and the T_{H_i}'s. The orientation counters are defined below.

We say that two triconnected component trees S_e and $T_{e'}$ are *equal according to the isomorphism order*, denoted by $S_e =_{\mathrm{T}} T_{e'}$, if neither $S_e <_{\mathrm{T}} T_{e'}$ nor $T_{e'} <_{\mathrm{T}} S_e$ holds. The first three steps of the isomorphism ordering are exactly the same as in [Lin92]. However, we use a different notion of *size of a tree*.

Definition 3.2 *For a triconnected component tree T, the* size of an individual component node *C of T is the number n_C of nodes in C. Note that the separating pair nodes are counted in every component where they occur. The* size of the tree *T, denoted by $|T|$, is the sum of the sizes of its component nodes.*

The size of T is at least as large as the number of vertices in the graph corresponding to the triconnected component tree T. The last step in the isomorphism ordering is fairly involved and we describe it here. The new idea here is the use of orientation. We now give a brief outline of Step 4, i.e. computing the orientation. More details of this can be found in the full version of this paper.

Isomorphism of two subtrees rooted at triconnected components *G*, *H*. In [DLN08], to compute the canon of a 3-connected planar graph, the canonizing algorithm takes as input the graph G, an embedding of the graph ρ, a starting edge $\{u, v\}$ and a starting vertex u or v. For a given starting edge, the canon of a cycle can be computed by traversing the cycle either clockwise or counter-clockwise. For bonds there is a unique canon. For the canon of the 3-connected

graph G with starting edge $\{a, b\}$ there are four possibilities: two choices of ρ (because a 3-connected graph has only two planar combinatorial embeddings [Whi33]) and two choices for the starting vertex, a or b. Thus, for a given triconnected component and a starting edge, the number of possible canons are $O(1)$. We construct and compare all $O(1)$ canons of G and H simultaneously, edge-by-edge. If the traversal of one canon encounters a virtual edge and the other does not then we eliminate the former one as the larger canon. If virtual edges are encountered simultaneously in two canons, we make a recursive call for the corresponding children. If the canon of G (H) is found to be smaller than the canon of H (G) then we return $G < H$ ($H < G$, respectively). If the minimum canon of G is the one where a is the starting vertex (i.e. the separating pair $\{a, b\}$ is traversed from a to b) then G is said to give the orientation $a \to b$ to the parent.

Isomorphism of two subtrees rooted at separating pairs s=$\{a, b\}$, t=$\{a', b'\}$. Partition subtrees rooted at the children of s and t into blocks according to their sizes (size-blocks), where sizes are computed as in Definition 3.2. Compare the corresponding size-blocks in both trees one by one starting from the smallest one to further partition the size-blocks into isomorphism classes as in steps $1-3$ of the isomorphism ordering. For Step 4, let $c_i^{\rightarrow}$ and $c_i^{\leftarrow}$ be the number of subtrees in the i-th isomorphism class that give $a \to b$ and $b \to a$ orientations, respectively. We get a pair of counters $O_i = (c_i^{\rightarrow}, c_i^{\leftarrow})$ and $O'_i = (d_i^{\rightarrow}, d_i^{\leftarrow})$ for the i-th isomorphism class of both trees. Define the reference orientation of $\{a, b\}$ and $\{a', b'\}$ as the orientation given by a majority of the children in the smallest isomorphism class where these counters differ. If these counts match across all isomorphism classes then we say that the subtrees are equal according to $=_{\mathrm{T}}$.

In both of the above cases, if the minimum canon of a component is obtained for both orientations of the parent then we say that it does not give any orientation to the parent.

The following theorem states that two trees are $=_{\mathrm{T}}$-equal, precisely when the underlying graphs are isomorphic. A detailed description of the isomorphism ordering for trees rooted at triconnected component nodes and separating pair nodes is provided in a full version of this paper.

Theorem 3.3 *The biconnected planar graphs G and H are isomorphic if and only if there is a choice of separating pairs e, e' in G and H such that $S_e =_{\mathrm{T}} T_{e'}$ when rooted at e and e', respectively.*

The first two steps of the isomorphism order algorithm can be computed in log-space as in Lindell's algorithm [Lin92]. We show that steps 3 and 4 can also be performed in log-space. We use the algorithm of [DLN08] to canonize a triconnected component G_i of size n_{G_i} in space $O(\log n_{G_i})$. In the full version, we carefully describe what can be stored on the work-tape at each level of recursion of the isomorphism ordering procedure. An important point is the following. Consider a triconnected component tree of size n which has a subtree rooted at a child of size $\geq n/2$. We call this a *large child*. It is important for the log-space bound not to store bits on the work-tape before going into recursion at such a large child. There can be only one large child. Hence, we treat large children a priori and locally store the result of the comparison *after* the return from this recursion.

We summarize: while comparing two trees of size N, the algorithm uses no space for making a recursive call for a subtree of size larger than $N/2$, and it uses $O(\log k_j)$ space if the subtrees are of size at most N/k_j, where $k_j \geq 2$. Hence, we get the same recurrence for the space $\mathcal{S}(N)$ as Lindell:

$$\mathcal{S}(N) \leq \max_j \ \mathcal{S}\left(\frac{N}{k_j}\right) + O(\log k_j),$$

where $k_j \geq 2$ for all j. Thus, $\mathcal{S}(N) = O(\log N)$. Note that the number n of nodes of G is in general smaller than N, because the separating pair nodes occur in all components split off by this pair. But we certainly have $n < N \leq O(n^2)$ [HT73]. This proves the following Theorem. A more detailed complexity analysis of the steps 3 and 4 is given in the full version of the paper.

Theorem 3.4 *The isomorphism order between two triconnected component trees of biconnected planar graphs can be computed in log-space.*

To get an intuition, the following example describes the canon for a triconnected component tree. Let $l(S, a, b)$ be the canonical list of edges for the tree $S_{\{a,b\}}$ of Figure 3. The canon is based on the string representation for trees and the canons of triconnected components. Let s_i be the edge connecting the vertices a_i with b_i. We also write for short $l'(S_i, s_i)$ which is one of $l(S_i, a_i, b_i)$ or $l(S_i, b_i, a_i)$. The isomorphism ordering procedure helps to decide the direction of s_i. Let $l(G_i, a, b)$ be the canon of the triconnected component G_i starting with (a, b). The canon is as follows.

$$\begin{aligned}
l(S, a, b) &= [\ (a, b)\ l(S_{G_1}, a, b)\ \ldots\ l(S_{G_k}, a, b)\ ] \text{ with} \\
l(S_{G_1}, a, b) &= [\ l(G_1, a, b)\ l'(S_1, s_1)\ \ldots\ l'(S_{l_1}, s_{l_1})\ ] \\
&\vdots \\
l(S_{G_k}, a, b) &= [\ l(G_k, a, b)\ l'(S_{l_k}, s_{l_k})\ ].
\end{aligned}$$

Finally, the canon for the biconnected planar graph G is obtained then by removing the brackets, by giving vertices new labels in the order of their first occurrences and finally, by ordering the edges of G lexicographically according to the new labels. These tasks can be done in log-space We state the main result of this section.

Theorem 3.5 *A biconnected planar graph can be canonized in log-space.*

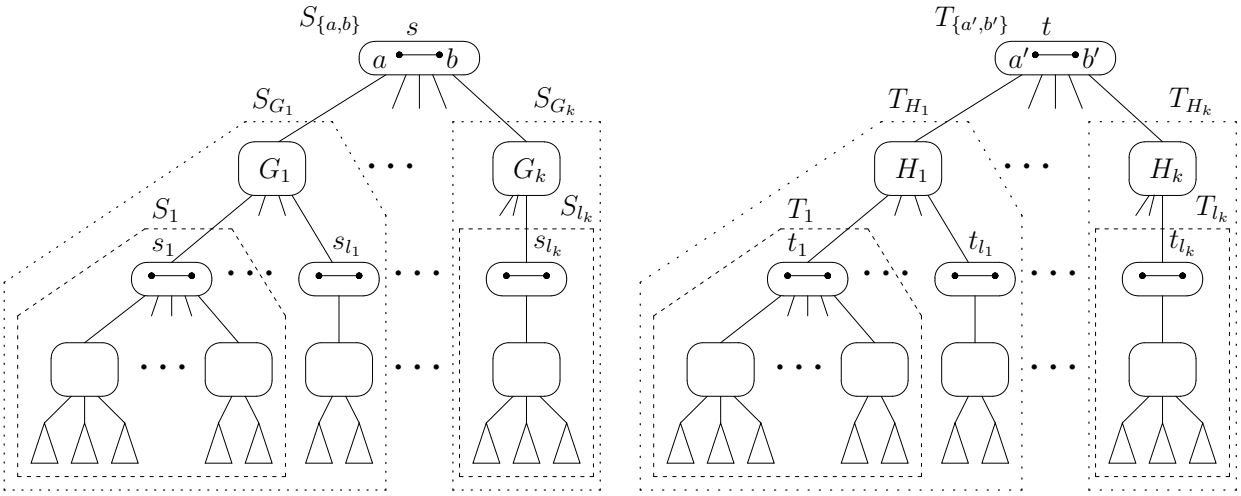

Figure 3: *Triconnected component trees.*

4. Isomorphism Order of Connected Planar Graphs

Similar to that for biconnected graphs, we describe an isomorphism ordering procedure for connected planar graphs. We decompose a connected planar graph into its biconnected components and obtain the biconnected component tree. We describe an isomorphism ordering procedure for this tree structure. This leads to an algorithm for isomorphism testing and canonization for connected planar graphs.

We assume that the planar graph is connected. Otherwise, we test the connected components pairwise for isomorphism. For canonization, we canonize the connected components separately and output the canons in lexicographical order.

4.1. Isomorphism Order of Biconnected Component Trees

We decompose a planar graph into its biconnected components and then construct a tree on these biconnected components and articulation points. We refer to this tree as the *biconnected component tree*. We also refer to the components as *biconnected component nodes* and *articulation point nodes*. This tree is unique and can be constructed in log-space [ADK08].

Similar to triconnected component trees, we put a copy of an articulation point a into each of the components formed by the removal of a. Thus, an articulation point a has a copy in each of the biconnected components obtained by its removal. Note that a naïve approach is to color all copies of an articulation point with a particular color and check the isomorphism of these coloured biconnected components separately. However, this approach does not work as we do not know a priori which articulation points from the two graphs will be mapped to each other. Also, using this method, we cannot ensure that all the copies of an articulation are mapped to the copies of another articulation point in log-space.

In the discussion below, we refer to a copy of an articulation point in a biconnected component B as an *articulation point in* B. Although an articulation point has at most one copy in each of the biconnected components, the corresponding triconnected component trees can have many copies of the same articulation point (if it belongs to a 3-connected separating pair in the biconnected component).

Given a planar graph G, we root its biconnected component tree at an articulation point. During the isomorphism ordering of two such trees S and T, we can fix the root of S arbitrarily and make an equality test for all choices of roots for T. As there are $O(n)$ articulation points, a log-space transducer can cycle through all articulation points in T for this task. We state some properties of articulation points in the following lemma.

Lemma 4.1 *Let B be a biconnected component in S and $\mathsf{T}(B)$ be its triconnected component tree. Then the following holds.*

1) *S has a unique center, similar to a triconnected component tree.*
2) *If an articulation point a of S appears in a separating pair node s in $\mathsf{T}(B)$, then it appears in all the triconnected component nodes adjacent to s in $\mathsf{T}(B)$.*
3) *If an articulation point a appears in two nodes C and D in $\mathsf{T}(B)$, it appears in all the nodes that lie on the path between C and D in $\mathsf{T}(B)$. Hence, there is a unique node A in $\mathsf{T}(B)$ that contains a which is nearest to the center of $\mathsf{T}(B)$. We call A the triconnected component* associated with *a. Thus, we can uniquely associate each articulation point contained in B with a triconnected component in $\mathsf{T}(B)$.*

The isomorphism order for biconnected component trees is defined in three steps that correspond to the first three steps of the isomorphism order for triconnected component trees in Section 3.2 on page 4. We mention the main differences in the isomorphism ordering for biconnected component trees from that of triconnected component trees.

1) The biconnected component nodes are connected by articulation point nodes. The resulting graph is a tree similar to the tree of triconnected component nodes and separating pair nodes. For articulation points, we do not need the notion of orientation. Instead, we color the copy of the parent articulation point in a biconnected component with a distinct color and then the pairwise isomorphism among the subtrees of S and T can be extended to the isomorphism between the corresponding planar graphs G and H in a straight forward way.
2) When we compare biconnected components B and B', then we do not have an obvious, uniquely defined edge as root for the corresponding triconnected component trees $\mathsf{T}(B)$ and $\mathsf{T}(B')$. The naive approach would be to cycle through all separating pairs and finally define the one as root that leads to a minimum canon. However, that way we cannot guarantee that the algorithm works in log-space. Let n_B be the size of B. Note that there can be up to $O(n_B)$ separating pairs. When we go into a recursion at some point, we need to store the edge that is currently the root. That is, we need $O(\log n_B)$ space at one level of recursion and this is too much for an overall log-space bound. Hence, our major task will be to limit the number of possible choices of roots appropriately so that the algorithm runs in log-space.
3) There are some more nontrivial tasks, to guarantee the log-space bound. It is not obvious, what to store on the work-tape when we go into recursion at some node in S or some node in $\mathsf{T}(B)$ and, what can be recomputed. We also need a new definition of the size of a subtree, for the isomorphism ordering to work correctly in log-space.

The size of a triconnected component tree is defined in Definition 3.2 on page 4. Here we extend the definition to biconnected component trees.

Definition 4.2 *Let B be a biconnected component node in a biconnected component tree S, and let $\mathsf{T}(B)$ be the triconnected component tree of B. The* size of B *is defined as $|\mathsf{T}(B)|$ as in Definition 3.2. The size of an articulation point node in S is defined as* 1*. Note that the articulation points may be counted several times, namely in every component they occur. The* size of S*, denoted by $|S|$, is the sum of the sizes of its components.*

We define the isomorphism order for two biconnected component trees S_a and $T_{a'}$ rooted at nodes s and t corresponding to articulation points a and a', respectively, see Figure 4. Define $S_a <_{\mathtt{B}} T_{a'}$ if:

1) $|S_a| < |T_{a'}|$ or
2) $|S_a| = |T_{a'}|$ but $\#s < \#t$ or
3) $|S_a| = |T_{a'}|$, $\#s = \#t = k$, but $(S_{B_1}, \ldots, S_{B_k}) <_{\mathtt{B}} (T_{B'_1}, \ldots, T_{B'_k})$ lexicographically, where we assume that $S_{B_1} \leq_{\mathtt{B}} \cdots \leq_{\mathtt{B}} S_{B_k}$ and $T_{B'_1} \leq_{\mathtt{B}} \cdots \leq_{\mathtt{B}} T_{B'_k}$ are the ordered subtrees of S_a and $T_{a'}$, respectively. To compare the order between the subtrees S_{B_i} and $T_{B'_j}$ we compare the triconnected component trees $\mathsf{T}(B_i)$ of B_i and $\mathsf{T}(B'_j)$ of B'_j. When we reach the first occurrences of some articulation points in $\mathsf{T}(B_i)$ and $\mathsf{T}(B'_j)$ (i.e. the *reference copies* of these articulation points as described later) then we compare *recursively* the corresponding subtrees rooted at the children of B_i and B'_j. Note that these children are again articulation point nodes.

We say that two biconnected component trees are *equal*, denoted by $S_a =_{\mathtt{B}} T_{a'}$, if neither of $S_a <_{\mathtt{B}} T_{a'}$ and $T_{a'} <_{\mathtt{B}} S_a$ holds. The inductive ordering of the subtrees of S_a and $T_{a'}$ proceeds exactly as in Lindell's algorithm, by partitioning them into size-classes and comparing the children in the same size-class recursively. The book-keeping required (e.g. the order profile of a node, the number of nodes in a size-class that have been compared so far) is similar to that in Lindell's algorithm. A detailed description of the isomorphism ordering for trees rooted at biconnected component nodes and articulation point nodes can be found in the full version of the paper. We discuss now how to compare two such subtrees S_B and $T_{B'}$, rooted at biconnected component nodes B and B', respectively. One problem is, that we cannot compare the canons of B and B' bit by bit. The problem is, that we cannot compute the canons without recursively considering the children of B and B'. We rather compare their triconnected component trees. One important computational task is the following.

Limiting the number of possible choices for the root. Let S_a be a biconnected component tree rooted at articulation point a. Let B be a child of a in S_a and $\mathsf{T}(B)$ be the triconnected component tree of the biconnected component B. We show how to limit the number of potential root nodes for $\mathsf{T}(B)$. This is a crucial part to obtain a log-space bound.

Besides the parent a, let B have articulation points $a_1, \ldots, a_l$ for some integer $l \geq 0$, such that a_j is the root node of the subtree S_{a_j} of S_a as in Figure 4. We partition the subtrees $S_{a_1}, \ldots, S_{a_l}$ into classes $E_1, \ldots, E_p$ of equal size subtrees (i.e. size according to Definition 4.2). Let k_j be the number of subtrees in E_j. Let the order of the size classes be such that $k_1 \leq k_2 \leq \cdots \leq k_p$. All articulation points with their subtrees in size class E_j are colored with color j.

To limit the number of potential root nodes for $\mathsf{T}(B)$, we

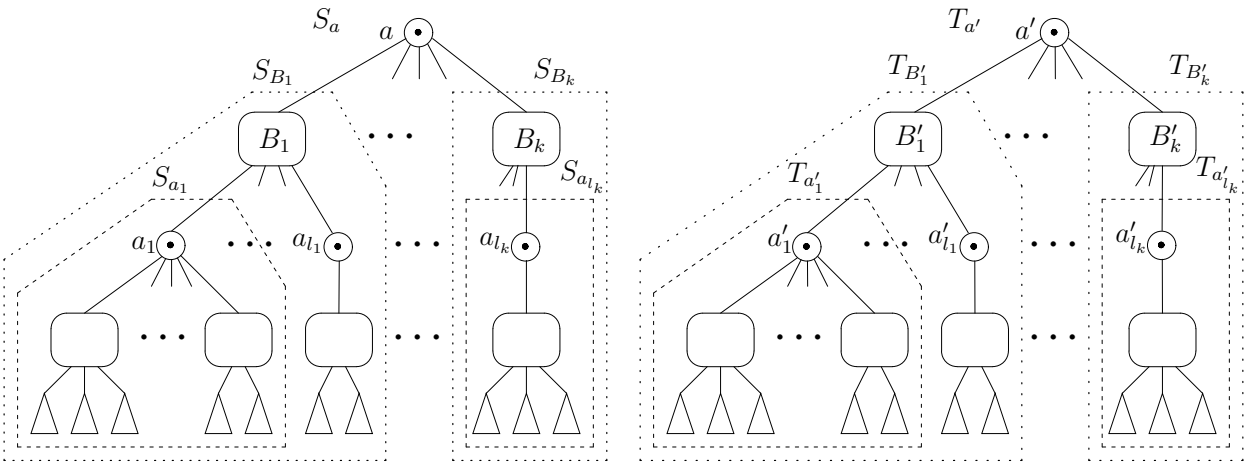

Figure 4: *Biconnected component trees.*

distinguish several cases below. The center of $\mathsf{T}(B)$, denoted by C, will play an important role thereby. In some of the cases we will show that the number of automorphisms of the center C is small. This already suffices for our purpose: in this case, for every edge as starting edge, we canonize the component C separately and construct a set of the edges A that lead to the minimum canon. Though the possible canons are polynomially many, the minimum ones are bounded by the number of automorphisms of C, which are small. Note, in this process, we do not recurse on the separating pairs or articulation points.

Now we take the first separating pair encountered in each of the canons obtained from edges in A as starting edges. This set of separating pairs forms the potential root nodes for $T(B)$, and hence its cardinality is bounded by the number of automorphisms of C. If B contains no separating pairs (i.e. B=C) then we take edges in A to compute the canon of B, recursively. We start our case analysis by considering properties of the center C of $\mathsf{T}(B)$.

- **The center C of $\mathsf{T}(B)$ is a separating pair:** We choose this separating pair as the root of $\mathsf{T}(B)$. Thus we have only one choice for the root, and the subtree rooted at B can be canonized in a unique way.
- **C is a triconnected component and a is not associated with C:** Let a be associated with a triconnected component R. We find the path from R to C in $\mathsf{T}(B)$ and find the separating pair closest to C on this path. This serves as the unique choice for the root of $\mathsf{T}(B)$.
- **a is associated with C and C is a cycle:** We canonize C for the two edges incident on a as starting edges, and a as the starting vertex. We construct these canons till a virtual edge is encountered in one or both of them. We choose the separating pairs corresponding to the first virtual edges encountered in these canons as the roots of $\mathsf{T}(B)$. Thus we get at most two choices for the root of $\mathsf{T}(B)$.

For the following cases, we assume that the center C is a 3-connected component and a is associated with C. We proceed with the case analysis according to the number l of articulation points in B besides a.

Case I: $l = 0$. B is a leaf node in S_a, it contains no articulation points besides a. We color a with a distinct color. In this case we can cycle through all separating pairs as root for $\mathsf{T}(B)$.

Case II: $l = 1$. If B has exactly one articulation point besides a, then we process this child a priori and store the result. We color a and a_1 with distinct colors and proceed with B as in the case of a leaf node.

Case III: $l \geq 2$. We distinguish two sub-cases.

1) **Some articulation point a_j in E_1 is not associated with C.** Let a_j be associated with a triconnected component $D \neq C$. Find the path from D to C in $\mathsf{T}(B)$ and select the separating pair node closest to C on this path. Thus a_j uniquely defines a separating pair. In the worst case, this may happen every a_j in E_1. Therefore, we get up to k_1 separating pairs as candidates for the root.
2) **All articulation points in E_1 are associated with C.** We distinguish three further sub-cases.
   a) $k_1 = k_2 = 1$. C has at least three vertices that are fixed by all its automorphisms (i.e. a and the articulation point with its subtree in E_1 and that in E_2). We will show in Corollary 4.7 below that C has at most one non-trivial automorphism in this case. Thus, we have at most two ways of choosing the root of $\mathsf{T}(B)$.
   b) $k_1 = 1$ **and** $k_2 \geq 2$. We process the child in E_1 a priori and store the result. We prove in Lemma 4.3 below that C can have at most $4k_2$ automorphisms in this case. Thus, we have at

most $4k_2$ ways of choosing the root of $\mathsf{T}(B)$.

c) $\boldsymbol{k_1 \geq 2}$. Again by Lemma 4.3 below, C can have at most $4k_1$ automorphisms. Thus, we have at most $4k_1$ ways of choosing the root of $\mathsf{T}(B)$.

Let $N = |S_B|$. The subtrees in the size class E_m clearly have size $\leq N/k_m$. Since the size classes are ordered according to increasing k_j's, the subtrees in E_j also have size $\leq N/k_m$ for all $j \geq m$. Therefore we have: in the sub-cases (1) and (2c) of case III above we use $O(\log k_1)$ space to keep track of which of the potential root edges is currently being used, and all subtrees are of size $\leq N/k_1$. The same holds with respect to k_2 in sub-case (2b). This will suffice to bound the total space used for the subtree rooted at B by $O(\log N)$.

The following lemma gives a relation between the size of the smallest color class and the number of automorphisms for a 3-connected planar graph with one distinctly colored vertex.

Lemma 4.3 *Let G be a 3-connected planar graph with colors on its vertices such that one vertex a is colored distinctly, and let $k \geq 2$ be the size of the smallest color class apart from the one which contains a. G has $\leq 4k$ automorphisms.*

To prove Lemma 4.3, we refer to the following results.

Lemma 4.4 [Bab95](P. Mani) *Every triconnected planar graph G can be embedded on the 2-sphere as a convex polytope P such that the automorphism group of G coincides with the automorphism group of the convex polytope P formed by the embedding.*

Lemma 4.5 [AD04], [Bab95], [Art96] *For any convex polytope other than tetrahedron, octahedron, cube, icosahedron, dodecahedron, the automorphism group is the product of its rotation group and $(1, \tau)$, where τ is a reflection. The rotation group is either C_k or D_k, where C_k is the cyclic group of order k and D_k is the dihedral group of order $2k$.*

Proof of Lemma 4.3: Let H be the subgroup of the rotation group, which permutes the vertices of the smallest color class among themselves. Then H is cyclic since the rotation group is cyclic. Let H be generated by a permutation π.

Notice that a non-trivial rotation of the sphere fixes exactly two points of the sphere, namely the end-points of the axis of rotation. Then, the following claim holds.

Claim 4.6 *In the cycle decomposition of π each non-trivial cycle has the same length.*

Proof of Claim 4.6: Suppose π_1, π_2 are two non-trivial cycles of lengths $p_1 < p_2$ respectively in the cycle decomposition of π. Then π^{p_1} fixes all elements of π_1 but not all elements of π_2. Thus $\pi^{p_1} \in H$ cannot be a rotation of the sphere which contradicts the definition of H. □

As a consequence, the order of H is bounded by k, since the length of any cycle containing one of the k colored points is at most k. □

This leads to the following corollary, which justifies sub-case (2a) of case III.

Corollary 4.7 *Let G be a 3-connected planar graph with at least 3 colored vertices, each having a distinct color. Then G has at most one non-trivial automorphism.*

Proof: An automorphism of G has to fix all the colored vertices. Consider the embedding of G on a 2-sphere. The only possible symmetry is a reflection about the plane containing the colored vertices, which leads to exactly one non-trivial automorphism. □

Note, if the triconnected component C is one of the exceptions stated in Lemma 4.5, it implies that C has $O(1)$ size. Thus, we do not have to limit its number of possible minimum canons. The preceding discussion implies that if two biconnected component trees are equal for the isomorphism order for some choice of the root, then the corresponding graphs are isomorphic. The reverse direction clearly holds as well.

Theorem 4.8 *Given two connected planar graphs G and H, and their biconnected component trees S and T, then $G \cong H$ if and only if there is a choice of articulation points a, a' in G and H such that $S_a =_{\mathrm{B}} T_{a'}$.*

4.2. Space Complexity of the Isomorphism Order Algorithm

The first two steps of the isomorphism order algorithm can be computed in log-space as in Lindell's algorithm [Lin92]. To see, that step 3 can also be performed in log-space, we give an outline here. We highlight the differences needed in the analysis first.

When we compare biconnected components B and B' in the biconnected component tree then a typical query is of the form (s, r), where s is the chosen root of the triconnected component tree and r is the index of the edge in the canon, which is to be retrieved. If there are k choices for the root for the triconnected component trees of B and B', the base machine cycles through all of them one by one, keeping track of the minimum canon. This takes $O(\log k)$ space. From the discussion above, we know that the possible choices for the root can be restricted to $O(k)$, and that the subtrees rooted at the children of B have size $\leq |S_B|/k$, when $k \geq 2$. Hence the comparison of B and B' can be done in log-space in this case.

We compare the triconnected component trees $\mathsf{T}(B)$ and $\mathsf{T}(B')$ according to B and B'. When we compare triconnected components in $\mathsf{T}(B)$ and $\mathsf{T}(B')$ then the algorithm asks oracle queries to the triconnected planar graph canonization algorithm. The base machine retrieves edges in these canons one by one from the oracle and compares them. Two edges (a, b) and (a', b') are compared by first comparing a and a'. If both are articulation points then we check whether we reach them for the first time. If so, we compare the biconnected subtrees S_a and $S_{a'}$ rooted at a and a'. If these are equal then we look, whether (a, b) and (a', b') are separating pairs. If so, then we compare their triconnected subtrees. If these are equal then we proceed with the next edge e.g. (b, c) and continue in the same way.

When we go into recursion at an articulation point or a separating pair then similar as in the algorithm of Lindell, we have counters to distinguish between isomorphic subtrees and the isomorphism classes. For k isomorphic subtrees, counters of $O(\log k)$ bits are needed. When we go into recursion at a triconnected component or a biconnected component then the situation is more difficult. We give an outline for this now. More details can be found in a full version.

Limiting the number of recursive calls for articulation points. When we compare the triconnected component trees $\mathsf{T}(B)$ and $\mathsf{T}(B')$, respectively (see Figure 5), then a may occur in several components in $\mathsf{T}(B)$, because a can be part of a separating pair. We want to go into recursion on a to the subtree S_a only once. This will be either directly when we reach $\mathsf{T}(B)$, in the case that S_a is a large child of B, or at a uniquely defined point in $T(B)$. The first case will be described in more detail later.

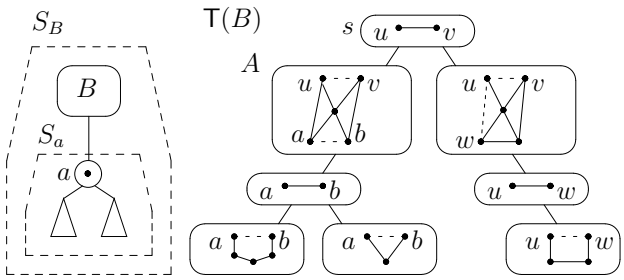

Figure 5: *A biconnected component tree S_B rooted at biconnected component B which has an articulation point a as child, which occurs in the triconnected component tree $\mathsf{T}(B)$ of B. In A and the other triconnected components the dashed edges are separating pairs.*

We explain now how to find the component A which contains the uniquely defined copy of a in $\mathsf{T}(B)$. From now on, we assume that we have the same situation for a' in $\mathsf{T}(B')$. If not, then we found an inequality. We define now a unique component A, where a is contained. We distinguish three cases. If a occurs in the root separating pair of $\mathsf{T}(B)$, i.e. already at the beginning then we define A as the root. If a occurs only in separating pairs other than the root of $\mathsf{T}(B)$ then a occurs in all the component nodes which contain this separating pair. Because these nodes form a connected subtree of $\mathsf{T}(B)$, a triconnected component node is the closest to the root. Then, a is compared to a' before comparing the triconnected subtrees of the separating pairs. Also, if a does not occur in a separating pair then a occurs in a triconnected component A (respectively a' in A') first. In the comparison of A with A' we compare their canons bit-by-bit and go into recursion only at the first occurrence of a and a'. We call this occurrence of a the *reference copy of* a. We define the same for a', respectively. Note that the reference copy of a depends on the chosen root for $\mathsf{T}(B)$. Hence, the position of the reference copy of a can be found without storing extra information on the work-tape. For the isomorphism ordering algorithm, the reference copy of a in $\mathsf{T}(B)$ can be found in log-space.

Large children. As in the case of biconnected graphs in Section 4, we deviate from the algorithm described so far in the case that the recursion would lead to a large child. Large subtrees are again treated a priori.

However, the notion of a large child is somewhat subtle here. We already defined the size of biconnected component trees S_a and S_B with an articulation point a or a biconnected component B as root. A *large child* of such a tree of size N is a child with the corresponding subtree of size $\geq N/2$.

Now consider the triconnected component tree $\mathsf{T}(B)$ of B. We said that we go into recursion to S_a only once, namely either when we reach the reference copy of a or even before in the following case: let a be an articulation point in B and let A be the node in $\mathsf{T}(B)$ that contains the reference copy of a. Then it might be the case that S_a is a large child of S_B *and* of S_A (i.e. subtree of S_B rooted at A). In this case we visit S_a when we reach B, i.e. before we start to compute the root for $\mathsf{T}(B)$. Then, when we reach the reference copy of a in A, we first check whether we already visited S_a. In this case the comparison result (with some large child $S_{a'}$ of B') is already stored on the work-tape and we do not visit S_a a second time.

Consequently, we consider S_a as a subtree only at the place where we go into recursion to S_a. Figure 6 shows an example. Basically, we have to distinguish large children with respect to the biconnected and the triconnected component trees if we want to guarantee a log-space bound.

Analysis of the space requirement. We analyze the comparison algorithm when it compares subtrees rooted at separating pairs and subtrees rooted at articulation points. For the analysis, the recursion goes here from depth d to $d+2$ of the trees. Observe, that large children are handled a priori at any level of the trees. We set up the following recursion equation for the space requirement of our algorithm.

$$\mathcal{S}(N) = \max_j \ \mathcal{S}\left(\frac{N}{k_j}\right) + O(\log k_j),$$

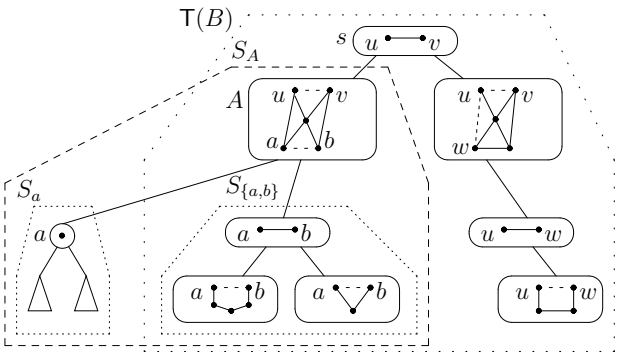

Figure 6: *The triconnected component tree $\mathsf{T}(B)$ of the biconnected component B. The triconnected component A contains the reference copy of articulation point a. If S_a is not a large child of B, then the subtree S_A consists of the subtree of $\mathsf{T}(B)$ rooted at A* **and** *the subtree S_a. In contrast, S_a is not part of the subtree $S_{\{a,b\}}$ because it does not contain the reference copy of a.*

where $k_j \geq 2$ (for all j) are the values mentioned above in the corresponding cases. Hence, $\mathcal{S}(N) = O(\log N)$.

For the explanation of the recursion equation it is helpful to imagine that we have two work-tapes. We use the first work-tape when we go into recursion at articulation point nodes, and the second work-tape when we go into recursion at separating pair nodes. The total space needed is the sum of the space of the two work-tapes. At an articulation point node a, the value k_j is the number of elements in the j-th size class among the children $B_1, \ldots, B_k$ of a. We store $O(\log k_j)$ bits and recursively consider subtrees of size $\leq N/k_j$. At a separating pair node s the value k_j is the number of elements in the j-th isomorphism class among the children $G_1, \ldots, G_k$ of s. We store $O(\log k_j)$ bits and recursively consider subtrees of size $\leq N/k_j$. This finishes the complexity analysis.

Theorem 4.9 *The isomorphism order between two planar graphs can be computed in log-space.*

We immediately get the following corollary.

Corollary 4.10 *Planar graph isomorphism is in log-space.*

To get an intuition, the following example describes the canon for a triconnected component tree. Let $l(S, a)$ be the canonical list of edges for the tree S_a of Figure 4. The canon is based on the string representation for trees and the canons of biconnected components of the previous section. Let $l(B_i, a)$ be the canonical list of edges of the biconnected component B_i (i.e. the canonical list of $\mathsf{T}(B_i)$ with a the parent articulation point). Let $a_1, \ldots, a_{l_1}$ be the order of the reference copies of articulation points as they occur in the canon of $\mathsf{T}(B_i)$. We get the following canonical list for S_a.

$$
\begin{aligned}
l(S,a) &= [\,(a)\; l(S_{B_1},a)\; \ldots\; l(S_{B_k},a)\,] \text{ with}\\
l(S_{B_1},a) &= [\,l(B_1,a)\; l(S_{a_1},a_1)\; \ldots\; l(S_{a_{l_1}},a_{l_1})\,]\\
&\vdots\\
l(S_{B_k},a) &= [\,l(B_k,a)\; l(S_{a_{l_k}},a_{l_k})\,]
\end{aligned}
$$

Finally, the canon for the connected planar graph G is obtained then by removing the brackets, by giving vertices new labels in the order of their first occurrences and finally, by ordering the edges of G lexicographically according to the new labels. These tasks can be done in log-space. We finish the proof of Theorem 1.1.

5. Conclusion

In this paper, we improve the known upper bound for isomorphism and canonization of planar graphs from AC^1 to L. This implies L-completeness for this problem, thereby settling its complexity. An interesting question is to extend it to other important classes of graphs.

6. Acknowledgement

We thank V. Arvind, Bireswar Das, Raghav Kulkarni, Meena Mahajan, K. V. Subrahmanyam, Jacobo Torán and the anonymous referees for helpful comments and discussions.

References

[AD04] V. Arvind and Nikhil Devanur. Symmetry breaking in trees and planar graphs by vertex coloring. In *The Nordic Combinatorial Conference (NORCOM)*, 2004.

[ADK08] V. Arvind, Bireswar Das, and Johannes Köbler. A logspace algorithm for partial 2-tree canonization. In *Computer Science Symposium in Russia (CSR)*, pages 40–51, 2008.

[ADR05] Eric Allender, Samir Datta, and Sambuddha Roy. The directed planar reachability problem. In *Proceedings of the 25th annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 238–249, 2005.

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

[AM00] Eric Allender and Meena Mahajan. The complexity of planarity testing. In *Proceedings of the 17th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 87–98, 2000.

[Art96] Michael Artin. Algebra. *Prentice Hall*, 1996.

[Bab95] László Babai. Automorphism groups, isomorphism, reconstruction. *Handbook of combinatorics*, 2:1447–1540, 1995.

[BHZ87] Ravi B. Boppana, Johan Hastad, and Stathis Zachos. Does co-NP have short interactive proofs? *Information Processing Letters*, 25(2):127–132, 1987.

[BL83] László Babai and Eugene M. Luks. Canonical labeling of graphs. In *Proceedings of the fifteenth annual ACM Symposium on Theory of Computing (STOC)*, pages 171–183, 1983.

[BTV07] Chris Bourke, Raghunath Tewari, and N.V. Vinodchandran. Directed planar reachability is in unambiguous log-space. In *IEEE Conference on Computational Complexity (CCC)*, pages 217–221, 2007.

[Bus97] Samuel R. Buss. Alogtime algorithms for tree isomorphism, comparison, and canonization. In *Proceedings of the 5th Kurt Gödel Colloquium on Computational Logic and Proof Theory*, pages 18–33, 1997.

[Coo85] Stephen A. Cook. A taxonomy of problems with fast parallel algorithms. *Information and Control*, 64(1-3):2–22, 1985.

[DLN08] Samir Datta, Nutan Limaye, and Prajakta Nimbhorkar. 3-connected planar graph isomorphism is in log-space. In *Proceedings of the 28th annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 153–162, 2008.

[HT73] John E. Hopcroft and Robert E. Tarjan. Dividing a graph into triconnected components. *SIAM Journal on Computing*, 2(3):135–158, 1973.

[HT74] John E. Hopcroft and Robert Tarjan. Efficient planarity testing. *Journal of the ACM*, 21(4):549–568, 1974.

[HW74] John E. Hopcroft and J.K. Wong. Linear time algorithm for isomorphism of planar graphs (preliminary report). In *Proceedings of the 6th annual ACM Symposium on Theory of Computing (STOC)*, pages 172–184, 1974.

[JKMT03] Birgit Jenner, Johannes Köbler, Pierre McKenzie, and Jacobo Torán. Completeness results for graph isomorphism. *Journal of Computing and System Sciences*, 66(3):549–566, 2003.

[KHC04] Jacek P. Kukluk, Lawrence B. Holder, and Diane J. Cook. Algorithm and experiments in testing planar graphs for isomorphism. *Journal of Graph Algorithms and Applications*, 8(2):313–356, 2004.

[KST93] Johannes Köbler, Uwe Schöning, and Jacobo Torán. *The Graph Isomorphism Problem*. Birkhäuser, 1993.

[Lin92] Steven Lindell. A logspace algorithm for tree canonization (extended abstract). In *Proceedings of the 24th annual ACM Symposium on Theory of Computing (STOC)*, pages 400–404, 1992.

[Mac37] Saunders Maclane. A structural characterization of planar combinatorial graphs. *Duke Mathematical Journal*, 3:460–472, 1937.

[MJT98] Pierre McKenzie, Birgit Jenner, and Jacobo Torán. A note on the hardness of tree isomorphism. In *Proceedings of the 13th Annual IEEE Conference on Computational Complexity (CCC)*. IEEE Computer Society, 1998.

[MR91] Gary L. Miller and John H. Reif. Parallel tree contraction part 2: further applications. *SIAM Journal on Computing*, 20(6):1128–1147, 1991.

[RA97] Klaus Reinhardt and Eric Allender. Making nondeterminism unambiguous. In *IEEE Symposium on Foundations of Computer Science*, pages 244–253, 1997.

[Rei05] Omer Reingold. Undirected st-connectivity in log-space. In *Proceedings of the 37th annual ACM Symposium on Theory of Computing (STOC)*, pages 376–385, 2005.

[RR94] Vijaya Ramachandran and John H. Reif. Planarity testing in parallel. *Journal of Computer and System Sciences*, 49:517–561, 1994.

[Sch88] Uwe Schöning. Graph isomorphism is in the low hierarchy. *Journal on Computing and System Sciences*, 37(3):312–323, 1988.

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

[TW08] Thomas Thierauf and Fabian Wagner. The isomorphism problem for planar 3-connected graphs is in unambiguous logspace. In *25th International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 633–644, 2008.

[TW09] Thomas Thierauf and Fabian Wagner. Reachability in $k_{3,3}$-free graphs and k_5-free graphs is in unambiguous log-space. Technical Report TR09-029, Electronic Colloquium on Computational Complexity (ECCC), 2009.

[Ver07] Oleg Verbitsky. Planar graphs: Logical complexity and parallel isomorphism tests. In *24th International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 682–693, 2007.

[Wag07] Fabian Wagner. Hardness results for tournament isomorphism and automorphism. In *32nd International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 572–583, 2007.

[Wei66] Louis Weinberg. A simple and efficient algorithm for determining isomorphism of planar triply connected graphs. *Circuit Theory*, 13:142–148, 1966.

[Whi33] Hassler Whitney. A set of topological invariants for graphs. *American Journal of Mathematics*, 55:235–321, 1933.