Graph Isomorphism for $K_{3,3}$-free and K_5-free graphs is in Log-space

Samir Datta[1] Prajakta Nimbhorkar[2]

Thomas Thierauf[3] Fabian Wagner[4*]

[1] Chennai Mathematical Institute
sdatta@cmi.ac.in
[2] The Institute of Mathematical Sciences
prajakta@imsc.res.in
[3] Fak. Elektronik und Informatik, HTW Aalen
[4] Institut für Theoretische Informatik,
Universität Ulm, 89073 Ulm
{thomas.thierauf,fabian.wagner}@uni-ulm.de

March 27, 2010

Abstract

Graph isomorphism is an important and widely studied computational problem, with a yet unsettled complexity. However, the exact complexity is known for isomorphism of various classes of graphs. Recently [DLN+09] proved that planar graph isomorphism is complete for log-space. We extend this result of [DLN+09] further to the classes of graphs which exclude $K_{3,3}$ or K_5 as a minor, and give a log-space algorithm.

Our algorithm for $K_{3,3}$ minor-free graphs proceeds by decomposition into triconnected components, which are known to be either planar or K_5 components [Vaz89]. This gives a triconnected component tree similar to that for planar graphs. An extension of the log-space algorithm of [DLN+09] can then be used to decide the isomorphism problem.

For K_5 minor-free graphs, we consider 3-connected components. These are either planar or isomorphic to the four-rung mobius ladder on 8 vertices or, with a further decomposition, one obtains planar 4-connected components [Khu88]. We give an algorithm to get a unique decomposition of K_5 minor-free graphs into bi-, tri- and 4-connected components, and construct trees, accordingly. Since the algorithm of [DLN+09] does not deal with four-connected component trees, it needs to be modified in a quite non-trivial way.

*Supported by DFG grants TO 200/2-2.

1 Introduction

The graph isomorphism problem GI consists of deciding whether there is a bijection between the vertices of two graphs, which preserves the adjacencies among vertices. It is an important problem with a yet unknown complexity.

The problem is clearly in NP and is also in SPP [AK06]. It is unlikely to be NP-hard, because otherwise the polynomial time hierarchy collapses to the second level [BHZ87, Sch88]. As far as lower bounds are concerned, GI is hard for DET [Tor04], which is the class of problems NC^1-reducible to the determinant [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].

The complexity of isomorphism is settled for trees [Lin92, MJT98], partial 2-trees [ADK08], and for planar graphs [DLN$^+$09]. We extend the result of [DLN$^+$09] to isomorphism of $K_{3,3}$ and K_5 minor-free graphs. The previously known upper bound for these graph classes is P due to [Pon91]. Both of these graph classes include planar graphs, and hence are considerably larger than the class of planar graphs.

We consider undirected graphs without parallel edges and loops, also known as *simple* graphs. For directed graphs or graphs with loops and parallel edges, there are log-space many-one reductions to simple undirected graphs (cf. [KST93]). Our log-space algorithm relies on the following properties of $K_{3,3}$ and K_5 minor-free graphs:

- The 3-connected components of $K_{3,3}$ minor-free graphs are either planar graphs or complete graphs on 5 vertices i.e. K_5's [Vaz89].
- The 3-connected components of K_5 minor-free graphs are either planar or V_8's (where V_8 is a four-rung mobius ladder on 8 vertices) or the following holds. The 4-connected components of the remaining non-planar 3-connected components are planar [Khu88].

Planar 3-connected components have two embeddings on the sphere (cf. [Whi33]). The embedding is given roughly as follows: For each vertex, its neighbours are ordered by a cyclic permutation. The second embedding for the 3-connected component is obtained by the mirror image, (i.e. consider the reverse permutations for all vertices). Allender and Mahajan [**?**] showed that an embedding of a planar graph can be computed in log-space. These facts are used in [**?**] and [DLN08] to show that the canonization of 3-connected planar graphs is in L. If one edge is fixed, then there are at most four possibilities (i.e. two embeddings and two orientations of the starting edge) to traverse a triconnected component in log-space. Such a traversal gives a unique order of the vertices. For a cycle we have two possibilities to canonize, i.e. traverse the cycle from the fixed edge in both directions.

There is a related result where reachability in $K_{3,3}$ and K_5 minor-free graphs are reduced to reachability in planar graphs under log-space many-one reductions [TW09]. The basic idea is that the non-planar components are transformed into new planar components, carefully copying subgraphs in a recursive manner, such that the graph remains polynomial in size of the input graph. This technique is designed to keep the reachability conditions unchanged but it is not applicable for isomorphism testing.

We give a log-space algorithm to get these decompositions in a *canonical* way. From these decompositions, we construct the biconnected and triconnected component trees for $K_{3,3}$

minor-free graphs, and extend the log-space algorithm of [DLN+09] to detect an isomorphism between two given $K_{3,3}$ minor-free graphs. The isomorphism of K_5 minor-free graphs is more complex, as in addition to biconnected and triconnected component trees, it also has four-connected component trees. This needs considerable modifications and new ideas. We also give log-space algorithms for canonization of these graphs.

The rest of the paper is organized as follows: Section 2 gives the necessary definitions and background. Section 3 gives the decomposition of $K_{3,3}$ minor-free graphs and proves the uniqueness of such decompositions. We give a log-space algorithm for isomorphism testing and canonization for these graphs. In Section 4 we extend the results to K_5 minor-free graphs.

2 Definitions and Notations

We consider undirected and loop-free graphs $G = (V, E)$ with vertices V and edges E. For $U \subseteq V$ let $G \setminus U$ be the *induced subgraph* of G on $V \setminus U$.

A connected graph G is called *k-connected* if one has to remove $\geq k$ vertices to disconnect G. In a k-connected graph G there are k vertex-disjoint paths between any pair of vertices in G. A 1-connected graph is simply called *connected* and a 2-connected graph *biconnected*.

In a k-connected graph G, a set $S \subseteq V$ with $|S| = k$ is called a *k-separating set*, if $G \setminus S$ is disconnected. The vertices of a k-separating set are called *articulation point* (or *cut vertex*) for $k = 1$, *separating pair* for $k = 2$, and *separating triple* for $k = 3$.

Let S be a separating set in G, let C be a connected component in $G \setminus S$, and $S' \subseteq S$ be those vertices from S connected to $V(C)$ in G. A *split component* of S in G is the induced subgraph of G on vertices $V(C)$ where we add the vertices of S' and edges connecting C with S', and *virtual edges* between all pairs of S'. More formally, the edges of the split component are

$$E(C) \ \cup \ \{\, \{u, v\} \mid u \in V(C),\ v \in S' \,\} \ \cup \ \{\, \{u, v\} \mid u, v \in S' \,\}.$$

The last set of edges might not have been edges in G and are called *virtual edges*.

Definition 2.1 The biconnected component tree. *We define nodes for the biconnected components and articulation points. There is an edge between a* biconnected component node *and an* articulation point node *if the articulation point is contained in the corresponding component. The resulting graph is a tree, the* biconnected component tree $\mathcal{T}^{\mathtt{B}}(G)$ *(see Figure 1).*

Next we consider biconnected graphs and their decomposition into 3-connected components along separating pairs. If a separating pair $\{a, b\}$ is connected by only two vertex-disjoint paths, then a and b lie on a cycle. As we will see below, it is not necessary to decompose cycles any further. Instead, we maintain them as a special component. Therefore we decompose a biconnected graph only along separating pairs which are connected by at least three disjoint paths.

Definition 2.2 *A separating pair $\{a, b\}$ is called* 3-connected *if there are three vertex-disjoint paths between a and b in G.*

The decomposition is described next.

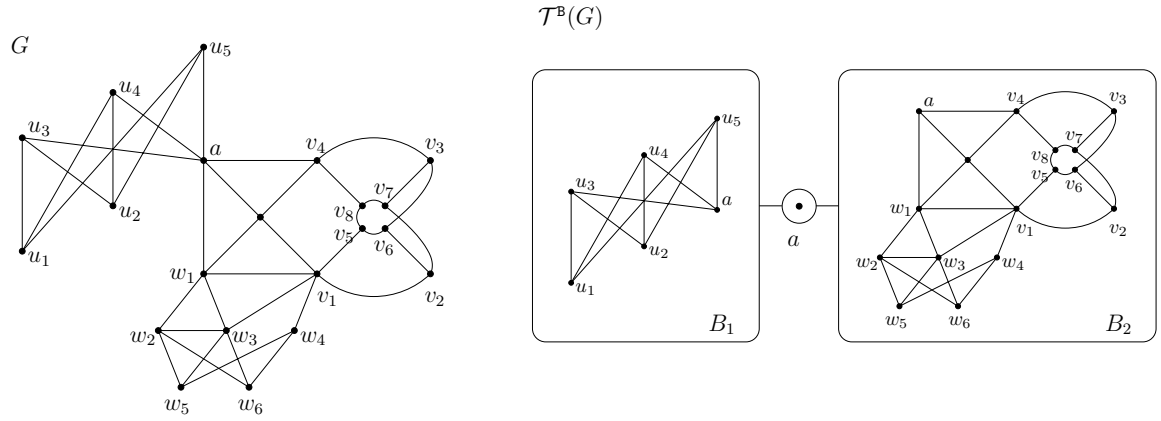

Figure 1: *Decomposition of a K_5-free graph G into biconnected components B_1 and B_2 and the corresponding biconnected component tree $\mathcal{T}^{\mathsf{B}}(G)$.*

Definition 2.3 *Two vertices u, v belong to a* 3-connected component *or a* cycle component *if there is no 3-connected separating pair which separates u from v. The components formed are the induced subgraphs of G, where we introduce in addition a special edge, called* virtual edge*, which connects the 3-connected separating pair u, v. Furthermore we add a* 3-bond component *for $\{u, v\}$. A 3-bond is a pair of vertices connected by 3 edges. A* triconnected component *is a 3-connected component, a cycle component or a 3-bond component. Accordingly, a graph is* triconnected *if it is either 3-connected, a cycle or a 3-bond.*

Based on the triconnected components, we define the triconnected component tree.

Definition 2.4 The triconnected component tree. *Define nodes for the triconnected components and 3-connected separating pairs for a biconnected graph G. There is an edge between a* triconnected component node *and a* separating pair node *if the separating pair is contained in the corresponding component. The resulting graph is a tree, the* triconnected component tree $\mathcal{T}^{\mathsf{T}}(G)$ *(see Figure 2).*

For a component tree T, we define $\mathsf{graph}(T) = G$ as the graph that has the component tree T.

Definition 2.5 The size of a tree and large children. *The* size of an individual component node C *of T is the number of vertices in C. The vertices of separating sets are counted in 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. Note that the size of T is at least as large as the number of vertices in* $\mathsf{graph}(T)$*. We also write T_C instead of just T when we want to make the root C of T explicit. Let D be child node of C. Then D is called a* large child *of C if $|T_D| > |T_C|/2$. The number of children of C is denoted by $\#C$.*

A graph H is a minor of a graph G if and only if H can be obtained from G by a finite sequence of edge-removal and edge-contraction operations. A *$K_{3,3}$-free graph* (*K_5-free graph*) is an undirected graph which does not contain a $K_{3,3}$ (or K_5) as a minor.

For two isomorphic graphs we write $G \cong H$. A *canon* for G is a sorted list of edges with renamed vertices $f(G)$, such that for all graphs G, H we have $G \cong H \Leftrightarrow f(G) = f(H)$. We

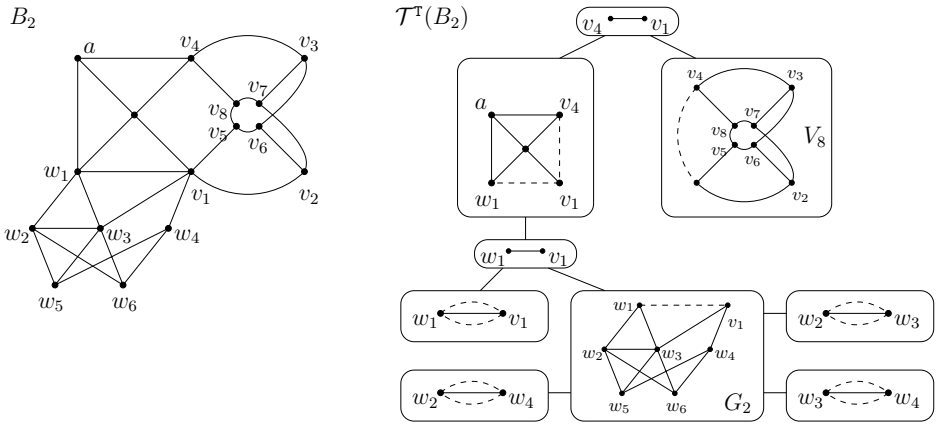

Figure 2: *Decomposition of the biconnected component B_2 and the corresponding triconnected component tree $\mathcal{T}^{\mathsf{T}}(B_2)$. There are leaf separating pairs connected to the triconnected component G_2. Virtual edges are indicated with dashed lines.*

also use *canon* with respect to some fixed starting edge. A *code* of G is the lexicographically sorted list of edges when given an arbitrary labeling of vertices.

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

3 $K_{3,3}$-free Graphs

We first consider the decomposition of biconnected $K_{3,3}$-free graphs into triconnected components. Then we show how to canonize $K_{3,3}$-free graphs.

3.1 Decomposition of $K_{3,3}$-free graphs

Tutte [Tut66] proved that the decomposition of a biconnected $K_{3,3}$-free graph into triconnected components is unique.

Lemma 3.1 [Tut66] *The decomposition of biconnected $K_{3,3}$-free graphs into triconnected components is unique, i.e. independent of the order in which the separating pairs are removed.*

Moreover, Asano [Asa85] proved that the decomposition has the following form.

Lemma 3.2 [Asa85] *Each triconnected component of a $K_{3,3}$-free graph is either planar or exactly the graph K_5.*

Miller and Ramachandran [MR87] showed that the triconnected component tree of a $K_{3,3}$-free graph can be computed in NC^2. Thierauf and Wagner [TW09] describe a construction that works in log-space. They showed, that it suffices to compute the triconnected components and recognize the K_5-components by running through all 5-sets and checking for each pair whether it is an edge or a 3-connected separating pair.

In the following lemma we extend Lemma 3.1 from biconnected $K_{3,3}$-free graphs to biconnected graphs and show that the decomposition can still be computed in logspace.

Lemma 3.3 *In a simple undirected biconnected graph G, the removal of 3-connected separating pairs gives a unique decomposition, irrespective of the order in which they are removed. This decomposition can be computed in log-space.*

Proof. Let G be a biconnected graph and let $s_1 = \{u_1, v_1\}$ and $s_2 = \{u_2, v_2\}$ be 3-connected separating pairs in G. It suffices to show that the decomposition of G after the removal of s_1 and s_2 does not depend on the order of their removal.

Claim 3.4 *s_2 is a 3-connected separating pair in a split component of s_1 and vice versa.*

Proof. Observe first that u_2 and v_2 must be in one split component of s_1 because the removal of s_2 can cut off at most 2 of the 3 vertex disjoint paths between u_2 and v_2.

Consider three paths between u_2 and v_2 which are pairwise vertex-disjoint except for their endpoints. If a path contains at most one of u_1 or v_1, then the path remains intact in a split component of s_1, because in the split components we have copies of u_1 and v_1.

If a path contains both of u_1 and v_1, then the part between u_1 and v_1 is split off. However, since we introduce a virtual edge $\{u_1, v_1\}$, we still have a path between u_2 and v_2 disjoint from the other two paths in one split component of s_1. This proves the claim. □

Two vertices end up in different split components only if they are separated by a 3-connected separating pair. Hence, removing split components from s_1 before or after removing those from s_2 has no effect on the resulting components. This shows that 3-connected separating pairs uniquely partition the graph into triconnected components. Thus they can be removed in parallel.

It remains to argue that the decomposition can be computed in log-space.

Claim 3.5 *In a biconnected graph G, 3-connected separating pairs can be detected in log-space.*

Proof. Separating pairs of G can easily be computed in log-space: find all pairs of vertices such that their removal from G disconnects the graph. This can be done with queries to reachability which is in L [Rei05]. Among those separating pairs we identify the 3-connected ones as follows. A separating pair $\{u, v\}$ in G is *not* 3-connected if either

- there are exactly two split components of $\{u, v\}$ (without attaching virtual edges) and both are not biconnected, or
- there is an edge $\{u, v\}$ and one such split component which is not biconnected.

To see this note that a split component which is not biconnected has an articulation point a. All paths from u to v through this split component must go through a. Hence there are no two vertex disjoint paths from u to v.

To check the above conditions we have to find articulation points in split components of $\{u, v\}$ This can be done in log-space with queries to reachability. This proves the claim. □

With the 3-connected separating pairs in hand the decomposition of a biconnected graph into its triconnected components can be done in log-space with appropriate reachability tests. This finishes the proof of the lemma. □

Corollary 3.6 *For a biconnected $K_{3,3}$-free graph, the triconnected planar components and K_5-components can be computed in log-space.*

3.2 Isomorphism order and canonization of $K_{3,3}$-free graphs

A $K_{3,3}$ free graph can be decomposed into triconnected components which are planar and K_5-components. We extend the algorithm of [DLN+09] for planar graphs to $K_{3,3}$-free graphs. It remains to explain how to handle the K_5-components in the isomorphism order of a $K_{3,3}$-free graph.

Isomorphism order for K_5-components

We consider a K_5 as a component for which we have a node in the triconnected component tree. There are 5! ways of labeling the vertices of a K_5, but the first two vertices will always be the vertices from the parent separating pair. There remain $2 \cdot 3! = 12$ ways of labeling the vertices. For example, one possibility to label the vertices a, b, c, d, e of a K_5 is by $1, 2, 3, 4, 5$, respectively. The canonical description of the graph with this labeling is defined as

$$(1,2),\ (1,3),\ (1,4),\ (1,5),\ (2,1),\ (2,3),\ \ldots,\ (5,4).$$

The canonical descriptions of all these labelings are candidates for the canon of a K_5. To keep notation short, we say *code* instead of *candidate for a canon*.

For each code, the isomorphism order algorithm starts with comparing two codes edge-by-edge. Thereby, it goes into recursion at child separating pairs and compares their subtrees. If the subtrees are not isomorphic, the larger code is eliminated. The comparison and the elimination of codes is done similarly as for the planar triconnected components in Datta et.al. [DLN+09]. The comparison takes $O(1)$ space on the work-tape to keep counters for the not eliminated codes.

The orientation that a K_5 component gives to its parent separating pair $\{a, b\}$ is defined as (a, b) (resp. (b, a)) if the majority of the minimum codes start with (a, b) (resp. (b, a)). If there is no majority for either direction, then G_0 does not give an orientation to the parent. In particular, if there exists a majority, then *all* minimum codes start with one of (a, b) or (b, a): if there is an automorphism which swaps a and b, then there are an equal number of minimum codes for (a, b) and (b, a), and hence there is no majority. If there is no such automorphism, then there cannot be minimum codes starting with (a, b) and (b, a).

Comparison of triconnected component trees

While comparing two triconnected component trees $S_{\{a,b\}}$ and $T_{\{a',b'\}}$ rooted at separating pairs $\{a, b\}$ and $\{a', b'\}$, we make cross-comparisons between equal sized subtrees rooted at their children G_i and H_j, respectively. These children are triconnected components. We start canonizing them in all possible ways. The number of possible codes depend on the type of the component. For example, for cycles we have two and for 3-connected planar components we have four codes.

Let C and C' be two codes to be compared. The base case is that G_i and H_j are leaf nodes and contain no further virtual edges. In this case we use the lexicographic order between C and C'. If G_i and H_j contain further virtual edges then these edges are specially treated in the bitwise comparison of C and C' the same way as we did for the comparison of triconnected components.

1. If a virtual edge is traversed in the construction of one of the codes C or C' but not in the other, then we define the one without the virtual edge to be the *smaller code.*

2. If C and C' encounter virtual edges $\{u, v\}$ and $\{u', v'\}$ corresponding to a child of G_i and H_j, respectively, we need to recursively compare the subtrees rooted at $\{u, v\}$ and $\{u', v'\}$. If we find in the recursion that one of the subtrees is smaller than the other, then the code with the smaller subtree is defined to be the *smaller code.*

3. If we find that the subtrees rooted at $\{u, v\}$ and $\{u', v'\}$ are equal then we look at the orientations given to $\{u, v\}$ and $\{u', v'\}$ by their children. This orientation, called the *reference orientation*, is defined below. If one of the codes traverses the virtual edge in the direction of its reference orientation but the other one not, then the one with the same direction is defined to be the *smaller code.*

We eliminate the codes which were found to be the larger codes in at least one of the comparisons. In the end, the codes that are not eliminated are the *minimum codes*. If we have the same minimum code for both, G_i and H_j, then we define $S_{G_i} =_{\mathtt{T}} T_{H_j}$.

Finally, we define the *orientation given to the parent separating pair of* G_i and H_j as the direction in which the minimum code traverses this edge. If the minimum codes are obtained for both choices of directions of the edge, we say that S_{G_i} and T_{H_j} are *symmetric about their parent separating pair*, and thus do not give an orientation.

Observe, that we do not need to compare the sizes and the degree of the root nodes of S_{G_i} and T_{H_j} in an intermediate step, as it is done when the root is a separating pair. That is, because the degree of the root node G_i is encoded as the number of virtual edges in G_i. The size of S_{G_i} is checked by the length of the minimal codes for G_i and when we compare the sizes of the children of the root node G_i with those of H_j.

Comparison of biconnected component trees

When comparing two biconnected components B and B', we compute their triconnected component trees and compare them. One important task is to find a small set of root separating pairs. In the case of planar 3-connected components a intricate case analysis is given. To extend the description from Datta et.al. [DLN$^+$09], it suffices to consider the case when the parent articulation point of B, say a, is located in the center C of the triconnected component tree $\mathcal{T}^{\mathtt{T}}(B)$.

- a **is associated with** C **and** C **is a** K_5**:** Since a is fixed there remain 4! codes. We construct these codes until a virtual edge is encountered in at least one of the codes. We choose the separating pairs corresponding to the first virtual edges encountered in these codes as the roots of $\mathcal{T}^{\mathtt{T}}(B)$. Because there are 4 edges incident to a and 6 edges not incident to a, we get at most 6 choices for the root of $\mathcal{T}^{\mathtt{T}}(B)$.

Together with the complexity analysis in [DLN$^+$09] we get:

Theorem 3.7 *Graph isomorphism for $K_{3,3}$-free graphs can be solved in log-space.*

Canonization of $K_{3,3}$-free graphs

We extend the canonization procedure as described for planar graphs, for the details we refer to [DLN$^+$09]. For a K_5-node G_0 and a parent separating pair $\{a, b\}$ we define the canon $l(G_0, a, b)$ exactly as if G_0 is a 3-connected component. That is, (a, b) followed by the canon

of G_0 and then the canons for the child separating pairs of G_0 in the order in which the child separating pairs appear in the canon of G_0.

Consider now the canonization procedure for the whole $K_{3,3}$-free graph, say G. A log-space transducer renames then the vertices according to their first occurrence in this list $l(G, a_0)$ (for some root articulation point a_0), to get the final tree-canon for the biconnected component tree. This canon depends upon the choice for the root a_0. Further log-space transducers cycle through all the articulation points as roots to find the minimum canon among them, then rename the vertices according to their first occurrence in the canon and finally, remove the virtual edges and delimiters to obtain a canon for G. We get

Theorem 3.8 *A $K_{3,3}$-free graph can be canonized in log-space.*

4 K_5-free Graphs

4.1 Decomposition of K_5-free graphs

We decompose the given K_5-free graph into 3-connected components. Datta et. al. [DLN+09] gave a unique decomposition of planar graphs into biconnected and these further into triconnected components which can be computed in log-space. Thierauf and Wagner proved that such decompositions can also be computed for K_5-free graphs in log-space.

Lemma 4.1 [TW09] *The triconnected component tree for a K_5-free biconnected graph can be computed in log-space.*

4.1.1 The structure of the 3-connected components

Wagner [Wag37] showed that the 3-connected K_5-free graphs can be constructed with a clique-sum operation from planar graphs and the four-rung Mobius ladder, also called V_8 (see Figure 3), a 3-connected graph on 8 vertices, which is non-planar because it contains a $K_{3,3}$.

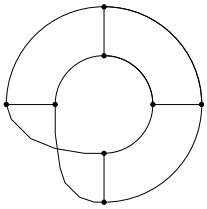

Figure 3: *The four-rung Mobius ladder, also called V_8.*

Definition 4.2 *Let G_1 and G_2 be two graphs each containing cliques of equal size. The* clique-sum of G_1 and G_2 *is a graph G formed from their disjoint union by identifying pairs of vertices in these two cliques to form a single shared clique, and then possibly deleting some of the clique edges. A k-*clique-sum *is a clique-sum in which both cliques have at most k vertices.*

Let $\mathcal{G}$ be some class of graphs. We define $\langle \mathcal{G} \rangle_k$ as the set of graphs G that can be constructed by repeatedly taking k-clique-sums starting from graphs in $\mathcal{G}$.

The class of K_5-free graphs can be decomposed as follows.

Theorem 4.3 [Wag37] *Let $\mathcal{C}$ be the class of all planar graphs together with the four-rung Mobius ladder V_8. Then $\langle\mathcal{C}\rangle_3$ is the class of all graphs with no K_5-minor.*

We make the following observations.

- If we build the 3-clique-sum of two planar graphs, then the three vertices of the joint clique are a separating triple in the resulting graph. Hence, the 4-connected components of a graph cannot be built by 3-clique sums of planar graphs and must all be planar.
- The V_8 is not planar, 3-connected and cannot be part of a 3-clique sum, because it does not contain a triangle as subgraph.

By Theorem 4.3 and these observations we have the following situation.

Corollary 4.4 (cf. [Khu88]) *A non-planar 3-connected component of a K_5-free undirected graph is either the V_8 or its 4-connected components are all planar.*

4.1.2 Decomposition into 4-connected components

Let $G \neq V_8$ be a non-planar 3-connected graph. Thierauf and Wagner [TW09] further decomposed such components into 4-connected components. The decomposition there is not unique up to isomorphism. We describe here a different way to decompose such a graph. The main difference is that we just decompose G at those separating triples which cause the non-planarity: consider a separating triple τ such that $G \setminus \tau$ splits into ≥ 3 connected components. Collapse these connected components into single vertices, and it is easy to see that G has a $K_{3,3}$ as minor.

Definition 4.5 *Let τ be a separating triple of a component G' of graph G. Then τ is called* 3-divisive *if in $G \setminus \tau$ the component G' is split into ≥ 3 connected components.*

Consider a $K_{3,3}$ and let τ be the three vertices of one side, and σ be the vertices of the other side. Then τ and σ are 3-divisive separating triples. If we remove σ then the remaining graph $K_{3,3} \setminus \sigma$ consists of three single vertices, namely τ. Each split component of σ is a K_4 which consists of the vertices of σ and one vertex of τ. Hence, τ is not a separating triple anymore in any of the split components of σ. We show next that the $K_{3,3}$ is an exception: if G is a K_5-free 3-connected graph different from $K_{3,3}$, and τ and σ are two 3-divisive separating triples, then τ is still a 3-divisive separating triple in a split component of σ.

Definition 4.6 *Let G be an undirected and 3-connected graph. Two 3-divisive separating triples $\tau \neq \sigma$ are* conflicting *if one of them, say τ, is not a 3-divisive separating triple in a split component of σ.*

Lemma 4.7 *Let G be an undirected and 3-connected graph. There is a conflicting pair of 3-divisive separating triples in G if and only if G is the $K_{3,3}$.*

Proof. Let $\tau = \{v_1, v_2, v_3\}$ and $\sigma = \{v'_1, v'_2, v'_3\}$ be two 3-divisive separating triples in G. The proof is based on the following claims.

Claim 4.8 *If all vertices of $\sigma \setminus \tau$ are contained in one split component of τ and vice versa (i.e. all vertices of $\tau \setminus \sigma$ are contained in one split component of σ) then τ and σ are not conflicting.*

This claim is clearly true because by the assumption, σ will be in one split component of τ, and conversely τ will be in one split component of σ. See also Figure 4 and Figure 5 (a). G_1, G_2, G_3 indicate split components of τ and G'_1, G'_2, G'_3 of σ. The split components G_i are obtained by attaching a copy of τ where each pair of vertices of τ is connected by a virtual edge. The resulting component is 3-connected.

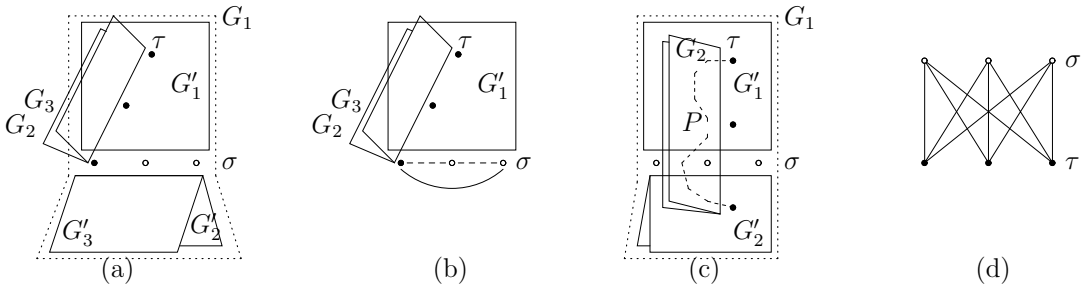

Figure 4: *(a) Two 3-divisive separating triples σ (shown as circles) and τ (shown as filled circles) which have one vertex in common, and their split components G'_1, G'_2, G'_3 and G_1, G_2, G_3, respectively.*
(b) The split component of σ where G'_2 and G'_3 are replaced by virtual edges (indicated with dashed lines).
(c) The situation is shown where σ and τ seem to be conflicting. But this situation cannot occur: σ would not be a separating triple because of path P.
(d) In a $K_{3,3}$, the triples σ and τ are conflicting.

The next claim shows that we can weaken the assumption in Claim 4.8.

Claim 4.9 *If all vertices of $\sigma \setminus \tau$ are contained in one split component of τ (or vice versa) then τ and σ are not conflicting.*

Proof. If all vertices of σ are contained in split component G_1 of τ, then there is another split component of τ, say G_2, that does not contain any vertices of $\sigma \setminus \tau$. Therefore G_2 is contained in one split component of σ, and τ is in the same split component of σ as G_2. In particular, τ must be in one split component of σ. Now the claim follows from Claim 4.8. □

From the proof of Claim 4.9 we deduce:

Claim 4.10 *If there is a split component of τ that contains* no *vertices of $\sigma \setminus \tau$ (or vice versa) then τ and σ are not conflicting.*

The assumption of Claim 4.10 is fullfilled in the following cases:

- τ or σ has ≥ 4 split components,
- there is a split component of σ that contains ≥ 2 vertices of τ (or vice versa), or
- $\tau \cap \sigma \neq \emptyset$.

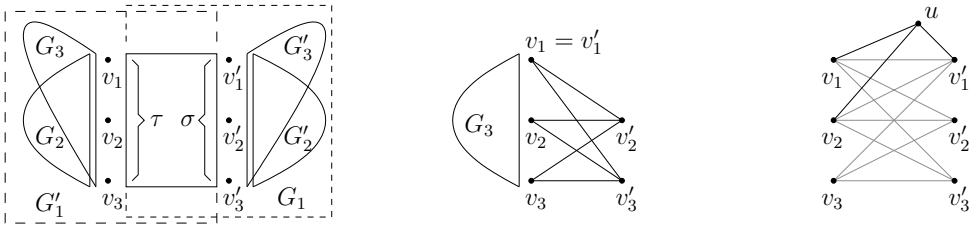

Figure 5: *(a) The 3-divisive separating triples τ, σ are shown schematically, as well as the split components G_i of τ and the G_i' of σ, indicated by dashed and solid shapes.*
(b) Separating triples τ, σ share the vertex $v_1 = v_1'$ and are pairwise connected.
(c) The split component G_1 of τ collapsed to one vertex u, except vertex v_1'.

The first two items are obvious. For the last item note that if τ and σ have a vertex in common, then the ≤ 2 vertices of $\sigma \setminus \tau$ cannot be in all split components of τ.

Hence the only case that remains where τ and σ might be conflicting is when τ and σ are disjoint, τ has precisely 3 split components and each component contains a vertex of σ, and vice versa. Let us consider this case.

If all the split components of τ and σ are K_4's then G must be a $K_{3,3}$ and τ and σ are conflicting. Hence we consider the case that there is a split component that is not a K_4, say component G_1 of τ. Component G_1 has ≥ 5 vertices: the 3 vertices of τ, a vertex of σ, say v_1', and at least one more vertex, say u, because G_1 is not a K_4 (see Figure 5(c)). Because G_1 is 3-connected, there are paths from u to at least two vertices of τ that do not go through v_1'. Therefore u and these vertices from τ will be in one split component of σ in G. Hence, because σ is 3-divisive, there is at least one split component which does not contain vertices from τ. By Claim 4.10, τ and σ are not conflicting. This finishes the proof of Lemma 4.7. □

4.1.3 The four-connected component tree

If we fix one 3-divisive separating triple as root then we get a uniqe decomposition for G up to isomorphism, even if G is the $K_{3,3}$. Hence, a log-space transducer cycles then through all possible triples τ of G and counts the number of split components in $G \setminus \tau$. If this number is ≥ 3 then τ is a 3-divisive separating triple.

We decompose the given graph G at 3-divisive separating triples and obtain split components which are free of 3-divisives separating triples. We denote such components as *four-connected*.

Two vertices u, v belong to a *four-connected component* if for all 3-divisive separating triples τ the following is true:

- at least one of u, v belongs to τ or
- there is a path from u to v in $G \setminus \tau$.

Note that a four-connected component is planar and 3-connected. We define a graph on these components and 3-divisive separating triples.

We define nodes for the four-connected components and 3-divisive separating triples. A *four-connected component node* is connected to a *3-divisive separating triple node* τ if the vertices of τ are also contained in the corresponding four-connected component. The resulting graph is a tree, the *four-connected component tree* $\mathcal{T}^{\mathtt{F}}(G)$. For an example see Figure 6. The node τ is incident to ≥ 3 four-connected component nodes. There is a 3-bond connected to τ_2, because the edge $\{w_2, w_3\}$ is present in T_2.

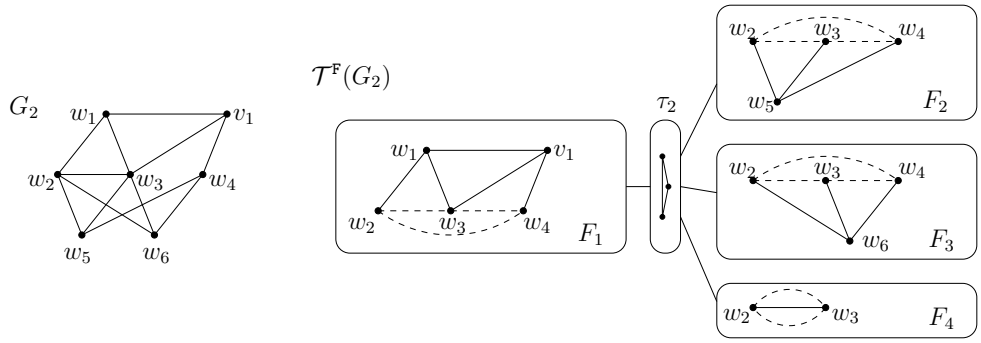

Figure 6: *The decomposition of G_2 into four-connected components $F_1, \ldots, F_4$ is shown, together with the four-connected component tree $\mathcal{T}^{\mathtt{F}}(G_2)$. The edges with both ends in the 3-divisive separating triple $\tau_2 = \{w_2, w_3, w_4\}$ are virtual edges indicated by dashed lines. There is a 3-bond connected to τ_2, because the edge $\{w_2, w_3\}$ is present in G_2.*

A unique decomposition of G into four-connected components can be computed in log-space, because every computation step can be queried to the reachability problem in undirected graphs which is in log-space [Rei05].

Theorem 4.11 *A unique decomposition of a 3-connected non-planar K_5-free graph (not the V_8) into four-connected components can be computed in log-space.*

4.2 Isomorphism order of K_5-free graphs

The difference to the planar case comes with the non-planar triconnected components. Therefore we mainly show how to compute an isomorphism order for these components. We start with an overview of the isomorphism order algorithm.

Overview of the isomorphism order algorithm

We extend the isomorphism order for planar graphs in [DLN$^+$09]. We define a total order $<_{\mathtt{T}}$ on triconnected component trees $S_{\{a,b\}}$ and $T_{\{a',b'\}}$ which are rooted at separating pairs $s = \{a, b\}$ and $t = \{a', b'\}$. We define $S_{\{a,b\}} <_{\mathtt{T}} T_{\{a',b'\}}$ if one of the following holds:

1. $|S_{\{a,b\}}| < |T_{\{a',b'\}}|$ or
2. $|S_{\{a,b\}}| = |T_{\{a',b'\}}|$ but $\#s < \#t$ or
3. $|S_{\{a,b\}}| = |T_{\{a',b'\}}|$, $\#s = \#t = k$, but $(S_{G_1}, \ldots, S_{G_k}) < (T_{H_1}, \ldots, T_{H_k})$ lexicographically (with respect to $<_{\mathtt{T}}$), where we assume that $S_{G_1} \leq_{\mathtt{T}} \cdots \leq_{\mathtt{T}} S_{G_k}$ and $T_{H_1} \leq_{\mathtt{T}} \cdots \leq_{\mathtt{T}} T_{H_k}$ are the ordered subtrees of $S_{\{a,b\}}$ and $T_{\{a',b'\}}$, respectively. For the isomorphism order

between the subtrees S_{G_i} and T_{H_i} we compare the types of the nodes first and then we compare lexicographically the codes 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. $|S_{\{a,b\}}| = |T_{\{a',b'\}}|$, $\#s = \#t = k$, $(S_{G_1}, \ldots, S_{G_k}) = (T_{H_1}, \ldots, T_{H_k})$ lexicographically (with respect to $<_{\mathrm{T}}$), 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 in [DLN$^+$09]. They define a direction for the edge $\{a, b\}$ for each isomorphism class. The orientation counters compute how often the edge $\{a, b\}$ is traversed in direction (a, b) resp. (b, a) in each isomorphism class. We adapt the notion also for the non-planar components. This is straight forward in the case of V_8-components. The other non-planar 3-connected components are further decomposed into four-connected component trees. In the minimum codes for these trees we take the direction of the first occurrence of $\{a, b\}$ in these codes for the orientation counters.

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.

For the isomorphism order of subtrees rooted at planar triconnected components we refer to [DLN$^+$09]. In the following we refine the isomorphism order for the new types of non-planar components.

4.2.1 Isomorphism order of subtrees rooted at V_8-components

Consider the subtree S_{G_i} rooted at a V_8-component node G_i. The isomorphism order algorithm makes comparisons with T_{H_j} rooted at H_j, accordingly.

To canonize G_i with the parent separating pair $\{a, b\}$, we could proceed as in the case of a K_5 on Page 6: there are $2 \cdot 6!$ ways of labeling the vertices of G_i. Because this is a constant, we can try all of them. We obtain the codes by arranging the edges in lexicographical order, according to the new vertex names. The minimum code gives the isomorphism order. If the minimum code occurs in G_i and H_j then they are found to be equal. In the rest of this subsection we take a closer look and show that we can do even better: we need to check ≤ 4 possibilities.

To rename the vertices, we use a Hamiltonian cycle in G_i starting at the parent separating pair $\{a, b\}$ of G_i. We rename the vertices in the order of their first occurrence on the Hamiltonian cycle. The code then starts with one of (a, b) or (b, a). It is defined as the list of all edges of G_i in lexicographical order with the new names.

The V_8 has 5 *undirected* Hamiltonian cycles (each corresponding to two directed Hamiltonian cycles). Consider the V_8 in Figure 7. Let $E' = \{\{v_1, v_5\}, \{v_2, v_6\}, \{v_3, v_7\}, \{v_4, v_8\}\}$. The edges in E' are contained in two simple cycles of length 4, whereas all the other edges are in only 1 such cycle.

- There is one Hamiltonian cycle which contains no edge from E':

$$D_0 = (v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8).$$

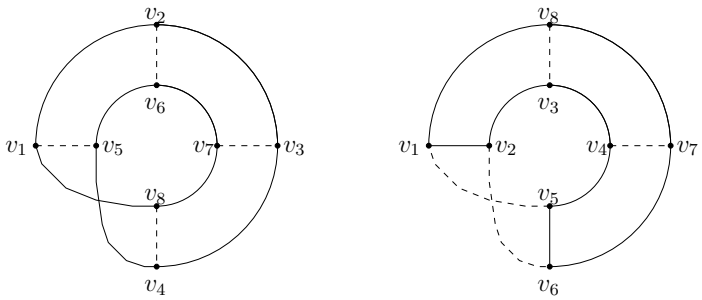

Figure 7: *(a) A V_8 component where solid edges indicate the Hamiltonian cycle D_0. (b) A V_8 component where solid edges indicate the Hamiltonian cycle $C_2[\{v_1, v_5\}, \{v_4, v_8\}]$.*

- There are 4 Hamiltonian cycles which have two edges from E':

$$C_2[\{v_2, v_6\}, \{v_3, v_7\}] = (v_1, v_2, v_6, v_5, v_4, v_3, v_7, v_8),$$
$$C_2[\{v_3, v_7\}, \{v_4, v_8\}] = (v_1, v_2, v_3, v_7, v_6, v_5, v_4, v_8),$$
$$C_2[\{v_4, v_8\}, \{v_5, v_1\}] = (v_1, v_2, v_3, v_4, v_8, v_7, v_6, v_5),$$
$$C_2[\{v_1, v_5\}, \{v_2, v_6\}] = (v_1, v_5, v_4, v_3, v_2, v_6, v_7, v_8).$$

- There are no further Hamiltonian cycles.

We distinguish the situation whether $\{a, b\} \in E'$ or not.

- **Case $\{a, b\} \notin E'$:** We use cycle D_0 to define the codes. We get two codes, one for each direction of $\{a, b\}$ we start with. For example, let $\{a, b\} = \{v_1, v_2\}$ and consider direction (v_1, v_2).

  Then the new names for the vertices are

vertex	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
new name	1	2	3	4	5	6	7	8

  The code is the enumeration of all edges in lexicographic order:

  $(\{1,2\},\ \{1,5\},\ \{1,8\},\ \{2,3\},\ \{2,6\},\ \{3,4\},\ \{3,7\},\ \{4,5\},\ \{4,8\},\ \{5,6\},\ \{6,7\},\ \{7,8\})$.

- **Case $\{a, b\} \in E'$:** Observe that each edge of E' occurs in exactly 2 of the 4 Hamiltonian cycles. Since we have two directions for each cycle, we get 4 codes. The direction of $\{a, b\}$ together with the subsequent edge determines exactly one Hamiltonian cycle. For example, let $\{a, b\} = \{v_1, v_5\}$ and consider the direction (v_1, v_5). Now we have two choices to proceed. If we choose (v_5, v_6) this determines the cycle $C_2[\{v_4, v_8\}, \{v_5, v_1\}]$ and we get the following new names for the vertices:

vertex	v_1	v_5	v_6	v_7	v_8	v_4	v_3	v_2
new name	1	2	3	4	5	6	7	8

  The code we obtain is

  $(\{1,2\},\ \{1,5\},\ \{1,8\},\ \{2,3\},\ \{2,6\},\ \{3,4\},\ \{3,8\},\ \{4,5\},\ \{4,7\},\ \{5,6\},\ \{6,7\},\ \{7,8\})$.

4.2.2 Isomorphism order of the 3-connected non-planar components $\neq V_8$

Let S_{G_i} and T_{H_j} be two triconnected component trees rooted at 3-connected non-planar component nodes G_i and H_j which are different to the V_8. Let $s = \{a, b\}$ and $t = \{a', b'\}$ be the parent separating pairs of G_i and H_j. The isomorphism order algorithm computes the isomorphism order between G_i and H_j and returns an orientation of the parent separating pairs s and t as described after the overview on page 13.

We partition G_i and H_j into their four-connected component trees $\mathcal{T}^{\mathtt{F}}(G_i)$ and $\mathcal{T}^{\mathtt{F}}(H_j)$. The isomorphism order algorithm invokes other algorithms as subroutines which we describe below. One algorithm computes a sufficiently small set of candidates for the root separating triples. Another algorithm handles the situations when we go into recursion at biconnected and triconnected subtrees. This is crucial in particular for large children with respect to the different kinds of subtrees.

In summary, we have two possible orientations of the parent separating pair, we have a sufficiently small number of root separating triples and for each root 6! possibilities to arrange the six directed edges of it. Hence, the number of tests depends up to a constant factor on the number of root separating triples. These tests are done via cross comparisons. The choice that leads to the minimum code is used for the isomorphism order.

We start with the main algorithm, the isomorphism order algorithm on four-connected component trees.

Overview of the isomorphism order algorithm

We define a total order $<_{\mathtt{F}}$ on four-connected component trees S_τ and $T_{\tau'}$ which are rooted at 3-divisive separating triples τ and τ'. We define $S_\tau \leq_{\mathtt{F}} T_{\tau'}$ if one of the following holds:

1. $|S_\tau| < |T_{\tau'}|$ or
2. $|S_\tau| = |T_{\tau'}|$, but $\#\tau < \#\tau'$ or
3. $|S_\tau| = |T_{\tau'}|$, $\#\tau = \#\tau' = k$, but $(S_{F_1}, \ldots, S_{F_k}) < (T_{F'_1}, \ldots, T_{F'_k})$ lexicographically, where we assume that $S_{F_1} \leq_{\mathtt{F}} \ldots \leq_{\mathtt{F}} S_{F_k}$ and $T_{F'_1} \leq_{\mathtt{F}} \ldots \leq_{\mathtt{F}} T_{F'_k}$ are the ordered subtrees of S_τ and $T_{\tau'}$, respectively. For the isomorphism order between the subtrees S_{F_i} and $T_{F'_i}$ we compare lexicographically the codes of F_i and F'_i and *recursively* the subtrees rooted at the children of F_i and F'_i. These children are again separating triple nodes, say τ_j and τ'_j, and the comparison is done together with an induced order on their edges: i.e. we compare S_{τ_j} with $T_{\tau'_j}$ and check whether τ_j is mapped onto τ'_j properly. This induced order comes from the arrangement of the virtual edges in the codes of F_i and F'_i which are compared currently. We describe this in more detail below (see Steps $(i), (ii)$ and (iii) on page 16).
4. $|S_\tau| = |T_{\tau'}|$, $\#\tau = \#\tau' = k$, and $(S_{F_1}, \ldots, S_{F_k}) = (T_{F'_1}, \ldots, T_{F'_k})$ lexicographically, but $\mathsf{ref\text{-}orient}(\tau) < \mathsf{ref\text{-}orient}(\tau')$, where $\mathsf{ref\text{-}orient}(\tau)$ is the *reference orientation of* τ which is defined below on page 18.

The reference orientation of a separating triple τ is a sequence of counters which consider each of the 6 directed edges that connect the vertices of τ. This is an extension of the orientation counters for separating pairs.

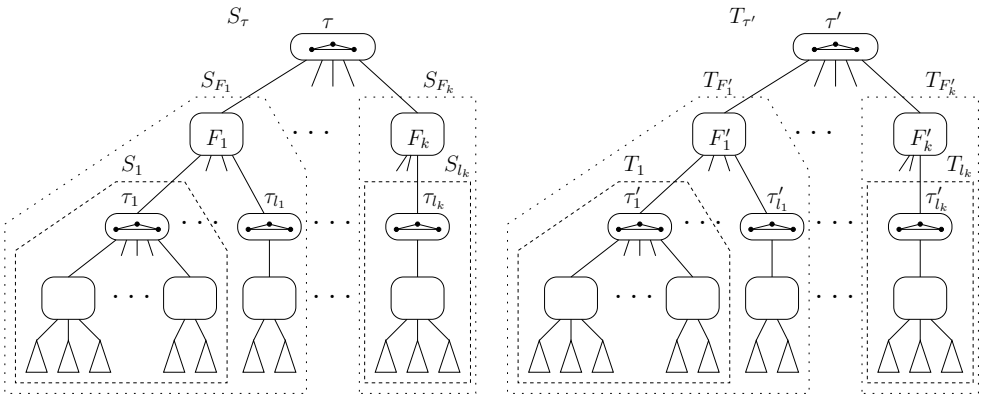

Figure 8: *The four-connected component trees.*

We say that two four-connected component trees S_τ and $T_{\tau'}$ are *equal according to the isomorphism order*, denoted by $S_\tau =_{\mathtt{F}} T_{\tau'}$, if neither $S_\tau <_{\mathtt{F}} T_{\tau'}$ nor $T_{\tau'} <_{\mathtt{F}} S_\tau$ holds.

We define the isomorphism order on four-connected component trees and distinguish the cases when the trees are rooted at four-connected component nodes or separating triple nodes.

Isomorphism order of two subtrees rooted at four-connected component nodes

We consider the isomorphism order of two subtrees S_{F_1} and $T_{F'_1}$ rooted at four-connected component nodes F_1 and F'_1, respectively, see Figure 8. To canonize a four-connected component F_1, we use the log-space algorithm from [DLN08]. Besides F_1, the algorithm gets as input a starting edge and a combinatorial embedding ρ of F_1. Let $\tau = \{a, b, c\}$ be the parent 3-divisive separating triple of F_1. There are three choices of selecting a starting edge, namely $\{a, b\}$, $\{b, c\}$, or $\{a, c\}$. Then there are two choices for the direction of each edge. Further, a 3-connected planar graph has two planar combinatorial embeddings [Whi33]. Hence, there are 12 possible ways to canonize F_1.

Let C and C' be two codes of F_1 and F'_1 respectively, to be compared. The base case is that F_1 and F'_1 are leaf nodes and therefore contain no further virtual edges. In this case we use the lexicographic order between C and C'.

Assume now, F_1 and F'_1 contain further virtual edges. The vertices of virtual edges belong to a child separating triple. These edges are specially treated in the bitwise comparison of C and C':

(i) If a virtual edge is traversed in the construction of one of the codes C or C' but not in the other, then we define the one without the virtual edge to be the smaller code.

(ii) If C and C' encounter all virtual edges at the same positions then we do the following. We order the child separating triples according to the positions of their virtual edges in the codes. Since a virtual edge of a separating triple τ might occur several times, we consider only the position of its first occurrence. This order is called the *edge position order of τ with respect to C*, denoted by $\mathsf{edge\text{-}pos}_C(\tau)$. That is, $\mathsf{edge\text{-}pos}_C(\tau)$ consists of the positions of the 6 directed virtual edges between the vertices of τ in the code C. We compare two such edge position orders lexicographically.

We sort the separating triples for C and C' according to their edge position order. Let $\mathsf{edge\text{-}pos}_C(\tau_1) < \cdots < \mathsf{edge\text{-}pos}_C(\tau_{l_1})$ be all the separating triples of C and $\mathsf{edge\text{-}pos}_{C'}(\tau'_1) < \cdots < \mathsf{edge\text{-}pos}_{C'}(\tau'_{l_1})$ those of C'. Then we compare $(\mathsf{edge\text{-}pos}_C(\tau_1), \ldots, \mathsf{edge\text{-}pos}_C(\tau_{l_1}))$ and $(\mathsf{edge\text{-}pos}_{C'}(\tau'_1), \ldots, \mathsf{edge\text{-}pos}_{C'}(\tau'_{l_1}))$. If they are not equal, then their lexicographic order defines the order of C and C'.

(iii) If in the comparison of C and C' we find that $(\mathsf{edge\text{-}pos}_C(\tau_1), \ldots, \mathsf{edge\text{-}pos}_C(\tau_{l_1})) = (\mathsf{edge\text{-}pos}_{C'}(\tau'_1), \ldots, \mathsf{edge\text{-}pos}_{C'}(\tau'_{l_1}))$, then we go into recursion and compare the corresponding subtrees rooted at the child separating triples. That is, we compare $(S_{\tau_1}, \ldots, S_{\tau_{l_1}})$ with $(T_{\tau'_1}, \ldots, T_{\tau'_{l_1}})$. If we encounter an inequality the first time, then this defines the order of C and C'.

We eliminate the codes which were found to be the larger codes in at least one of the comparisons. In the end, the codes that are not eliminated are the *minimum codes*. In general, if we have the same minimum code for both F_i and F'_j then we define $S_{F_i} =_{\mathtt{F}} T_{F'_j}$. The construction of the codes also defines an isomorphism between the subgraphs associated to S_{F_i} and $T_{F'_j}$, i.e. $\mathsf{graph}(S_{F_i}) \cong \mathsf{graph}(T_{F'_j})$. For a single four-connected component this follows from [DLN08]. If the trees contain several components, then our definition of $S_{F_i} =_{\mathtt{F}} T_{F'_j}$ guarantees that we can combine the isomorphisms of the components to an isomorphism between $\mathsf{graph}(S_{F_i})$ and $\mathsf{graph}(T_{F'_j})$.

Definition 4.12 *The orientation given to the parent separating triple τ of F_i is an* orientation graph *X_i, a complete graph with vertices $V(X_i) = \tau$. Accordingly, we define X'_j for τ' and F'_j. The 6 directed edges of X_i get colors as follows: we take each of the edges as starting edge for the isomorphism order on S_{F_i} and $T_{F'_j}$. This is done via cross comparisons, i.e. the isomorphism order starts with the edges in X_i and the corresponding edges in the orientation graph X'_j of $T_{F'_j}$. That is, we start with comparing and sorting the 6 codes in F_i and F'_j. If edge e as root leads to the r-th smallest code for some $1 \leq r \leq 6$, then e gets color r. Note that edges which lead to the same code will have the same color.*

Every subtree rooted at a four-connected component node gives an orientation graph to the parent separating triple. If the orientation is consistent, then we define $S_\tau =_{\mathtt{F}} T_{\tau'}$ and we will show that the corresponding graphs are isomorphic in this case.

For a single child subtree F_i of S_τ and F'_i of $T_{\tau'}$ this is true: an isomorphism ϕ that maps τ onto τ' can be extended to the whole of F_i which is mapped onto F'_i if and only if ϕ maps a directed edge e in τ onto an edge e' in τ' such that the code of F_i starting with e is the same as the code of F'_i starting with e'. When comparing the codes then we go into recursion at child separating triples (say τ_{i_0} of F_i and τ'_{j_0} of F'_j). This means that we also check whether ϕ can be extended to the corresponding subgraphs $\mathsf{graph}(S_{\tau_{i_0}})$ and $\mathsf{graph}(T_{\tau'_{j_0}})$. By induction we know whether $\mathsf{graph}(S_{\tau_{i_0}}) \cong \mathsf{graph}(T_{\tau'_{j_0}})$ such that τ_{i_0} is mapped onto τ'_{j_0}. By induction hypothesis, the reference orientation tells us all such possibilities to map τ_{i_0} onto τ'_{j_0}.

We consider next the situation where τ and τ' have more than one subtree.

Isomorphism order of subtrees rooted at separating triple nodes

Let τ and τ' be the roots of S_τ and $T_{\tau'}$ with the subtrees $S_{F_1}, \ldots, S_{F_k}$ and $T_{F'_1}, \ldots, T_{F'_k}$. We partition the subtrees into isomorphism classes. The isomorphism order involves the

comparison of the *orientations* given by the corresponding isomorphism classes defined as follows:

We first order the subtrees, say $S_{F_1} \leq_{\mathtt{F}} \cdots \leq_{\mathtt{F}} S_{F_k}$ and $T_{F'_1} \leq_{\mathtt{F}} \cdots \leq_{\mathtt{F}} T_{F'_k}$, and verify that $S_{F_i} =_{\mathtt{F}} T_{F'_i}$ for all i. If we find an inequality then the one at the smallest index i defines the isomorphism order between S_τ and $T_{\tau'}$. Now assume that $S_{F_i} =_{\mathtt{F}} T_{F'_i}$ for all i. Inductively, the corresponding split components are isomorphic, i.e. $\mathsf{graph}(S_{F_i}) \cong \mathsf{graph}(T_{F'_i})$ for all i.

The next comparison concerns the orientation of τ and τ'. Assume that S_τ and $T_{\tau'}$ are not large children of their parent nodes. The case of large children is treated separately below.

We already explained above the orientation graph X_i for τ defined by each of the S_{F_i}'s. We define a reference orientation for τ from all these orientations as a sequence of counters as follows: we partition $(S_{F_1}, \ldots, S_{F_k})$ into classes of isomorphic subtrees, say $I_1, \ldots, I_p$ for some $p \leq k$ and where the classes are ordered according to $<_{\mathtt{F}}$. For each directed edge e in τ, each isomorphism class I_j and each color r we define a counter $c_{e,j,r}$ as the number of times that e has color r in an orientation graph in the class I_j:

$$c_{e,j,r} = |\{\, i \mid e \text{ has color } r \text{ in } X_i, \text{ for } S_{F_i} \in I_j \,\}|.$$

The *reference orientation of* τ is defined as the sequence of all these counters in a fixed order. To define this order, we have to define an order $(e_1, \ldots, e_6)$ on the edges of τ.

(i) If τ is not the root of the overall tree, then τ has a parent component with a current code C. We enumerate the edges of τ according to the edge position order $\mathsf{edge\text{-}pos}_C(\tau)$.

(ii) If τ is the root of the overall tree $\mathcal{T}^{\mathtt{F}}(G_i)$, then we have a current order of the vertices of τ, because the algorithm that compares two such trees cycles through all permutations of the vertices of τ as possible orders of these vertices. Any order of the vertices of τ induces a lexicographical order on the edges of τ. For example for $\tau = \{a, b, c\}$, the order $a < b < c$ induces the lexicographic enumeration of the edges $(e_1, \ldots, e_6) = ((a,b),(a,c),(b,a),(b,c),(c,a),(c,b))$.

Now the reference orientation of τ is defined as

$$\mathsf{ref\text{-}orient}(\tau) = (c_{e_i,j,r})_{i=1,\ldots,6,\ j=1,\ldots,p,\ r=1,\ldots,6}\,.$$

Let $\mathsf{ref\text{-}orient}(\tau')$ be the corresponding reference orientation of τ'. Clearly, for isomorphic graphs, τ and τ' must have the same reference orientation.

We argue that the counters of S_τ are equal to the counters of $T_{\tau'}$ if and only if the underlying graphs are isomorphic, i.e. $\mathsf{graph}(S_\tau) \cong \mathsf{graph}(T_{\tau'})$. That is, we are interested in an isomorphism which maps τ onto τ'. We already showed that this is the case if we have one child of τ and τ'. First, the counters $c_{e,j,r}$ distinguish between subtrees of different isomorphism classes. Second, since the isomorphism classes already are of the same cardinality, it is possible that F_i is isomorphic to F'_i but up to a certain mapping of τ onto τ'. Hence, an isomorphism that maps F_i onto F'_i guarantees to map all directed edges of τ onto directed edges of τ' which have the same color. When combining both arguments, counting the edges of the same color gives information about how many isomorphic subtrees can be mapped onto each other such that they induce an isomorphism.

Large child. In the case that τ is a large child of its parent node F_0, we go into recursion at τ *a priori*, i.e. before computing a code C for F_0. Note that the definition of the reference orientation of τ relies on C. Hence we have to deviate from the algorithm described above in this case.

Another point is, that we want to store the result of the recursion of a large child in $O(1)$ bits. However, the reference orientation cannot be stored within $O(1)$ bits. We solve these issues as follows.

Let $\tau = \{a, b, c\}$. We cycle though all the orders of the vertices of τ, and consider $\mathsf{ref\text{-}orient}(\tau)$ for all induced lexicographical orders on the edges of τ. Let M be the set of those orders of τ, where $\mathsf{ref\text{-}orient}(\tau)$ becomes minimal. The set M has size $O(1)$ and is stored on the work tape when we return from the recursion of the large child.

Then we start computing the codes of the parent node F_0, where we try all orders in M for τ. The minimum ones are used for the definition of the isomorphism order.

We summarize the correctness of the isomorphism order.

Theorem 4.13 *Let G and H be 3-connected non-planar graphs which contain 3-divisive separating triples. Then $G \cong H \iff S_\tau =_{\mathtt{F}} T_{\tau'}$ for some separating triples τ in G and τ' in H.*

Proof. The direction from left to right is clearly true. So let $S_\tau =_{\mathtt{F}} T_{\tau'}$. The argument is an induction on the depth of the trees which follows the inductive definition of the isomorphism order.

We already considered the case for trees of depth ≤ 1. For trees of depth $d > 1$ we consider first the subtrees rooted at four-connected component nodes at depth 1, say F_i and F'_j. The comparison states equality if the components have the same canon, i.e. are isomorphic. Let τ_0 and τ'_0 be children of F_i and F'_j. Let S_{τ_0} and $T_{\tau'_0}$ be the corresponding subtrees. By the induction hypothesis we know that $\mathsf{graph}(S_{\tau_0})$ is isomorphic to $\mathsf{graph}(T_{\tau'_0})$. For each such isomorphism consider the mapping ϕ of τ_0 to τ'_0. If we can find a code for F_i and one for F'_j such that virtual edges of τ_0 and τ'_0 appear at the same places in the canons (in such a way that ϕ describes the mapping which brings the order of edges from τ_0 into the order of edges from τ'_0), then ϕ can be extended to an isomorphism of F_i to F'_j. The $=_{\mathtt{F}}$-equality between child separating triples (together with their subtrees) inductively describes an isomorphism between the corresponding subgraphs. In the preceding discussion we argued how ϕ can be extended to an isomorphism of $\mathsf{graph}(S_\tau)$ onto $\mathsf{graph}(T_{\tau'})$. This is done with the help of the counters which form the reference orientation of τ and τ'. We argued that the counters are equal exactly if the corresponding subgraphs (rooted at the children of τ and τ') are isomorphic.

Finally, we consider the whole trees rooted at separating triple nodes. The comparison describes an order on the subtrees which corresponds to the split components of the separating triples. The order describes an isomorphism among the split components.

Hence, the isomorphism between the children at any level can be extended to an isomorphism between the corresponding subgraphs in G and H and therefore to G and H itself.

□

4.3 Limiting the number of choices for the roots of the component trees

The isomorphism order algorithm explores for a K_5-free graph G the biconnected, triconnected, and four-connected component trees. There are usually several candidates for the

root of these trees. In order to maintain the log-space bound, we cannot afford to cycle through all of them in case we have to go into recursion on more than one subtree. We describe how the algorithm chooses a small set of root candidates if necessary. We consider first the case of triconnected component trees and then the four-connected component trees.

As we will see below, the sub-routine to choose a small set of roots makes itself calls to the isomorphism test in some cases. This is similar to the case of a large child, where we deviate from the main algorithm and explore the large child a priori. Here we also have in some cases a single child to be explored and deviate to an isomorphism test for it. When we return, we store the result and continue with the root finding procedure.

Limiting the number of choices for the root of a triconnected component tree

We extend the algorithm of Datta et.al. [DLN$^+$09] where they showed for given planar graphs that the number of possible choices for the root of a triconnected component tree can be bounded by a sufficiently small number of separating pair nodes.

Let G be a K_5-free graph. An easy case is when G has no articulation points, i.e. G is 2-connected. Then the isomorphism order algorithm runs through all possibilities of separating pairs $\{a, b\}$ as root and explores $S_{\{a,b\}}$, the triconnected component tree rooted at $\{a, b\}$. If there is no separating pair in G, then G is even triconnected. If G is additionally planar then we are done, because isomorphism testing is in log-space [DLN08]. If G is not planar, it can be the V_8. In this case G is of constant size and isomorphism testing requires constant effort. Otherwise G contains 3-divisive separating triples. The isomorphism order algorithm runs through all possibilities of separating triples τ as roots and explores S_τ, the four-connected component tree rooted at τ.

The more interesting case is when G has articulation points. Then the isomorphism order algorithm runs through all articulation points as roots. Let S_a be a biconnected component tree of G rooted at articulation point a. Let B be a biconnected component which is a child of a in S_a. Let $\mathcal{T}^{\mathtt{T}}(B)$ be the triconnected component tree of B, see Figure 9.

We want to determine a small set of root candidates for $\mathcal{T}^{\mathtt{T}}(B)$. Let C_0 be the center of $\mathcal{T}^{\mathtt{T}}(B)$. If C_0 is a separating pair node then we take this pair as the unique root. If C_0 is a triconnected planar component node, then we can find roots in log-space due to Datta et.al. [DLN$^+$09]. If C_0 is the V_8, then we can run through all edges as roots for $\mathcal{T}^{\mathtt{T}}(B)$, because there are only constantly many.

The interesting case is when C_0 is non-planar and contains 3-divisive separating triples. Then we consider $\mathcal{T}^{\mathtt{F}}(C_0)$, the four-connected component tree of C_0 and invoke the computation of the parent separating triples for C_0 which is described below. This algorithm either directly returns a small set of separating pairs as root candidates for $\mathcal{T}^{\mathtt{T}}(B)$, or a small set of separating triples as root candidates for $\mathcal{T}^{\mathtt{F}}(C_0)$, say k triples. We use each of the three edges which are part of a 3-divisive separating triple as a root candidate for $\mathcal{T}^{\mathtt{T}}(B)$. Note that an edge e of such a separating triple may not be a separating pair in $\mathcal{T}^{\mathtt{T}}(B)$, and hence may not be a node in $\mathcal{T}^{\mathtt{T}}(B)$. Therefore, we extend $\mathcal{T}^{\mathtt{T}}(B)$ in this case by a node for e and connect it with the component where its separating triple is contained. Now we take the new node as one of the root candidates. Hence, we get $\leq 3k$ root separating pairs. We will see that this is good enough.

This means that we invoke the computation of the parent separating triples for C_0 *twice*: the first time to get roots for $\mathcal{T}^{\mathtt{T}}(B)$, and the second time when we actually compute roots for $\mathcal{T}^{\mathtt{F}}(C_0)$. The result can be different, because we start the second pass with a *rooted*

tree $\mathcal{T}^{\mathtt{T}}(B)$ and hence C_0 will have a parent separating pair.

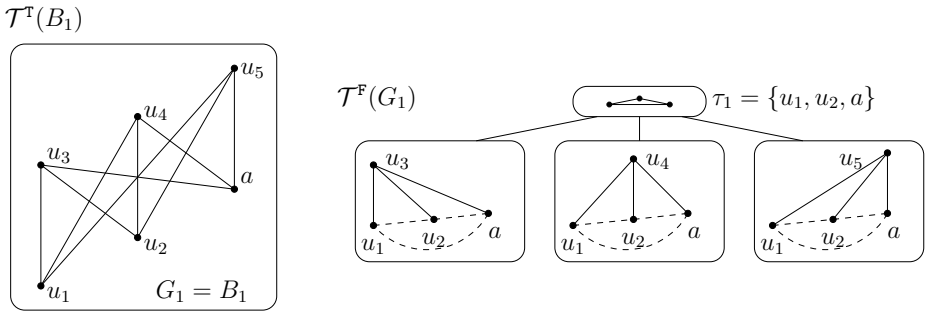

Figure 9: *The biconnected component B_1 contains no separating pairs and is therefore trivially triconnected such that the corresponding triconnected component tree $\mathcal{T}^{\mathtt{T}}(B_1)$ contains only one node, i.e. $G_1 = B_1$. G_1 is further decomposed into four-connected components as illustrated by the four-connected component tree $\mathcal{T}^{\mathtt{F}}(G_1)$.*

Limiting the number of choices for the root of a four-connected component tree

Let G be a K_5-free graph, B a biconnected component of G with parent articulation points a. Let C a triconnected component in B and C_0 the center node in $\mathcal{T}^{\mathtt{T}}(B)$. We describe how to determine a small number of separating triples as root candidates for C. We may assume that C is non-planar, otherwise isomorphism testing is in log-space.

We cannot take all the separating triples as possible roots, because then we would need $O(\log n)$ bits on the work-tape to remember the current out of $O(n)$ possible roots. If we do this recursively then we end up in a polynomial amount of space on the work-tape.

If C is the $K_{3,3}$, then we have exactly two choices of selecting a root separating triple in the decomposition of C. Figure 9 shows an example. Because these two separating triples are conflicting, we obtain two trees which contain one separating triple node each.

It remains the case that C is non-planar and contains 3-divisive separating triples. We distinguish the cases whether there is a parent separating pair for C in B or not. If C has a parent separating pair we show that we can determine ≤ 4 separating triples as root candidates for $\mathcal{T}^{\mathtt{F}}(C)$. The case that C has no parent separating pair can occur in two situations:

- There is no separating pair in B. Then C is the single triconnected component in B.
- There are separating pairs in B and $C = C_0$, the center of $\mathcal{T}^{\mathtt{T}}(B)$.

C has a parent separating pair. Let $\{a, b\}$ be the parent separating pair of C. We consider the four-connected component tree $\mathcal{T}^{\mathtt{F}}(C)$. Let D_0 be the center of $\mathcal{T}^{\mathtt{F}}(C)$. If D_0 is a separating triple node then we choose it as root and are done. Otherwise, D_0 corresponds to a four-connected planar component.

To each separating pair we can associate a unique component in $\mathcal{T}^{\mathtt{F}}(C)$.

Definition 4.14 *Let D be the node closest to the center D_0 in $\mathcal{T}^{\mathtt{F}}(C)$, whose associated four-connected component or separating triple contains the edge $\{a, b\}$. Then we define $\{a, b\}$ to be* associated with *D.*

To define the root candidates, we distinguish two cases:

1. $\{a, b\} \notin D_0$. Let $\{a, b\}$ be associated with D. We choose the separating triple as root that is nearest to D_0 on the unique simple path between D_0 and D.

2. $\{a, b\} \in D_0$. We canonize D_0 with $\{a, b\}$ as the starting edge. This gives four codes. In the smallest code among these, choose the separating triple in D_0 which gets the lexicographically smallest label. Thus, we have at most four choices for the root.

C has no parent separating pair. We start with some definitions. We associate with each articulation point a in B a unique component in $\mathcal{T}^{\mathtt{T}}(B)$ (unique with respect to C). The nodes of $\mathcal{T}^{\mathtt{T}}(B)$ which contain an articulation point a form a subtree of $\mathcal{T}^{\mathtt{T}}(B)$. Therefore there is a unique node in this subtree which is closest to C. We associate a with this node.

Definition 4.15 *Let C' be the node in $\mathcal{T}^{\mathtt{T}}(B)$ which contains a and is closest to C in $\mathcal{T}^{\mathtt{T}}(B)$. We define a to be* associated with C'.

We define colors for child articulation points which occur in the biconnected component B and for the adjacent separating pairs of C in $\mathcal{T}^{\mathtt{T}}(B)$. Let a be the parent articulation point of B.

- Let $a_1, \ldots, a_l$ be the child articulation points of C in $\mathcal{T}^{\mathtt{T}}(B)$. Let a_j be the root node of the biconnected subtree S_{a_j} of S_a. We partition the subtrees $S_{a_1}, \ldots, S_{a_l}$ into classes $E_1, \ldots, E_p$ of equal size subtrees. 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.

- Let $s_1, \ldots, s_m$ be the separating pairs which are connected to C in $\mathcal{T}^{\mathtt{T}}(B)$. Let S_{s_j} be the subtree of $\mathcal{T}^{\mathtt{T}}(B)$ rooted at s_j. We partition the subtrees $S_{s_1}, \ldots, S_{s_m}$ into classes $\widehat{E}_1, \ldots, \widehat{E}_{\hat{p}}$ of equal size subtrees. Let $\widehat{k}_j$ be the number of subtrees in $\widehat{E}_j$. Let the order of the size classes be such that $\widehat{k}_1 \leq \widehat{k}_2 \leq \cdots \leq \widehat{k}_{\hat{p}}$. All virtual edges of separating pairs with their subtrees in size class $\widehat{E}_j$ are colored with color j.

Let n_a be the size of S_a and n_B be the size of S_B. The subtrees in E_j have size $\leq n_a/k_i$, for $j \geq i$, and similar, the subtrees in $\widehat{E}_j$ have size $\leq n_B/\widehat{k}_i$, for $j \geq i$. Let

$$k_0 = \begin{cases} k_1, & \text{if } k_1 > 1, \\ \min\{k_2, \widehat{k}_1\}, & \text{if } k_1 = 1. \end{cases}$$

We will show that in the cases where we have to bound the number of root candidates, we either have a constant number of candidates, if $k_0 = 1$, or $\leq 8k_0$ candidates, if $k_0 \geq 2$. In the latter case we go into recursion on trees of size $\leq n_a/k_0$, respectively n_B/k_0. This is good enough to maintain the space bound.

To limit the number of potential root nodes for $\mathcal{T}^{\mathtt{F}}(C)$, we distinguish several cases below. The center D_0 of $\mathcal{T}^{\mathtt{F}}(C)$ will play an important role thereby. In some of the cases we will show that the number of automorphisms of D_0 is small. This already suffices for our purpose: in this case, we cycle through every edge of D_0 as starting edge, and canonize the component D_0 separately. Let A be the set of separating triples that lead to the minimum code. Although

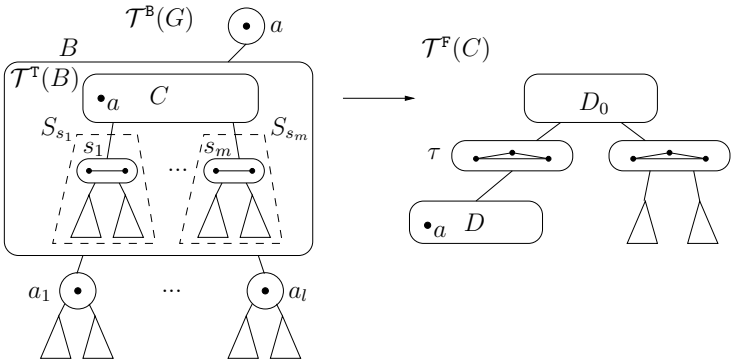

Figure 10: *(a) The decomposition of a graph G into biconnected and triconnected components. In $\mathcal{T}^{\mathtt{B}}(G)$ we have a biconnected component B with its parent articulation point a. B is decomposed into its triconnected component tree $\mathcal{T}^{\mathtt{T}}(B)$. C is a triconnected component in $\mathcal{T}^{\mathtt{T}}(B)$, where $a_1, \ldots, a_l$ are child articulation points and $s_1, \ldots, s_m$ are adjacent separating pairs of C in $\mathcal{T}^{\mathtt{T}}(B)$.*
(b) The decomposition of C into its four-connected component tree $\mathcal{T}^{\mathtt{F}}(C)$. The parent articulation point is contained in C and hence in $\mathcal{T}^{\mathtt{F}}(C)$. The situation is shown where a occurs not in the center D_0 of the tree.

there can be polynomially many possible candidates for the canon, the minimum ones are bounded by the number of automorphisms of D_0, which is small. We take the separating triples as root candidates for $\mathcal{T}^{\mathtt{F}}(C)$ which come first in the position order encountered in each of these minimum codes. Hence the number of roots is bounded by the number of automorphisms of D_0.

Consider the case where the parent articulation point a is associated with some component $D \neq D_0$. We find the path from D to D_0 in $\mathcal{T}^{\mathtt{F}}(C)$ and find the separating triple closest to D_0 on this path. This serves as the unique choice for the root of $\mathcal{T}^{\mathtt{F}}(C)$, see Figure 10.

For the following cases, we assume that a is associated with D_0. We proceed with the case analysis according to the number $l \geq 0$ of child articulation points and the number $m \geq 0$ of separating pairs adjacent to C in $\mathcal{T}^{\mathtt{F}}(C)$.

Case I: $l = 0$. There are no articulation points associated with C and hence, C is a leaf node in S_a.

1) $m = 0$. There are no separating pairs in C and hence, C is also a leaf node in $\mathcal{T}^{\mathtt{T}}(B)$. There is no recursion on biconnected or triconnected subtrees of C. In this case we can cycle through all separating triples as root for $\mathcal{T}^{\mathtt{F}}(C)$. We also color a with a distinct color.

2) $m \geq 1$. We return the $\widehat{k}_1$ separating pairs which are the roots of the subtrees in $\widehat{E}_1$ as root candidates for $\mathcal{T}^{\mathtt{T}}(B)$.

Case II: $l = 1$. We do the isomorphism test for the single child articulation point associated with C a priori and store the result. We color the parent and the unique child with distinct colors and proceed with C as in Case I.

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

1) $1 \leq \widehat{k}_1 < k_1$. We return the $\widehat{k}_1$ separating pairs which are the roots of the subtrees in $\widehat{E}_1$ as root candidates for $\mathcal{T}^{\mathtt{T}}(B)$.

2) $1 \leq k_1 \leq \widehat{k}_1$ or $m = 0$, i.e. $\widehat{k}_1 = 0$. We consider the following sub-cases.

   a) **Some articulation point a_j from E_1 is not associated with C.** Note that we must have $m > 0$ in this case, i.e. there are separating pairs in B. Let a_j be associated with component $C' \neq C$ in $\mathcal{T}^{\mathtt{T}}(B)$. Find the path from C' to C in $\mathcal{T}^{\mathtt{T}}(B)$ 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 for every articulation point from E_1. Therefore, we get up to k_1 separating pairs as candidates for the root.

   b) **All articulation points from E_1 are associated with C.** In this case we consider the four-connected component tree of C, $\mathcal{T}^{\mathtt{F}}(C)$, and its center D_0.

      (i) **Some articulation point a_j from E_1 is not associated with D_0.** Let a_j be associated with a four-connected component $D \neq D_0$. Find the path from D to D_0 in $\mathcal{T}^{\mathtt{F}}(C)$ and select the separating triple node closest to D_0 on this path. Thus a_j uniquely defines a separating triple. In the worst case, this may happen for every articulation point from E_1. Therefore, we get up to k_1 separating triples as candidates for the root.

      (ii) **All articulation points from E_1 are associated with D_0.**

      - $k_1 \geq 2$. Every automorphism of D_0 fixes the parent articulation point a and setwise fixes the k_1 articulation points from E_1. By Lemma 4.16 below, there are at most $4k_1$ automorphisms of D_0.
      - $k_1 = k_2 = 1$. Every automorphism of D_0 fixes the parent articulation point a and the two articulation points from E_1 and E_2. By Corollary 4.17 below, D_0 has at most one non-trivial automorphism in this case.
      - $k_1 = 1 < k_2$ **and** $(k_2 \leq \widehat{k}_1$ **or** $\widehat{k}_1 = 0)$**.** We do the isomorphism test for the single child articulation point from E_1 a priori and store the result. By an analogous argument as in the case where $k_1 \geq 2$, D_0 has at most $4k_2$ automorphisms.
      - $k_1 = 1 \leq \widehat{k}_1 < k_2$**.** Again we do the isomorphism test for the single child articulation point from E_1 a priori and store the result. We distinguish two sub-cases.
        - **Some separating pair s_j from $\widehat{E}_1$ is not associated with D_0.** Choose the separating triple nearest to D_0 which is on the unique path between D_0 and the 4-connected component associated with s_j. Hence, we get up to $\widehat{k}_1$ separating triples as candidates for the root.
        - **All separating pairs in $\widehat{E}_1$ are associated with D_0.** Every automorphism of D_0 fixes the parent articulation point a and setwise fixes the $2\widehat{k}_1$ separating pairs from $\widehat{E}_1$. By Lemma 4.16 below, there are at most $4 \cdot 2\widehat{k}_1$ automorphisms for D_0.

As already mentioned before the case analysis, we have shown that for

$$k_0 = \begin{cases} k_1, & \text{if } k_1 > 1, \\ \min\{k_2, \widehat{k}_1\}, & \text{if } k_1 = 1, \end{cases}$$

we have $\leq 8k_0$ root candidates, if $k_0 \geq 2$, and a constant number otherwise.

The following lemma bounds the number of automorphisms of a 3-connected planar component.

Lemma 4.16 [DLN+09] *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.*

Corollary 4.17 [DLN+09] *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.*

Note that Lemma 4.16 holds for all 3-connected planar graphs, except for some special cases which are of constant size. For these special cases, we do not have to limit the number of possible minimum codes.

The preceding discussion implies that if two biconnected component trees have equal isomorphism order for some choice of the root, then the corresponding graphs are isomorphic. The reverse direction clearly holds as well.

Theorem 4.18 *Given two biconnected graphs B and B' and their triconnected component trees S and T, then $B \cong B'$ if and only if there is a choice of separating pairs $\{a, b\}, \{a', b'\}$ in B and B' such that $S_{\{a,b\}} =_{\mathrm{T}} T_{\{a',b'\}}$.*

Proof. We refer to Theorem 4.13 for the correctness of isomorphism order of four-connected component trees, and assume the correctness of the isomorphism order on four-connected component trees.

We prove the right to left implication first. Assume $S_{\{a,b\}} =_{\mathrm{T}} T_{\{a',b'\}}$. Then an inductive argument on the depth of the trees that follows the definition of the isomorphism order implies that B and B' are isomorphic. If the grandchildren of $\{a, b\}$ and $\{a', b'\}$, are equal up to step 4 of the isomorphism order, then the corresponding subgraphs are isomorphic by induction hypothesis. We compare the children of $\{a, b\}$ and $\{a', b'\}$. If they are equal then we can extend the $=_{\mathrm{T}}$-equality to the separating pairs s and t.

When subtrees are rooted at separating pair nodes, the comparison describes an order on the subtrees which correspond to split components of the separating pairs. The order describes an isomorphism among the split components.

When subtrees are rooted at triconnected component nodes, say C and H_j, the comparison states equality if the components are isomorphic. If C and H_j are 3-connected non-planar components, then their isomorphism is checked from the isomorphism order of their four-connected component trees. This algorithm not only gives an isomorphism between C and H_j, but also checks whether the children of C and H_j mapped to each other are indeed isomorphic, and also ensures that parents of C and H_j are mapped to each other.

Hence, the isomorphism between the children of $\{a, b\}$ and $\{a', b'\}$ can be extended to an isomorphism between B and B'.

The reverse direction holds obviously as well. Namely, if B and B' are isomorphic and there is an isomorphism that maps the separating pair $\{a, b\}$ of B to the separating pair $\{a', b'\}$ of B', then the triconnected component trees $S_{\{a,b\}}$ and $T_{\{a',b'\}}$ rooted respectively at $\{a, b\}$ and $\{a', b'\}$ are clearly equal. Hence, such an isomorphism maps separating pairs of B onto

separating pairs of B'. This isomorphism describes a permutation on the split components of separating pairs, which means we have a permutation on triconnected components, the children of the separating pairs. By induction hypothesis, the children (at depth $d+2$) of two such triconnected components are isomorphic and equal according to $=_{\mathtt{T}}$. More formally, one can argue inductively on the depth of $S_{\{a,b\}}$ and $T_{\{a',b'\}}$. □

4.4 Limiting the number of recursive calls for articulation points and separating pairs

When we explore the triconnected component tree $\mathcal{T}^{\mathtt{T}}(B)$, we might find several copies of articulation point a. That is, a may occur in several components in $\mathcal{T}^{\mathtt{T}}(B)$ if a is part of a separating pair. We want to go into recursion on a to the subtree S_a only once. Datta et.al. [DLN$^+$09] defined the *reference copy* of articulation point a as the unique copy of a in the node C in $\mathcal{T}^{\mathtt{T}}(B)$ which is closest to the root in $\mathcal{T}^{\mathtt{T}}(B)$. Their algorithm goes into recursion to S_a only once, when it reaches a in node C.

If C is a component node which is not further decomposed, then we can directly take the algorithm of [DLN$^+$09]. If C is decomposed into its four-connected component tree $\mathcal{T}^{\mathtt{F}}(C)$, then we define the *reference copy of* a as follows. This is the unique copy of a in the component or separating triple which is associated to the node D, which is closest to the root in $\mathcal{T}^{\mathtt{F}}(C)$.

We have an analog situation for child separating pairs of a triconnected component C in $\mathcal{T}^{\mathtt{F}}(C)$. For a child separating pair $\{a,b\}$ of C we define a *reference copy of* $\{a,b\}$ in $\mathcal{T}^{\mathtt{F}}(C)$ as as the unique copy of $\{a,b\}$ as follows. This is the unique copy of $\{a,b\}$ in the component or separating triple which is associated to the node D, which is closest to the root in $\mathcal{T}^{\mathtt{F}}(C)$.

If D is a component node, then we compute codes and explore all edges of D. The algorithm goes into recursion at S_a or $S_{\{a,b\}}$ only when it reaches a or $\{a,b\}$ for the first time in a code of D.

Lemma 4.19 *Let C be a triconnected component. The reference copy of an articulation point a or a separating pair $\{a,b\}$ in $\mathcal{T}^{\mathtt{F}}(C)$ can be found in log-space.*

Proof. The proof is similar to the proof of the corresponding lemma in [DLN$^+$09]. We distinguish three cases for a and $\{a,b\}$ in $\mathcal{T}^{\mathtt{F}}(C)$. We define a unique component D, where a or $\{a,b\}$ is contained. We distinguish the following cases. Let P be one of a or $\{a,b\}$.

- P occurs in the root separating triples of $\mathcal{T}^{\mathtt{F}}(C)$. That is, P occurs already at the beginning of the comparisons for $\mathcal{T}^{\mathtt{F}}(C)$. Then we define D as the root separating triple.

- P occurs in separating triples other than the root of $\mathcal{T}^{\mathtt{F}}(C)$. Then P occurs in all the component nodes, which contain such a separating triple. These nodes form a connected subtree of $\mathcal{T}^{\mathtt{F}}(C)$. Hence, one of these component nodes is the closest to the root of $\mathcal{T}^{\mathtt{F}}(C)$. This component is always a four-connected component node. Let D be this component. Note, that the comparison first compares P with P' before comparing the biconnected, triconnected or four-connected subtrees, so we reach these copies first in the comparison.

- P does not occur in a separating triple. Then, P occurs in only one four-connected component node in $\mathcal{T}^{\mathtt{F}}(C)$. Let D be this component node.

Clearly, the cases can be detected in logspace. □

As in [DLN+09], we define trees which are based on the actual recursive structure of the algorithm described above. We distinguish the case of triconnected and four-connected component trees. Recall that a *large child* of a tree of size N is a child of size $> N/2$. A large child is visited a priori.

Definition 4.20 *Let B be a biconnected component and $\mathcal{T}^{\mathtt{T}}(B)$ its triconnected component tree. Let C be a node in $\mathcal{T}^{\mathtt{T}}(B)$, i.e. a triconnected component node or a separating pair node. The* tree S^*_C rooted at C *is defined as*

- *the subtree of $\mathcal{T}^{\mathtt{T}}(B)$ rooted at C (with respect to the root of $\mathcal{T}^{\mathtt{T}}(B)$) and*
- *the subtrees S_a for all articulation points a that have a reference copy in the subtree of $\mathcal{T}^{\mathtt{T}}(B)$ rooted at C, except those S_a that are a large child of S_B.*

Let $\mathcal{T}^{\mathtt{F}}(C)$ be the four-connected component tree of C. Let D be a node in $\mathcal{T}^{\mathtt{F}}(C)$, i.e. a four-connected component node or a separating triple node. The tree S^*_D rooted at D *is defined as*

- *the subtree of $\mathcal{T}^{\mathtt{F}}(C)$ rooted at D (with respect to the root of $\mathcal{T}^{\mathtt{F}}(C)$) and*
- *the subtrees S_a for all articulation points a that have a reference copy in the subtree of $\mathcal{T}^{\mathtt{F}}(C)$ rooted at D, except those S_a that are a large child of B in S_B or a large child of C in S_C, and*
- *the subtrees $S_{\{u,v\}}$ for all separating pairs $\{u, v\}$ that have a reference copy in the subtree of $\mathcal{T}^{\mathtt{F}}(C)$, except those $S_{\{u,v\}}$ that are a large child of C in S_C.*

Figure 11 illustrates the definition.

4.5 Space requirement of the isomorphism order algorithm

We give an overview of what is stored on the work-tape when we go into recursion at triconnected and four-connected subtrees when rooted at the different kind of nodes.

In the case of the biconnected component tree and the triconnected component tree, when the root is an articulation point, a biconnected component or a separating pair, then we can directly take the algorithm presented in [DLN+09]. This is also the case if the roots are planar triconnected component nodes. Here we consider the other components.

Separating triples or four-connected components

We go into recursion at separating triples and four-connected components in $\mathcal{T}^{\mathtt{F}}(G_i)$ and $\mathcal{T}^{\mathtt{F}}(H_j)$. When we reach a reference copy of an articulation point or a separating pair in both trees, then we interrupt the comparison of G_i with H_j and go into recursion as described before, i.e. we compare the corresponding nodes, the children of B and B' or the children of G_i and H_j, respectively. When we return from recursion, we proceed with the comparison of $\mathcal{T}^{\mathtt{F}}(G_i)$ and $\mathcal{T}^{\mathtt{F}}(H_j)$.

In this part we concentrate on the comparison of $\mathcal{T}^{\mathtt{F}}(G_i)$ and $\mathcal{T}^{\mathtt{F}}(H_j)$. We give an overview of what is stored on the work-tape when we go into recursion at separating triples and four-connected components. Basically, the comparison is similar to that for biconnected and triconnected component trees in [DLN+09].

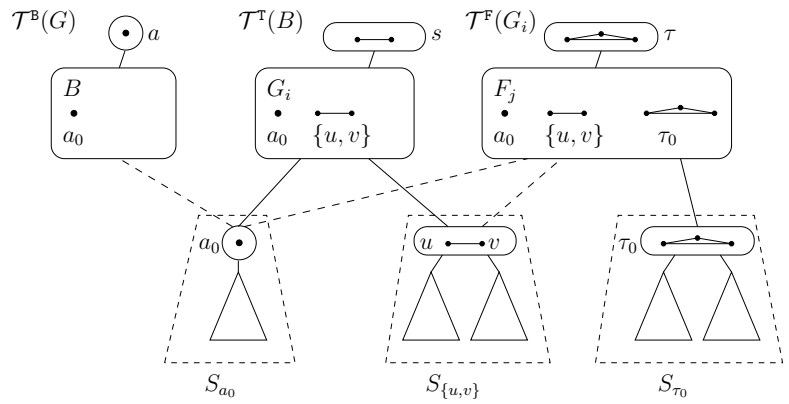

Figure 11: *The figure shows a series of decompositions upto four-connected components. The situation is shown where a_0 is a child articulation point of B and which has reference copies in C and D. From left to right, if it is a large child of B, C or D then it is treated at the corresponding position as large child a priori. Here, a_0 is no large child of B but a large child of C. The situation is similar for the separating pair $\{u, v\}$, which is a child of C and has a reference copy in D. Here, $\{u, v\}$ is a large child of C. The dashed lines summarize all possible connections for subtrees S_{a_0} and $S_{\{u,v\}}$. The algorithm selects only one connection for each subtree.*

- For a root separating triple node, we store at most $O(\log k)$ bits on the work-tape, when we have k candidates as root separating triples for $\mathcal{T}^{\mathtt{F}}(G_i)$. Hence, whenever we make recomputations in $\mathcal{T}^{\mathtt{F}}(G_i)$, we have to find the following in advance: first, recompute the root separating pair node, then we can determine the parent separating pair node. Then we can recompute the root separating triple node. For this, we compute $\mathcal{T}^{\mathtt{F}}(G_i)$ in log-space and with the rules described above, we find the candidate edges in log-space. With the bits on the work-tape, we know which of these candidate edges is the current root separating triple. We proceed as in the case of non-root separating triple nodes described next.

- For a non-root separating triple node and four-connected component nodes, we store the following. For separating triple nodes, we have some information of the orientation graphs on the work-tape. More precisely, for the i-th isomorphism class we compute the number of edges with color i. For these counters we need $O(\log k)$ bits if k is the number of subtrees in this isomorphism class.

  For four-connected component nodes we cannot afford to store the entire work-tape content when we go into recursion at a child separating triple node τ. It suffices to store the information of

  - the codes which are not eliminated,
  - the codes which encounter a virtual edge of τ for the first time
  - the direction in which the virtual edge of τ is encountered.

  This takes altogether $O(1)$ space.

When we return from recursion we do the following. First, recompute the root separating pair node, then we can determine the parent separating pair node. Then we can recompute the root separating triple node of the triconnected component where we are. With this, we can determine the triconnected node which is the parent. With the information on the work-tape, we can proceed with the computations. That is, for separating triple nodes we proceed with the next comparison of children and for four-connected component nodes, we look at the work-tape which the active codes are and proceed with the next edges in the bit-by-bit comparisons of the codes.

V_8-components

The tasks are similar to that for triconnected planar components. We compare the codes of the V_8-nodes and when we reach child articulation points associated to these V_8-components, then we go into recursion.

When we return from recursion we recompute the root for the triconnected component tree. Then, we can determine the parent of the V_8 and obtain this way the starting edge of the codes which we compare bit-by bit. There are $O(1)$ many codes, so we can store in constant size which the current codes are. We proceed at the first occurrence of an edge with the child articulation point node where we went into recursion. Hence, we need $O(1)$ bits on the work-tape.

Analysis of the space requirement for the component trees

We analyze the comparison algorithm when it compares subtrees rooted at separating triples, 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.

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

where $k_j \geq 2$ (for all j) is the number of subtrees of the same size. Hence, $\mathcal{S}(N) = O(\log N)$.

For the explanation of the recursion equation it is helpful to imagine that we have three work-tapes. We use the first work-tape when we go into recursion at articulation point nodes, we use the second work-tape when we go into recursion at separating pair nodes and we use the third work-tape when we go into recursion at separating triple nodes. The total space needed is the sum of the space of the three work-tapes.

- At an articulation point node, the value k_j is the number of elements in the j-th size class among the children $B_1, \ldots, B_k$ of the articulation point node. We store $O(\log k_j)$ bits and recursively consider subtrees of size $\leq N/k_j$.
- At a separating pair node the value k_j is the number of elements in the j-th isomorphism class among the children $G_1, \ldots, G_k$ of the separating pair node. We store $O(\log k_j)$ bits and recursively consider subtrees of size $\leq N/k_j$.
- At a separating triple node the value k_j is the number of elements in the j-th isomorphism class among the children $F_1, \ldots, F_k$ of the separating triple node. We store $O(\log k_j)$ bits and recursively consider subtrees of size $\leq N/k_j$.

This finishes the complexity analysis. We get the following theorem.

Theorem 4.21 *The isomorphism order between two K_5-free graphs can be computed in log-space.*

4.6 The canon of K_5-free graphs.

Using this algorithm for isomorphism order, we show that the canon of a planar graph can be output in log-space. The canonization of K_5-free graphs proceeds exactly as in the case of planar graphs. We extend the canonization procedure for the non-planar components.

The canon for a V_8-component

We already described, how to compute a canon for a single V_8-component. It consists of a list of all directed edges in the V_8. For a V_8-component node C and a parent separating pair $\{a, b\}$ we define the canon $l(C, a, b)$ exactly as if C is a 3-connected component. That is, (a, b) followed by the canon of C and then the canons for the child separating pairs of C in that order in which the child separating pairs appear in the canon of C.

The canon for a four-connected component tree

Let G_i be a 3-connected K_5-free component with 3-divisive separating triples. We describe now the canon of $\mathcal{T}^{\mathbf{F}}(G_i)$.

A log-space procedure traverses the four-connected component tree and makes oracle queries to the isomorphism order algorithm and outputs a canonical list of edges, along with delimiters to separate the lists for siblings.

For an example, consider the canonical list $l(S, \tau)$ of edges for the tree S_τ of Figure 8. Let $\tau = \{a, b, c\}$. Let $l(F_i, a, b, c)$ be the canonical list of edges of the four-connected component F_i and a given order on the vertices of τ (i.e. the canonical list of F_i with τ the parent separating triple). Since F_i is planar and at least 3-connected, we invoke the algorithm of [DLN08] for canonization. Let $\tau_1, \ldots, \tau_{l_1}$ be the order of the separating triples as they occur in the canon of F_i. We also write for short $l'(S_{\tau_i}, \tau_i)$ which is one of $l(S_{\tau_i}, \varphi(\tau_i))$ where $\varphi(\tau_i)$ is a permutation of separating triple τ_i. Then we get the following canonical list for S_τ.

$$\begin{aligned}
l(S, a, b, c) &= [\ (a, b, c)\ l(S_{F_1}, a, b, c)\ \ldots\ l(S_{F_k}, a, b, c)\ ], \text{ where} \\
l(S_{F_1}, a, b, c) &= [\ l(F_1, a, b, c)\ l'(S_{\tau_1}, \tau_1)\ \ldots\ l'(S_{\tau_{l_1}}, \tau_{l_1})\ ] \\
&\vdots \\
l(S_{F_k}, a, b, c) &= [\ l(F_k, a, b, c)\ l'(S_{\tau_{l_k}}, \tau_{l_k})\ ]
\end{aligned}$$

In the case the triconnected non-planar component is a $K_{3,3}$, say G_i, then for the code of $l(G_i, a, b)$ (if a, b is a parent separating pair) we have to fix one separating triple. To select the canonical smaller separating triple, we query the isomorphism order algorithm. We have a similar situation if the $K_{3,3}$ forms a biconnected component, say B_i, for the canon of $l(B_i, a)$ (if a is a parent articulation point). The canonical smaller separating triple is those which contains a.

A log-space transducer renames then the vertices according to their first occurrence in this list, to get the canon for the four-connected component tree. This canon depends upon

the choice of the root of the four-connected component tree. Further log-space transducers cycle through all the articulation points as roots to find the minimum canon among them, then rename the vertices according to their first occurrence in the canon and finally, remove the virtual edges and delimiters to obtain a canon for the planar graph.

For the planar 3-connected components and for the canonization of biconnected components we use the algorithm from Datta et.al. [DLN$^+$09]. We get

Theorem 4.22 *A K_5-free graph can be canonized in log-space.*

5 Acknowledgement

We thank V. Arvind, Bireswar Das, Raghav Kulkarni, Nutan Limaye, Meena Mahajan and Jacobo Torán for helpful discussions.

References

[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.

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

[Asa85] Tetsuo Asano. An approach to the subgraph homeomorphism problem. *Theoretical Computer Science*, 38, 1985.

[BHZ87] R. B. Boppana, J. Hastad, and S. Zachos. Does co-NP have short interactive proofs? *Inf. Process. Lett.*, 25(2):127–132, 1987.

[BL83] László Babai and Eugene M. Luks. Canonical labeling of graphs. In *STOC '83: Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pages 171–183, 1983.

[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.

[DLN$^+$09] Samir Datta, Nutan Limaye, Prajakta Nimbhorkar, Thomas Thierauf, and Fabian Wagner. Planar graph isomorphism is in log-space. Technical Report TR09-052, Electronic Colloquium on Computational Complexity (ECCC), 2009.

[Khu88] Samir Khuller. Parallel algorithms for K_5-minor free graphs. Technical Report TR88-909, Cornell University, Computer Science Department, 1988.

[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.

[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.

[MR87] Gary L. Miller and V. Ramachandran. A new graphy triconnectivity algorithm and its parallelization. In ACM, editor, *Proceedings of the nineteenth annual ACM Symposium on Theory of Computing, New York City, May 25–27, 1987*, pages 335–344, 1987.

[Pon91] Ilia N. Ponomarenko. The isomorphism problem for classes of graphs closed under contraction. *Journal of Mathematical Sciences (JSM, formerly Journal of Soviet Mathematics)*, 55, 1991.

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

[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.

[Tut66] William T. Tutte. *Connectivity in graphs*. University of Toronto Press, 1966.

[TW09] Thomas Thierauf and Fabian Wagner. Reachability in $K_{3,3}$-free graphs and K_5-free graphs is in unambiguous log-space. In *17th International Symposium, Fundamentals of Computation Theory (FCT)*, pages 323–334, 2009.

[Vaz89] Vijay V Vazirani. NC algorithms for computing the number of perfect matchings in $k_{3,3}$-free graphs and related problems. *Information and Computation*, 80, 1989.

[Wag37] Klaus Wagner. Über eine Eigenschaft der ebenen Komplexe. In *Mathematical Annalen*, volume 114, 1937.

[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.

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