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`

December 22, 2009

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 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 [AM00] showed that an embedding of a planar graph can be computed in log-space. These facts are used in [TW08] 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 $K_{3,3}$ and K_5 minor-free graphs and proves the uniqueness of such decompositions. In Section 4 we give a log-space algorithm for isomorphism and canonization of $K_{3,3}$ and 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.

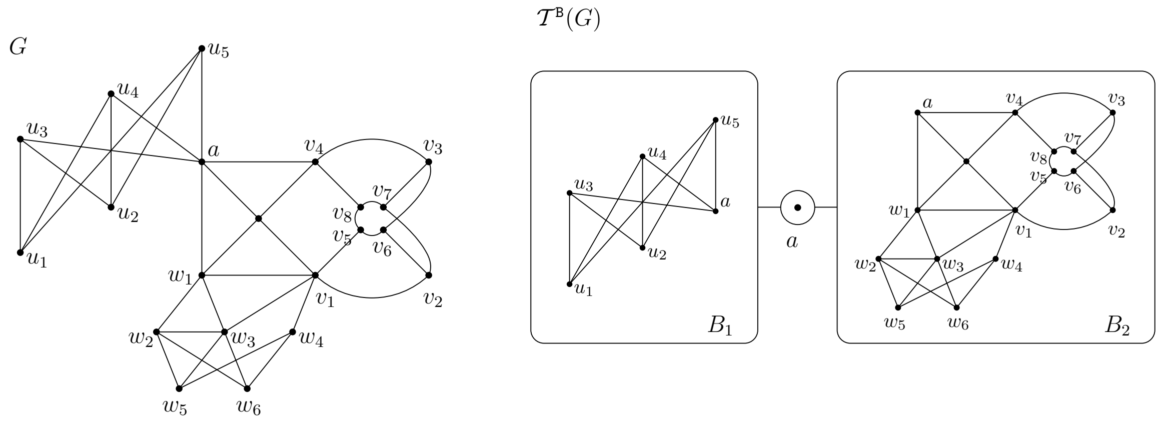

Figure 1: *Decomposition of a K_5-free graph G into biconnected components B_1 and B_2 and the corresponding biconnected component tree $\mathcal{T}^{\mathrm{B}}(G)$.*

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 $\mathcal{T}^{\mathrm{T}}(G)$ *(see Figure 2).*

For a component tree T, the *size of an individual component node C* of T is the number of nodes in C. The vertices of the 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. The size of T is at least as large as the number of vertices in $\mathsf{graph}(T)$, the graph corresponding to the component tree T. Let T_C be T when rooted at node C. A child of C is called a *large child* if $|T_C| > |T|/2$. $\#C$ denotes the number of children of 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 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.

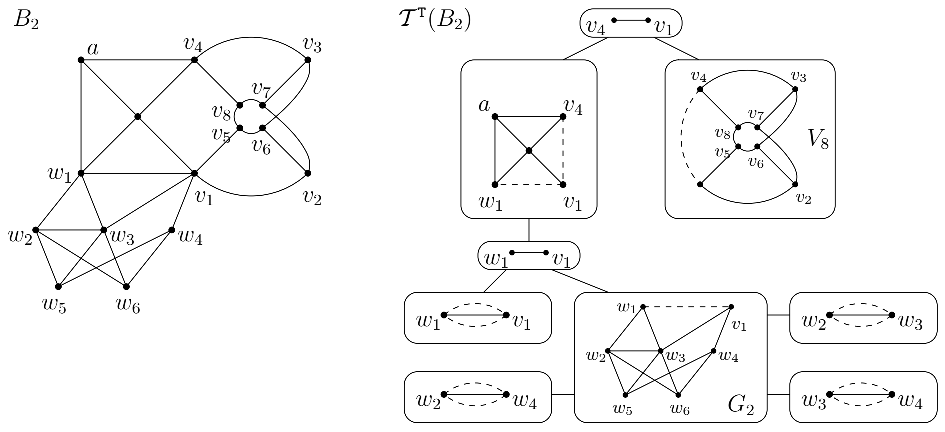

Figure 2: *Decomposition of the biconnected component B_2 and the corresponding triconnected component tree $\mathcal{T}^{\mathrm{T}}(B_2)$. There are leaf separating pairs connected to the triconnected component G_2. Virtual edges are indicated with dashed lines.*

3 Decomposition of $K_{3,3}$-free and K_5-free graphs

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

We consider the decomposition of biconnected $K_{3,3}$-free graphs into triconnected components. Tutte [Tut66] proved that the decomposition 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. $\square$

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. $\square$

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

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 Decomposition of K_5-free graphs

We decompose the given K_5-free graph into 3-connected and 4-connected components. It follows from a theorem of Wagner [Wag37] that besides planar components we obtain the following non-planar components that way:

- 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 $K_{3,3}$.
- The remaining 3-connected non-planar components are further decomposed into 4-connected components which are all planar.

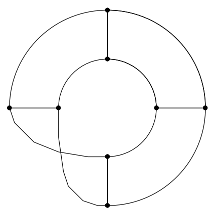

Figure 3: *The four-rung Mobius ladder, also called* V_8.

Khuller [Khu88] described a decomposition of K_5-free graphs with a clique-sum operation. If two graphs G_1 and G_2 each contain 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.

If G can be constructed by repeatedly taking k-clique-sums starting from graphs isomorphic to members of some graph class $\mathcal{G}$, then we say $G \in \langle \mathcal{G} \rangle_k$. The class of K_5-free graphs can be decomposed as follows.

Theorem 3.7 [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 which is built as the 3-clique-sum of planar graphs must all be planar.
- The V_8 is non-planar and 3-connected and cannot be part of a 3-clique sum, because it does not contain a triangle as subgraph.

By Theorem 3.7 and these observations we have the following situation.

Corollary 3.8 (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.*

Similar to the decomposition algorithm of Vazirani [Vaz89], we decompose the K_5-free graph into triconnected components. That is, we first decompose it into biconnected components and then the biconnected components further into triconnected 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. But the decomposition there is not unique up to isomorphism. We describe here a different way of decomposition. 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 3.9 *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 of the split components of σ is a K_4 which consists of the vertices of σ and one vertex of τ each. 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 some split component of σ.

Definition 3.10 *Let G be an undirected K_5-free 3-connected graph. Two 3-divisive separating triples $\tau \neq \sigma$ are* conflicting *if one of them, say τ, is no 3-divisive separating triple in in some split component of σ.*

Lemma 3.11 *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 3.12 *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.

The next claim shows that we can weaken the assumption in Claim 3.12.

Claim 3.13 *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. In particular, τ must be in one split component of σ. Now the claim follows from Claim 3.12. $\square$

From the proof of Claim 3.13 we deduce:

Claim 3.14 *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.*

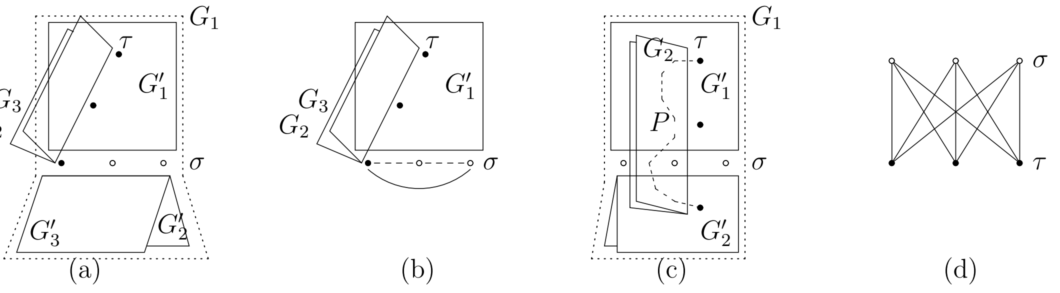

Figure 4: *(a) Two 3-divisive separating triples σ (indicated with white color) and τ (indicated with black color) which have one vertex in common, and their split components G'_1, G'_2, G'_3 and G_1, G_2, G_3, respectively.*
(b) The split component of σ where G'_2 and G'_3 are replaced by virtual edges (indicated with dashed lines).
(c) The situation is shown where σ and τ seem to be conflicting. But this situation cannot occur since there is a path P which proves that σ is no separating triple.
(d) In a $K_{3,3}$, the triples σ and τ are conflicting.

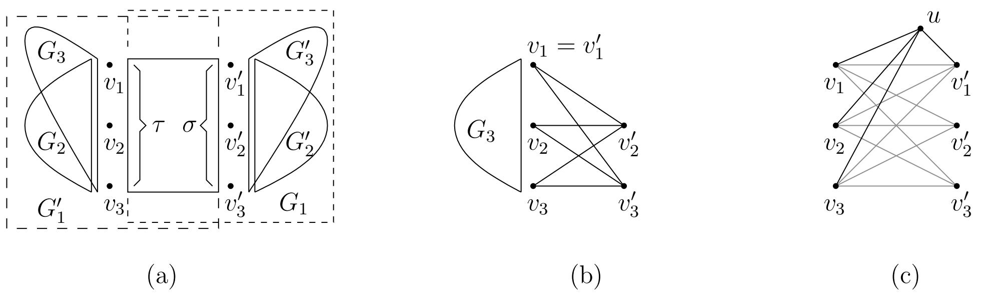

Figure 5: *(a) The 3-divisive separating triples τ, σ are shown schematically, as well as the split components G_i of τ and the G'_i of σ, indicated by dashed and solid shapes.*
(b) Separating triples τ, σ share the vertex $v_1 = v'_1$ and are pairwise connected.
(c) The split component G_1 of τ collapsed to one vertex u, except vertex v'_1.

The assumption of Claim 3.14 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
- $\tau \cap \sigma \neq \emptyset$.

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, for an example see Figure 5(b).

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. W.l.o.g. we can assume, that all the remaining vertices are collapsed to u. Also see Figure 5(c). Because G_1 is 3-connected, there are paths from u to the vertices of τ that do not go through v_1'. Therefore u and the vertices of τ will be in one split component of σ in G. By Claim 3.13, τ and σ are not conflicting. This finishes the proof of Lemma 3.11 $\square$

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, 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* $\mathcal{T}^{\mathrm{F}}(G)$. For an example see Figure 6. A special property of the node τ is that it 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.

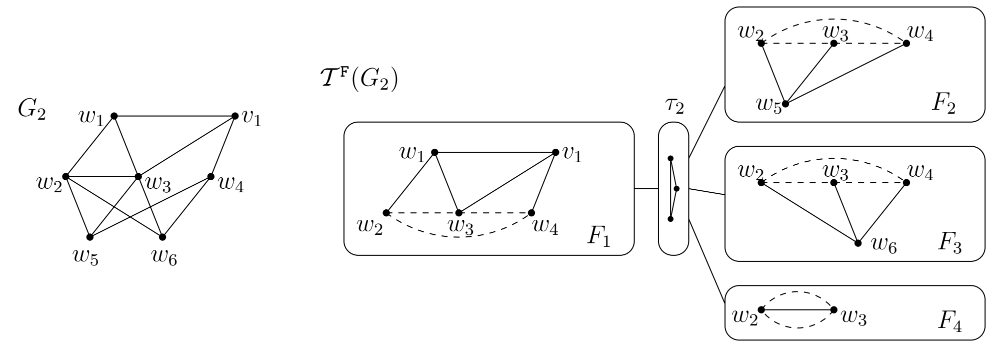

Figure 6: *The decomposition of G_2 into four-connected components $F_1, \ldots, F_4$ is shown, together with the four-connected component tree $\mathcal{T}^{\mathrm{F}}(G_2)$. The edges with both ends in the 3-divisive separating triple $\tau_2 = \{w_2, w_3, w_4\}$ are virtual edges indicated by dashed 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 3.15 *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.*

The triconnected component tree of K_5-free graphs Datta et. al. [DLN$^+$09] gave a unique decomposition of planar graphs into biconnected and these further into triconnected components witch 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 3.16 *[TW09] The triconnected component tree for a K_5-free biconnected graph can be computed in log-space.*

For technical reasons we make the following changes to this tree structure. Let B be a biconnected K_5-free graph with G_0 a triconnected non-planar component node in $\mathcal{T}^{\mathrm{T}}(B)$. In $\mathcal{T}^{\mathrm{T}}(B)$ there is a separating pair node s for each edge which is part of a 3-divisive separating triple in G. In $\mathcal{T}^{\mathrm{T}}(B)$ the node s is connected to the node G. We call s a *leaf separating pair* of $\mathcal{T}^{\mathrm{T}}(B)$ if it is connected to only one component node. With Theorem 3.15 we get the following.

Claim 3.17 *The set of leaf separating pairs can be computed in log-space.*

4 Canonization of $K_{3,3}$-free and K_5-free graphs

We describe the isomorphism order and canonization of $K_{3,3}$-free and K_5-free graphs.

4.1 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, see Section 3.1. We extend the algorithm of [DLN$^+$09] to $K_{3,3}$-free graphs. We show how to compare and canonize K_5-components.

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 labelling 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 can be no 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 $S_{G_i} =_{\mathrm{T}} T_{H_j}$.

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 $\mathcal{T}^{\mathrm{T}}(B)$.

- 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 $\mathcal{T}^{\mathrm{T}}(B)$. 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 $\mathcal{T}^{\mathrm{T}}(B)$.

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

4.2 Isomorphism order of K_5-free graphs

4.2.1 Isomorphism order of K_5-free 3-connected graphs

The isomorphism order of two triconnected component trees $S_{\{a,b\}}$ and $T_{\{a',b'\}}$ rooted at separating pairs $s = \{a, b\}$ and $t = \{a', b'\}$ is defined the same way as for planar graphs in [DLN$^+$09]. We use the triconnected component tree from Lemma 3.16 with the addition of leaf separating pairs and define $S_{\{a,b\}} <_{\mathrm{T}} T_{\{a',b'\}}$ if:

1. $|S_{\{a,b\}}| < |T_{\{a',b'\}}|$ or
2. $|S_{\{a,b\}}| = |T_{\{a',b'\}}|$ but $\#s < \#t$ or
3. $|S_{\{a,b\}}| = |T_{\{a',b'\}}|$, $\#s = \#t = k$, but $(S_{G_1}, \ldots, S_{G_k}) <_{\mathrm{T}} (T_{H_1}, \ldots, T_{H_k})$ lexicographically, where we assume that $S_{G_1} \leq_{\mathrm{T}} \ldots \leq_{\mathrm{T}} S_{G_k}$ and $T_{H_1} \leq_{\mathrm{T}} \ldots \leq_{\mathrm{T}} T_{H_k}$ are the ordered subtrees of $S_{\{a,b\}}$ and $T_{\{a',b'\}}$, respectively. 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. $|S_{\{a,b\}}| = |T_{\{a',b'\}}|$, $\#s = \#t = k$, $(S_{G_1} \leq_{\mathtt{T}} \dots \leq_{\mathtt{T}} S_{G_k}) =_{\mathtt{T}} (T_{H_1} \leq_{\mathtt{T}} \dots \leq_{\mathtt{T}} T_{H_k})$, but $(O_1, \dots, O_p) < (O'_1, \dots, O'_p)$ 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.

For the notion of orientation counters see [DLN$^+$09]. If two subtrees are isomorphic then an isomorphism enforces the root $\{a,b\}$ to be swapped. Hence, we count how the isomorphic subtrees are connected to $\{a,b\}$

We say that two triconnected component trees S_e and $T_{e'}$ are *equal according to the isomorphism order*, denoted by $S_e =_{\mathtt{T}} T_{e'}$, if neither $S_e <_{\mathtt{T}} T_{e'}$ nor $T_{e'} <_{\mathtt{T}} S_e$ holds.

There is one difference in step 3 which we describe now in more detail. When comparing nodes of the tree we first distinguish between the new types of nodes. We define planar triconnected components $<_{\mathtt{T}}$ V_8-components $<_{\mathtt{T}}$ non-planar 3-connected components.

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.

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 10: there are $2 \cdot 6!$ way of labelling the vertices of G_i when one of (a,b) or (b,a) is fixed. 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':

$$C_0 = (v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8).$$

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

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

- There are no further Hamiltonian cycles.

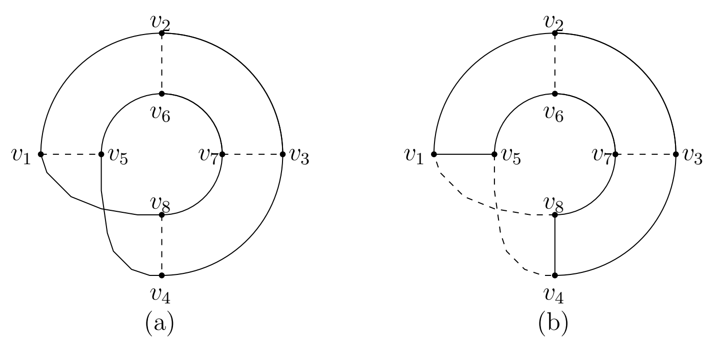

Figure 7: *(a) A V_8 component where solid edges indicate the Hamiltonian cycle C_0.*
(b) A V_8 component where solid edges indicate the Hamiltonian cycle $C_2[\{v_1, v_5\}, \{v_4, v_8\}]$.

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

- **Case $\{a, b\} \notin E'$:** We use cycle C_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

  Then, $Code(G_i, (v_1, v_2))$ 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

  Then $Code(G_i, (v_1, v_5))$ 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

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, respectively.

We define the orientation which is given to the parent separating pairs s and t. Then we describe the comparison algorithm of G_i with H_j. We partition G_i and H_j into their four-connected components and consider their four-connected component trees $\mathcal{T}^{\mathtt{F}}(G_i)$ and $\mathcal{T}^{\mathtt{F}}(H_j)$.

Overview of the steps in the isomorphism order. The isomorphism order of two four-connected component trees S_τ and $T_{\tau'}$ rooted at 3-divisive separating triples τ and τ' where given an order $\mathsf{order}(\tau) \in Sym(\tau)$ and $\mathsf{order}(\tau') \in Sym(\tau')$ is defined $S_\tau \leq_{\mathtt{F}} T_{\tau'}$ if:

1. $|S_\tau| < |T_{\tau'}|$ or
2. $|S_\tau| = |T_{\tau'}|$ but $\#\tau < \#\tau'$ or
3. $|S_\tau| = |T_{\tau'}|$, $\#\tau = \#\tau' = k$, but $(S_{F_1}, \dots, S_{F_k}) <_{\mathtt{F}} (T_{F'_1}, \dots, T_{F'_k})$ lexicographically, where we assume that $S_{F_1} \leq_{\mathtt{F}} \dots \leq_{\mathtt{T}} S_{F_k}$ and $T_{F'_1} \leq_{\mathtt{T}} \dots \leq_{\mathtt{T}} T_{F'_k}$ 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. Note, that these children are again separating triple nodes.
4. $|S_\tau| = |T_{\tau'}|$, $\#\tau = \#\tau' = k$, $(S_{F_1} \leq_{\mathtt{F}} \dots \leq_{\mathtt{F}} S_{F_k}) =_{\mathtt{F}} (T_{F'_1} \leq_{\mathtt{F}} \dots \leq_{\mathtt{F}} T_{F'_k})$, but the following holds.

   For all i, the return value from the recursion of S_{F_i} with $T_{F'_i}$ is an *orientation graph* X_i and X'_i with $V(X_i) = \tau$ and $V(X'_i) = \tau'$ and colored edges, respectively. We compute a *reference orientation graph* X and X' from all the X_i and X'_i. We compare lexicographically whether X with $\mathsf{order}(\tau) < X'$ with $\mathsf{order}(\tau')$. We describe the notion of (reference) orientation graph and $\mathsf{order}(\tau)$ below in more detail.

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.

Orientation given to the parent separating pair by a non-planar component with 3-divisive separating triples Given two non-planar 3-connected components G_i and H_j, we decompose them into $\mathcal{T}^{\mathtt{F}}(G_i)$ and $\mathcal{T}^{\mathtt{F}}(H_j)$. There is a set of candidates for root separating triples such that we obtain the minimum codes when the trees are rooted at them. For the isomorphism order procedure, we give distinct colors to the parent separating pair and the parent articulation point in the trees. We also have colors for the child separating pairs and child articulation points, according to their isomorphism order. We recompute these colors by interrupting the current isomorphism order procedure and going into recursion at the corresponding subtrees.

Finally, we consider the first occurrence of the parent separating pair in all the minimum codes. If the first occurrence is (a, b) in this direction in all the codes, then G_i gives this direction to the parent. If both (a, b) and (b, a) occur first in different minimum codes, then there is no orientation given to the parent.

Isomorphism order of four-connected component trees We describe the differences to the isomorphism order algorithm for triconnected component trees in Section 4.2.1 (see also [DLN$^+$09]). Figure 8 shows two trees to be compared.

- Instead of separating pairs we have 3-divisive separating triples. Therefore, we have to generalize the notion of orientations.

  Remark 4.2 *In the isomorphism order algorithm for two triconnected component trees, an orientation of a separating pair $\{a, b\}$ is computed, c.f. [DLN$^+$09]. The* orientation

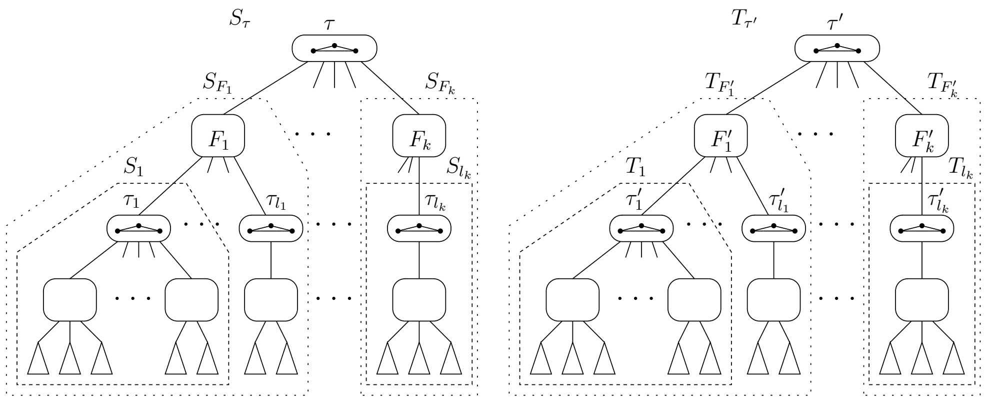

Figure 8: *The four-connected component trees.*

is one of (a, b) or (b, a) or no orientation is given from a child component, also called the symmetric case. *An orientation can be seen as an automorphism from $T_{\{a,b\}}$ onto the tree-canon of $T_{\{a,b\}}$, keeping the pair $\{a, b\}$ blockwise fixed. Observe, the three mentioned possibilities describe a set of partial automorphisms restricted to the pair $\{a, b\}$ in $T_{\{a,b\}}$. Each partial automorphism can be extended to an automorphism ϕ which brings $T_{\{a,b\}}$ to its tree-canon. We say, ϕ computes a* canonical form *for $T_{\{a,b\}}$. Let A be this set, it forms a coset, i.e. a group where each element is multiplied with a permutation. We give examples, where we use the cycle notation for permutations.*
(a, b)-orientation: $A = \{id\}$, the coset is $\langle id\rangle \circ id$,
reverse orientation: $A = \{(a, b)\}$, the coset is $\langle id\rangle \circ (a, b)$,
symmetric case: $A = \{id, (a, b)\}$, the coset is $Sym(\{a, b\}) \circ id$.

For a four-connected component tree S_τ with $\tau = \{a, b, c\}$, we describe now the analogon of orientations*: Accordingly, we have a set of partial automorphisms, each mapping $\{a, b, c\}$ onto $\{a, b, c\}$ which can be extended to an automorphism which computes a canonical form for S_τ. Let A be this set, it forms a coset. The analogon of the* symmetric case *is more subtile here. We give some examples:*
$A = \{(a, b, c)\}$, the coset is $\langle id\rangle \circ (a, b, c)$.
$A = \{id, (a, b, c), (a, c, b)\}$, the coset is $\langle (a, b, c)\rangle \circ id$.
$A = \{id, (a, b)\}$, the coset is $Sym(\{a, b\}) \circ id$ where c is pointwise fixed.
$A = \{(a, b), (a, b, c)\}$, the coset is the group $Sym(\{b, c\}) \circ (a, b)$.
The analogon to the symmetric case here is considering all the cosets, where A has more than one element.

- Instead of 3-connected planar components we have four-connected planar components in S_τ and $T_{\tau'}$. We can canonize these components with the canonization algorithm for 3-connected planar components from [DLN08].

Isomorphism order of two subtrees rooted at four-connected component nodes. We consider the isomorphism order of two subtrees S_{F_i} and $T_{F'_j}$ rooted at four-connected component nodes F_i and F'_j, respectively. To canonize a four-connected component F_i, we use the log-space algorithm from [DLN08]. Besides F_i, the algorithm gets as input a starting edge and a combinatorial embedding ρ of F_i. Let $\tau = \{a, b, c\}$ be the parent 3-divisive separating triple of F_i. 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_i.

We start the canonization of G_i and H_j in all the possible ways and compare these codes bit-by-bit. Let C and C' be two codes to be compared. The base case is that F_i and F'_j are leaf nodes and therefore contain no further virtual edges. In this case we use the lexicographic order between C and C'.

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

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 the virtual edges $\{u, v\}$ and $\{u', v'\}$ then we do the following. We consider only the child separating triples which do not have virtual edges considered earlier in the codes C and C'. We order the child separating triples according to the positions of all their virtual edges in the codes. We call this order the *position-order.* W.l.o.g. let τ_{i_0} (in C) and τ'_{j_0} (in C') be the separating triples which come first in this position-order.

   For τ_{i_0} and τ'_{j_0}, we will define below the *reference orientation graphs* X and X' with $V(X) = \tau_{i_0}$ and $V(X') = \tau'_{j_0}$, respectively. For all pairs in $\tau_{i_0} = \{u, v, w\}$ and $\tau'_{j_0} = \{u', v', w'\}$ we have virtual edges in C and C'. Consider the order of these virtual edges $\mathsf{order}(\tau_{i_0})$ and $\mathsf{order}(\tau'_{j_0})$ in the codes C and C', respectively. We compare X and X' with respect to these orders. This is described below in more detail. If we find an inequality, say $X < X'$ with respect to the vertex ordering, then C is defined to be the *smaller code.* Equivalently, if $X' < X$ with respect to the vertex ordering, then C' is the smaller code. Proceed with the next separating triples in the position-order until we ran through all of them.

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 F_i and F'_j then we define $S_{F_i} =_{\mathsf{F}} T_{F'_j}$. The construction of the codes also defines an isomorphism between the subgraphs described by 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 $S_{F_i} =_{\mathsf{F}} T_{F'_j}$ guarantees that we can combine the isomorphisms of the components to an isomorphism between $\mathsf{graph}(S_{F_i})$ and $\mathsf{graph}(T_{F'_j})$.

Finally, we define the *orientation given to the parent separating triple of* F_i and F'_j as follows.

- We compute an *orientation graph* X_i with $V(X_i) = \tau$.

- For each pair in τ when taken as starting edge for the canonization of S_{F_i} which leads to a minimum code (among all the codes for these edges) we have a directed edge in $E(X_i)$ with color (1).

- Also for the r-th minimum codes we have a directed edge in $E(X_i)$ with color (r), for all $1 \leq r \leq 6$. Here, 6 is the number of directed edges in τ.

For example, the reference orientation can define a complete ordering on the vertices of τ if X_i is a single colored complete graph. And there is a partial order if exactly one directed edge in $E(X_i)$ has a second color. If $\{(u,v),(v,w),(w,u)\}$ have color (1) and the other edges have color (2) then there is no ordering of the vertices, we rather have a cyclic rotation of the vertices of τ, i.e. a subgroup of $Sym(\tau)$.

We define a new graph X_i with $V(X_i) = \tau$ and X'_j with $V(X'_j) = \tau'$. For each of the remaining minimum codes we have a unique starting edge and this edge is also contained as a directed edge in X_i or X'_j, respectively.

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.

Isomorphism order of two subtrees rooted at separating triple nodes. We assume that the subtrees $S_{F_1}, \ldots, S_{F_k}$ and $T_{F'_1}, \ldots, T_{F'_k}$ of the roots τ and τ' are partitioned 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 with the smallest index i defines the 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'. We already explained above the orientation given by each of the S_{F_i}'s to τ. We define a *reference orientation* for the root nodes τ and τ' which is given by their children. This is done as follows, it is encoded in a graph. We partition $(S_{F_1}, \ldots, S_{F_k})$ into classes of isomorphic subtrees, say $I_1 <_{\mathtt{F}} \ldots <_{\mathtt{F}} I_p$ for some $p \leq k$, and similarly we partition $(T_{F'_1}, \ldots, T_{F'_k})$ into $I'_1 <_{\mathtt{F}} \ldots <_{\mathtt{F}} I'_p$. It follows that I_j and I'_j contain the same number of subtrees for every j.

- Consider the orientation given to τ by an isomorphism class I_j: For each child F_i which belongs to I_j we compute an *orientation graph* X_i with vertices $V(X_i) = \tau$. The orientation graph is defined as above but with the following changes. Instead of colors $(1), \ldots, (6)$ we have the colors $(j,1), \ldots, (j,6)$ for the edges.
- The *reference orientation given to* τ is defined as follows. We define the orientation graph X with vertices $V(X) = \tau$ and edges $E(X) = \bigcup_{1 \leq i \leq k} E(X_i)$ the disjoint union of the edges of the orientation graphs from all children of τ. Thus, X has multiple edges. We call X the *reference orientation graph for* τ.

Comparison of two orientation graphs. If τ is the root of a four-connected component tree then the isomorphism order algorithm computes X and X' and compares them for isomorphism. Assume now τ and τ' have isomorphic subtrees and have parent nodes F and F'. In this situation we return from recursion with $=_{\mathtt{F}}$ and give the orientation graphs X and X' to the parent.

We went into recursion because the virtual edges of τ and τ' appeared in the same positions in the codes of the parent. In these codes, we have a complete order on the vertices of τ and

τ'. Let $V(X) = \{u, v, w\}$ and let $\mathsf{order}(\tau) = u < v < w$ be an order of τ. We compute a list of counters for $(X, \mathsf{order}(\tau))$ as follows.

- We order the edges of X according to the order of their vertices, lexicographically. That is, $(u, v) < (u, w) < (v, u) < (v, w) < (w, u) < (w, v)$.
- Among directed edges with the same ends, we order them according to their color. That is, an edge with color (i_1, i_2) comes before an edge with color (j_1, j_2) if $(i_1, i_2) < (j_1, j_2)$ lexicographically.
- We define a counter for the number of edges with the same ends and the same color. For the edge (u, v) we have the counters $c_{(u,v),1}, \ldots, c_{(u,v),6p}$. Note, we have at most $6p$ colors because there are 6 colors for edges from orientation graphs of one isomorphism class and there are p isomorphism classes.
- We order the counters according to the order of the edges. That is, we have a list of counters $L(X, \mathsf{order}(\tau)) = (c_{(u,v),1}, c_{(u,v),2}, \ldots, c_{(u,v),6p}, \ldots, c_{(w,v),1}, \ldots, c_{(w,v),6p})$.

Note, among isomorphic graphs, there must be edges having the same color up to a permutation of them. Counting the colored edges allows to combine the orientations of all isomorphic subtrees. Keep into account that, when a orientation graph X_i for τ has two equal colored edges, then there is an automorphism that maps the one edge to the other same colored edge in τ. Furthermore, the permutation of one edge completely fixes the whole automorphism of τ. Hence, also when counting the edges from different orientation graphs X_1 and X_2, if w.l.o.g. there are the edges (u, v) with colors 1 and 2 then the mapping of (u, v) to other edges completely fixes the whole automorphism among τ and whether X_1 and X_2 are swapped. With an inductive argument, this can be generalized to the whole orientation graph X.

Let X' be the corresponding reference orientation graph for τ'. We define the isomorphism order $(X, \mathsf{order}(\tau)) < (X', \mathsf{order}(\tau'))$ exactly when $L(X, \mathsf{order}(\tau)) < L(X', \mathsf{order}(\tau'))$ lexicographically.

The preceding discussion leads to the following theorem which states that two trees are $=_{\mathtt{F}}$-equal, precisely when the underlying graphs are isomorphic.

Theorem 4.3 *The 3-connected non-planar graphs G and H which contained 3-divisive separating triples are isomorphic if and only if there is a choice of separating triples τ, τ' in G and H such that $S_\tau =_{\mathtt{F}} T_{\tau'}$ when rooted at τ and τ', respectively.*

Proof. Assume that $S_\tau =_{\mathtt{F}} T_\tau$. The argument is an induction on the depth of the trees that follows the inductive definition of the isomorphism order. The induction goes from depth d to $d + 2$. If the grandchildren of separating triples, say τ and τ', are $=_{\mathtt{F}}$-equal up to step 4, then we compare the children of s and t. If they are equal then we can extend the $=_{\mathtt{F}}$-equality to the separating triples τ and τ'.

When considering subtrees rooted at separating triple nodes, then the comparison describes an order on the subtrees which correspond to split components of the separating triples. The order describes an isomorphism among the split components.

When considering subtrees rooted at four-connected component nodes, say F_i and F'_j, then 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 onto τ_0'. The isomorphism group of X and X' exactly covers all these mappings. 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 in 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 a mapping of F_i onto F_j'. The $=_{\mathtt{F}}$-equality between child separating triples (together with their subtrees) inductively describes an isomorphism between the corresponding subgraphs.

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.

The reverse direction holds obviously as well. Namely, if G and H are isomorphic and there is an isomorphism that maps the separating triples τ of G to the separating triple τ' of H, then the triconnected component trees S_τ of G and T_τ of H rooted respectively at the triples will clearly be equal. Hence, such an isomorphism maps separating triples of G onto separating triples of H. This isomorphism describes a permutation on the split components of separating triples, which means we have a permutation on four-connected components, the children of the separating triples. By induction hypothesis, the children (at depth $d{+}2$) of two such four-connected components are isomorphic and equal according to $=_{\mathtt{F}}$. More formally, one can argue inductively on the depth of S_τ and $T_{\tau'}$. $\square$

4.3 Complexity of the isomorphism order algorithm

Let S_a be a biconnected component tree at an articulation point a. Let B be a biconnected component in S_a. Let S_B be the subtree of S_a with root B. Let $\mathcal{T}^{\mathrm{T}}(B)$ be the triconnected component tree of B. Let a_0 be an articulation point in B. Then a_0 has a copy in at least one triconnected component in $\mathcal{T}^{\mathrm{T}}(B)$. Actually, all the components which contain a copy of a form a subtree of $\mathcal{T}^{\mathrm{T}}(B)$. The copy of a which belongs to the node of $\mathcal{T}^{\mathrm{T}}(B)$ which is closest to the root is called the *reference copy of a*.

The following size definitions cover exactly those subtrees which are traversed by the isomorphism order algorithm.

Definition 4.4 *([DLN+09]) Let B be a biconnected component node in a biconnected component tree S, and let $\mathcal{T}^{\mathrm{T}}(B)$ be the triconnected component tree of B. The* size of B *is $|\mathcal{T}^{\mathrm{T}}(B)|$, the sum of the sizes of the components corresponding to the nodes in $\mathcal{T}^{\mathrm{T}}(B)$. The size of an articulation point node in S is defined as* 1. *Note that the articulation points may be counted several times, namely in every component they occur. The* size of S, *denoted by $|S|$, is the sum of the sizes of its components.*

In a triconnected component tree, the algorithm goes into recursion at a biconnected subtree only when the uniquely defined reference copy of a child articulation point is reached.

Definition 4.5 *([DLN+09]) Let B be a biconnected component and $\mathcal{T}^{\mathrm{T}}(B)$ its triconnected component tree. Let C be a node in $\mathcal{T}^{\mathrm{T}}(B)$, i.e. a triconnected component node or a separating pair node. The* tree S_C rooted at C *consists of the subtree of $\mathcal{T}^{\mathrm{T}}(B)$ rooted at C (with respect to the root of $\mathcal{T}^{\mathrm{T}}(B)$) and of the subtrees S_a for all articulation points a that have a reference copy in the subtree of $\mathcal{T}^{\mathrm{T}}(B)$ rooted at C, except those S_a that are a large child of S_B. The* size of S_C, *also denoted by $|S_C|$ is the sum of the sizes of its components.*

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. The isomorphism order algorithm explores the biconnected component trees and the triconnected component trees of the components of G. 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.

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 or the $K_{3,3}$. Then the graph 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. If B contains no separating pairs, i.e. B is triconnected, then we handle B as described above in the case when G was triconnected. So assume that B contains separating pairs and let $\mathcal{T}^{\mathtt{T}}(B)$ be the triconnected component tree of B. Also see Figure 9.

To determine roots, we consider the center C of $\mathcal{T}^{\mathtt{T}}(B)$. If C is a separating pair node or triconnected planar component node, then we can find roots in log-space due to Datta et.al. [DLN$^+$09]. If C is the V_8 or the $K_{3,3}$, 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 is non-planar and contains 3-divisive separating triples. Then we consider $\mathcal{T}^{\mathtt{F}}(C)$, the four-connected component tree of C. and invoke the computation of the parent separating triples for the center node 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 root separating triples in $\mathcal{T}^{\mathtt{F}}(C)$, say k triples. Each of the three edges which are part of a 3-divisive separating triple, has a separating pair node adjacent to C. We root $\mathcal{T}^{\mathtt{T}}(B)$ at all these separating pairs. Hence, we get $\leq 3k$ root separating pairs, which will be good enough.

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, and G_0 be a triconnected component in B. We describe how to determine a small number of separating triples as root candidates for G_0. We may assume that G_0 is non-planar, otherwise isomorphism testing is 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 root. If we do this recursively we

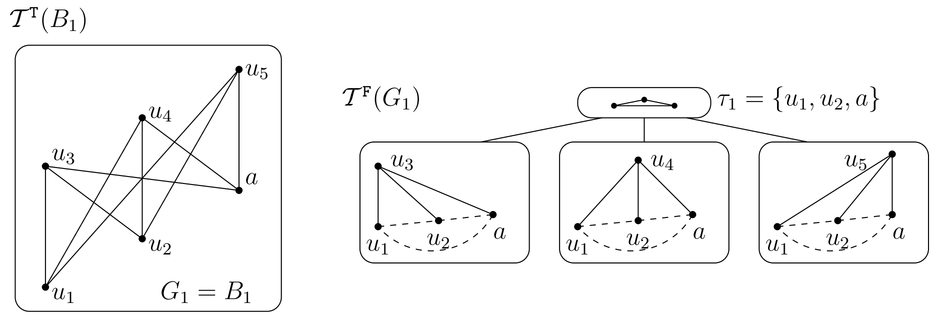

Figure 9: *The biconnected component B_1 contains no separating pairs and is therefore trivially triconnected such that the corresponding triconnected component tree $\mathcal{T}^{\mathtt{T}}(B_1)$ contains only one node, i.e. $G_1 = B_1$. G_1 is further decomposed into four-connected components as illustrated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

end up in a polynomial amount of space on the work-tape.

If G_0 is the $K_{3,3}$, then we have exactly two choices of selecting a root separating triple in the decomposition of G_0. 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 G_0 is non-planar and contains 3-divisive separating triples. We distinguish the cases whether there is a parent separating pair for G_0 in B or not. If G_0 has a parent separating pair we show that we can determine ≤ 4 separating triples as root candidates for $\mathcal{T}^{\mathtt{F}}(G_0)$. The case that G_0 has no parent separating pair can occur in two situations as explained in Section 4.3:

- There is no separating pair in B. Then G_0 is the single triconnected component in B and G_0 contains 3-divisive separating triples.
- There are separating pairs in B and G_0 is the center of the triconnected component tree $\mathcal{T}^{\mathtt{T}}(B)$ which contains 3-divisive separating triples.

In the latter situation the goal is to compute a small set of separating pairs as root candidates for $\mathcal{T}^{\mathtt{T}}(B)$. Because then the overall isomorphism algorithm will return again later to component G_0, but then *with* a parent separating pair for G_0, and hence we are in the first case when we really compare G_0. As already explained above, the small set of separating pairs maybe obtained from a small set of separating triples determined in G_0.

G_0 has a parent separating pair. Let $\{a, b\}$ be the parent separating pair of G_0. We consider the four-connected component tree $\mathcal{T}^{\mathtt{F}}(G_0)$. The nodes of $\mathcal{T}^{\mathtt{F}}(G_0)$ which contain the edge $\{a, b\}$ form a subtree of $\mathcal{T}^{\mathtt{F}}(G_0)$. Of these, the node that is closest to the unique center C_0 of $\mathcal{T}^{\mathtt{F}}(G_0)$ is associated with $\{a, b\}$.

If the center C_0 of $\mathcal{T}^{\mathtt{F}}(G_0)$ is a separating triple, choose this triple as root. Otherwise, the center is a 4-connected planar component.

1. If $\{a, b\} \notin C_0$, then choose the separating triple nearest to C_0 which is on the unique path between C_0 and the 4-connected component associated with $\{a, b\}$.

2. If $\{a, b\} \in C_0$, then canonize C_0 with $\{a, b\}$ as the starting edge. This gives four possible ways of canonization. In the smallest code among these, choose the separating triple in C_0 which gets the lexicographically smallest label. Thus, we have at most four choices for the root.

G_0 has no parent separating pair. We start by defining colors for child articulation points which occur in the component B and for the adjacent separating pairs of G_0 in $\mathcal{T}^{\mathrm{T}}(B)$.

- Let a be the the parent articulation point and $a_1, \dots, a_l$ be the articulation points which are associated with G_0 in $\mathcal{T}^{\mathrm{T}}(B)$. Let a_j be the root node of the biconnected subtree S_{a_j} of S_a.

  We partition the subtrees $S_{a_1}, \dots, S_{a_l}$ into classes $E_1, \dots, E_p$ of equal size subtrees (i.e. size according to Definition 4.4). Let k_j be the number of subtrees in E_j. Let the order of the size classes be such that $k_1 \le k_2 \le \cdots \le k_p$. All articulation points with their subtrees in size class E_j are colored with color j.

- Let $s_1, \dots, s_m$ be the separating pairs which are connected to G_0 in $\mathcal{T}^{\mathrm{T}}(B)$. Let S_{s_j} be the subtree of $\mathcal{T}^{\mathrm{T}}(B)$ rooted at s_j. We partition the subtrees $S_{s_1}, \dots, S_{s_m}$ into classes $\widehat{E}_1, \dots, \widehat{E}_{\hat{p}}$ of equal size subtrees (i.e. size according to Definition 4.5). Let $\widehat{k}_j$ be the number of subtrees in $\widehat{E}_j$. Let the order of the size classes be such that $\widehat{k}_1 \le \widehat{k}_2 \le \cdots \le \widehat{k}_{\hat{p}}$. All virtual edges of separating pairs with their subtrees in size class $\widehat{E}_j$ are colored with color j.

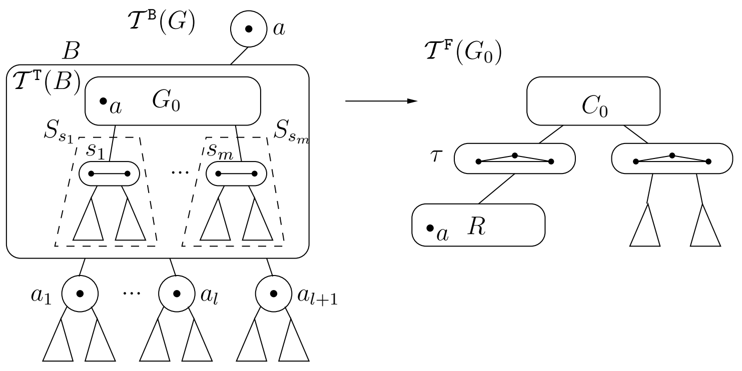
$\mathcal{T}^{\mathrm{B}}(G)$
a
B
$\mathcal{T}^{\mathrm{T}}(B)$
a
G_0
S_{s_1}
s_1
S_{s_m}
s_m
a_1
a_l
a_{l+1}
$\mathcal{T}^{\mathrm{F}}(G_0)$
C_0
τ
a
R

Figure 10: *(a) The decomposition of a graph G into biconnected and triconnected components. In $\mathcal{T}^{\mathrm{B}}(G)$ we have a biconnected component B with its parent articulation point a. B is decomposed into its triconnected component tree $\mathcal{T}^{\mathrm{T}}(B)$. G_0 is a triconnected component in $\mathcal{T}^{\mathrm{T}}(B)$, where $a_1, \dots, a_l$ are child articulation points associated to G_0 and $s_1, \dots, s_m$ are adjacent separating pairs of G_0 in $\mathcal{T}^{\mathrm{T}}(B)$. B may have further child articulation points which are not associated to G_0 (here a_{l+1}). (b) The decomposition of G_0 into its four-connected component tree $\mathcal{T}^{\mathrm{F}}(G_0)$. The parent articulation point is contained in G_0 and hence in $\mathcal{T}^{\mathrm{F}}(G_0)$. The situation is shown where a occurs not in the center C_0 of the tree.*

To limit the number of potential root nodes for $\mathcal{T}^{\mathrm{F}}(G_0)$, we distinguish several cases below. The center C_0 will play an important role thereby. In some of the cases we will show that the number of automorphisms of C_0 is small. This already suffices for our purpose: in this case, for every edge as starting edge, we canonize the component C_0 separately and construct a set of the separating triples A that lead to the minimum code. Although there can be polynomially many possible candidates for the canon, the minimum ones are bounded by the number of automorphisms of C_0, which is small. We take the separating triples as root candidates for $\mathcal{T}^{\mathtt{F}}(G_0)$ 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 C_0.

Consider the case where the parent articulation point a is not associated with C_0. We find the path from the node to which a is associated to C_0 in $\mathcal{T}^{\mathtt{F}}(G_0)$ and find the separating triple closest to C_0 on this path. This serves as the unique choice for the root of $\mathcal{T}^{\mathtt{F}}(G_0)$, see Figure 10.

For the following cases, we assume that the parent articulation point a is associated with C_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 G_0 in $\mathcal{T}^{\mathtt{F}}(G_0)$.

Case I: $l \leq 1$.

1. $m = 0$. If $l = 0$ then G_0 is a leaf node in $\mathcal{T}^{\mathtt{T}}(B)$ and also in $\mathcal{T}^{\mathtt{F}}(G_0)$. We color the parent articulation point a with a distinct color. In this case, we can cycle through all separating triples as root for $\mathcal{T}^{\mathtt{F}}(G_0)$. We can do this because, there is no recursion on biconnected or triconnected subtrees which bounds the number of root separating triples.

   If $l = 1$ then we process the corresponding child articulation point a priori and store the result. We color the parent and the unique child with distinct colors and proceed with G_0 as in the case of a leaf node.

2. $m \geq 1$. That is, we have separating pairs although we have no parent separating pair. In this case we return the separating pairs which are the roots of the subtrees in $\widehat{E}_1$ as root candidates for $\mathcal{T}^{\mathtt{T}}(B)$.

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

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

2. $k_1 \leq \widehat{k}_1$. We consider the following sub-cases.

   (a) **Some articulation point a_j in E_1 is not associated with C_0.** Let a_j be associated with a four-connected component $D \neq C_0$. Find the path from D to C_0 in $\mathcal{T}^{\mathtt{F}}(G_0)$ and select the separating triple node closest to C_0 on this path. Thus a_j uniquely defines a separating triple. In the worst case, this may happen for every a_j in E_1. Therefore, we get up to k_1 separating triples as candidates for the root.

   (b) **All articulation points in E_1 are associated with C_0.**

   - $k_1 \geq 2$. Every automorphism of C_0 fixes the parent articulation point a and setwise fixes the k_1 articulation points in E_1. By Lemma 4.6 below, there are at most $4k_2$ automorphisms of C_0.

- $k_1 = k_2 = 1$. Every automorphism of C_0 fixes the parent articulation point a and the two articulation points in E_1 and E_2. By Corollary 4.7 below, C_0 has at most one non-trivial automorphism in this case.
- $k_1 = 1$ **and** $1 < k_2 < \widehat{k}_1$**.** By an analogous argument as in the case where $k_1 \geq 2$, C_0 has at most $4k_2$ automorphisms.
- $k_1 = 1$ **and** $\widehat{k}_1 \leq k_2$ **and** $1 < k_2$**.** We distinguish two sub-cases.
  - **Some separating pair** s_j **in** $\widehat{E}_1$ **is not associated with** C_0**.** Choose the separating triple nearest to C_0 which is on the unique path between C_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** C_0**.** Every automorphism of C_0 fixes the parent articulation point a and setwise fixes the $2\widehat{k}_1$ separating pairs in $\widehat{E}_1$. By Lemma 4.6 below, there are at most $4 \cdot 2k_2$ automorphisms C_0.

We observe that the size of the root set A is $\leq 8k_1$ if $k_1 > 1$ and $\leq \min\{8k_2, 8\widehat{k}_1\}$ if $k_1 = 1$. Since the subtrees in E_i (and $\widehat{E}_i$) have size $\leq n/k_i$ (and $\leq n/\widehat{k}_i$ respectively), we have that $|A| \leq 8|E|$, where E is the class of equal sized biconnected subtrees which have maximum size and $|E| > 1$. The following lemma bounds the number of automorphisms for the center C.

Lemma 4.6 *([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 4.7 *([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 4.6 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 are equal for the isomorphism order for some choice of the root, then the corresponding graphs are isomorphic. The reverse direction clearly holds as well.

Theorem 4.8 *Given two 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 4.3 for the correctness of isomorphism order of four-connected component trees, and assume the correctness of the isomorphism order of 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 G_0 and H_j, the comparison states equality if the components are isomorphic. If G_0 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 G_0 and H_j, but also checks whether the children of G_0 and H_j mapped to each other are indeed isomorphic, and also ensures that parents of G_0 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 $T_{\{a',b'\}}$. $\square$

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

When we compare triconnected component trees $\mathcal{T}^{\mathtt{T}}(B)$ and $\mathcal{T}^{\mathtt{T}}(B')$, then we might find several copies of articulation points a and 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.

We have the same situation for articulation points and separating pairs in four-connected component trees $\mathcal{T}^{\mathtt{F}}(G_0)$ and $\mathcal{T}^{\mathtt{F}}(H_j)$ of two triconnected components G_0 and H_j. The separating pairs are adjacent to G_0 and H_j. The articulation points are adjacent to the biconnected components B and B', where G_0 and H_j are nodes in the trees $\mathcal{T}^{\mathtt{T}}(B)$ and $\mathcal{T}^{\mathtt{T}}(B')$, respectively.

Datta et.al.[DLN+09] defined a *reference copy* of a which is a unique copy in $\mathcal{T}^{\mathtt{F}}(G_0)$. More precisely, a is contained in the node A in $\mathcal{T}^{\mathtt{F}}(G_0)$ which is closest to the root in this tree. This always is a triconnected component node or the root node itself. Their algorithm goes only into recursion at S_a when it reaches a in this node A. We go into recursion at the first edge where a or $\{a, b\}$ occurs, when we examine A. This copy of a is the reference copy.

We will go into recursion either directly when we reach $\mathcal{T}^{\mathtt{F}}(G_0)$ in the case that S_a or $S_{\{a,b\}}$ is a large child of G_0, or at the reference copy in $\mathcal{T}^{\mathtt{F}}(G_0)$. The first case will be described in detail below on page 29. Note, the reference copy of a or $\{a, b\}$ depends on the chosen root of $\mathcal{T}^{\mathtt{F}}(G_0)$.

The following lemma is adapted from [DLN+09]. It is generalized from triconnected to four-connected component trees and also from articulation points to the recomputation of the reference copies of separating pairs when returning from recursion.

Lemma 4.9 *The reference copy of an articulation point a or a separating pair $\{a, b\}$ in $\mathcal{T}^{\mathtt{F}}(G_0)$ and a' or $\{a', b'\}$ in $\mathcal{T}^{\mathtt{F}}(H_j)$ for the comparison of four-connected component trees $\mathcal{T}^{\mathtt{F}}(G_0)$ with $\mathcal{T}^{\mathtt{F}}(H_j)$ 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 $\mathcal{T}^{\mathtt{F}}(G_0)$. Assume, that we have the same situation for a' and $\{a', b'\}$ in $\mathcal{T}^{\mathtt{T}}(B')$. If not, then we found an inequality. We define now a unique component A, where a or $\{a, b\}$ is contained. We distinguish the following cases. Let P be one of a or $\{a, b\}$ and P' one of a' or $\{a', b'\}$.

- P occurs in the root separating triples of $\mathcal{T}^{\mathtt{F}}(G_0)$. That is, P occurs already at the beginning of the comparisons for $\mathcal{T}^{\mathtt{F}}(G_0)$. Then we define A as the root separating triple.
- P occurs in separating triples other than the root of $\mathcal{T}^{\mathtt{F}}(G_0)$. Then P occurs in all the component nodes, which contain such a separating triple. These nodes form a connected subtree of $\mathcal{T}^{\mathtt{F}}(G_0)$. Hence, one of these component nodes is the closest to the root of $\mathcal{T}^{\mathtt{F}}(G_0)$. This component is always a four-connected component node. Let A 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 $\mathcal{T}^{\mathtt{F}}(G_0)$. Let A be this component.

In all except the first case, we find P in a four-connected component node A first. Let P' be found first in component node A', accordingly. Assume, we start the comparison of A and A'. More precisely, we start to compare the codes C of A and C' of A' bit for bit. We go into recursion if and only if we reach the first edge in the codes which contain P and P'. Note, if P is an articulation point a, then C can contain more than one edge with endpoint a. On all the other edges in C and C' we do not go again into recursion. It is easy to see, that we can recompute the first occurrence of P in A and of P' in A'. □

Comparing two subtrees rooted at several component nodes.

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].

- 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 component nodes. 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 are the current codes. 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.

Planar component nodes, separating pair nodes or articulation point nodes. The tasks for these types of nodes remain unchanged to the situation where we consider planar graphs. Therefore, we refer to the complexity analysis part in [DLN$^+$09].

Large children.

We deviate from the algorithm described so far in the case that the recursion would lead to a large child. We already defined the size of trees S_a, S_B and S_{G_i} with an articulation point a, a biconnected component B or a triconnected non-planar component G_i (which is not the V_8) as root. A *large child* of such a tree of size N is a child of size $\geq N/2$.

In the case of planar graphs, Datta et.al. [DLN$^+$09] considered for $\mathcal{T}^{\mathtt{T}}(B)$ the nodes for a triconnected component A and a separating pair $\{u, v\}$. They defined the subtrees S_A and $S_{\{}u, v\}$ rooted at A and $\{u, v\}$ respectively in a careful way.

For S_A they consider $S_{\{}u, v\}$ as a subtree if $\{u, v\}$ is a child separating pair of A For articulation points, there are some special properties. Let C be one of A or $\{u, v\}$. Let a_0 be a child articulation point of B. Some copies of a_0 may occur in several components of $\mathcal{T}^{\mathtt{T}}(B)$. Assume, a_0 is associated to C in $\mathcal{T}^{\mathtt{T}}(B)$. Then, S_{a_0} is considered to be a subtree of S_C. But there is one exception where we go into recursion at S_{a_0} even before. a_0 is not considered as a subtree of S_C if a_0 is a large child of B. Then we go into recursion for S_{a_0} a priori, even before we compute the triconnected component tree for B. In this case the comparison result (with some large child $S_{a'}$ of B') is already stored on the work-tape and we do not visit S_a a second time. Consequently, we consider S_a as a subtree only at the place where we go into recursion to S_a. Recall, this is not a static property, as e.g. the position of the reference copy depends on the chosen root of the tree, and we try several possibilities for the root.

We extend this now to K_5-free graphs and the three types of tree decompositions. For triconnected component trees, the situation is the same as described so far. For the four-connected component tree $\mathcal{T}^{\mathtt{F}}(G_i)$, we define S_{F_i} and S_τ as the subtrees of $\mathcal{T}^{\mathtt{F}}(G_i)$ when rooted at the four-connected component node F_i and at the separating triple node τ, respectively. Let C be a node in $\mathcal{T}^{\mathtt{F}}(G_i)$. If there is a reference copy of an articulation point a_0 or of a separating pair $\{u, v\}$ contained in C, then S_{a_0} and $S_{\{u,v\}}$ are considered to be subtrees of S_C. There are exceptions if the following situations occur.

- S_{a_0} is not a subtree of S_C if a_0 is a large child of B. Then S_{a_0} is treated a priori before computing the triconnected component tree of B.

- S_{a_0} is not a subtree of S_C if a_0 is a large child of G_i. Then S_{a_0} is treated a priori before computing the four-connected component tree of G_i.

- $S_{\{u,v\}}$ is not a subtree of S_C if $\{u, v\}$ is a large child of G_i. Then $S_{\{u,v\}}$ is treated a priori before computing the four-connected component tree of G_i.

Following the same arguments, we consider S_{a_0} and $S_{\{u,v\}}$ as a subtree only at the place where we go into recursion. Figure 11 shows an example. Also, this is not a static property. The position of the reference copy or whether a subtree is a large child, depends for example on the chosen roots of the triconnected and four-connected component trees.

We extend now Definition 4.5 to four-connected component trees.

Definition 4.10 *Let B be a biconnected component and $\mathcal{T}^{\mathtt{T}}(B)$ its triconnected component tree. Let G_i be a non-planar* 3*-connected component node in $\mathcal{T}^{\mathtt{T}}(B)$ with* 3*-divisive separating triples. Let $\mathcal{T}^{\mathtt{F}}(G_i)$ be the four-connected component of G_i. Let C be a node in $\mathcal{T}^{\mathtt{F}}(G_i)$, i.e. a four-connected component node or a separating triple node. The* tree S_C rooted at C *consists of the subtree of $\mathcal{T}^{\mathtt{F}}(G_i)$ rooted at C (with respect to the root of $\mathcal{T}^{\mathtt{F}}(G_i)$) and of*

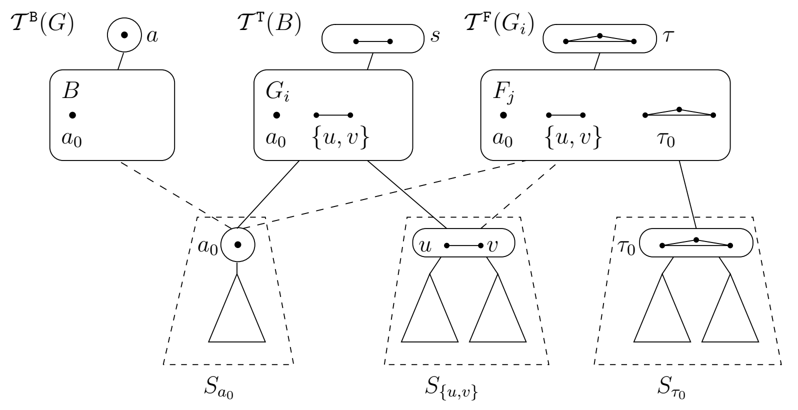

Figure 11: *The figure shows a series of decompositions upto four-connected components. The situation is shown where a_0 is a child articulation point of B and which has reference copies in G_i and F_j. From left to right, if it is a large child of B, G_i or F_j then it is treated at the corresponding position as large child a priori. Here, a_0 is no large child of B but a large child of G_i. The situation is similar for the separating pair $\{u,v\}$, which is a child of G_i and has a reference copy in F_j. Here, $\{u,v\}$ is a large child of G_i. 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.*

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

The size of S_C *is the sum of the sizes of its components.*

Whenever the algorithm reaches a node for a,$\{u,v\}$,B or C as above, it first checks whether the corresponding tree S_a,$S_{u,v}$,S_B, or S_C has a large child and treats it a priori. The result is stored with $O(1)$ bits. In the case of triconnected components, we also store the orientation. We distinguish large children as follows.

- Large children with respect to the biconnected component tree. These are children of node a in S_a or B in S_B. These children are biconnected component nodes or articulation point nodes. When comparing S_B with $S_{B'}$, then we go for large children into recursion before computing the trees $\mathcal{T}^{\mathrm{T}}(B)$ and $\mathcal{T}^{\mathrm{T}}(B')$.
- Large children with respect to the triconnected component tree. These are children of node C in S_C. These children are separating pair nodes, triconnected component nodes or reference copies of articulation point nodes in C.
- Large children with respect to the four-connected component tree. These are children of node C in S_C. These children are separating triple nodes, four-connected component nodes or reference copies of separating pair nodes or articulation point nodes in C.

Analysis of space requirement.

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

4.4 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 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 that order in which the child separating pairs appear in the canon of G_0.

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 $\mathcal{T}^{\mathtt{F}}(G_i)$.

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 4.12 *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.

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

[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.

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

[TW09] Thomas Thierauf and Fabian Wagner. Reachability in $K_{3,3}$-free graphs and K_5-free graphs is in unambiguous log-space. 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.