CHICAGO JOURNAL OF THEORETICAL COMPUTER SCIENCE 2014, Article 2, pages 1–29
http://cjtcs.cs.uchicago.edu/

Reachability in $K_{3,3}$-free and K_5-free Graphs is in Unambiguous Logspace *

Thomas Thierauf Fabian Wagner

Received May 30, 2012; Revised October 14, 2013, February 8, 2014, March 3, 2014, received in final form April 10, 2014; Published April 14, 2014

Abstract: We show that the reachability problem for directed graphs that are either $K_{3,3}$-free or K_5-free is in unambiguous log-space, $\mathsf{UL} \cap \mathsf{coUL}$. This significantly extends the result of Bourke, Tewari, and Vinodchandran that the reachability problem for directed planar graphs is in $\mathsf{UL} \cap \mathsf{coUL}$.

Our algorithm decomposes the graphs into biconnected and triconnected components. This gives a tree structure on these components. The non-planar components are replaced by planar components that maintain the reachability properties. For K_5-free graphs we also need a decomposition into 4-connected components. Thereby we provide a logspace reduction to the planar reachability problem.

We show the same upper bound for computing distances in $K_{3,3}$-free and K_5-free directed graphs and for computing longest paths in $K_{3,3}$-free and K_5-free directed acyclic graphs.

Key words and phrases: Reachability, logspace, planar graphs, $K_{3,3}$-free graphs, K_5-free graphs

1 Introduction

In this paper we consider the *reachability problem on graphs*:

> Reachability
> *Input:* a graph $G = (V, E)$ and two vertices $s, t \in V$.
> *Question:* is there a path from s to t in G?

*Supported by DFG grants TH 472/4-1 and TO 200/2-2.

 DOI: 10.4086/cjtcs.2014.002

The complexity of Reachability is essentially solved for directed and undirected graphs. In general, i.e. for directed graphs, Reachability is complete for the class nondeterministic logspace, NL. For undirected graphs, the complexity was open for a long time, until Reingold [22] showed in a break-through result that Reachability is complete for the class logspace, L. This result is one of the major tools in our paper.

The complexity of Reachability in planar graphs is still not completely settled. Bourke, Tewari and Vinodchandran [7] proved that Reachability on planar graphs is in the class unambiguous nondeterministic logspace, UL $\cap$ coUL. It is also known to be hard for L. They built on work of Reinhard and Allender [23] and Allender, Datta, and Roy [2]. Jacoby and Tantau [16] showed that for series-parallel graphs, reachability is complete for L.

More recently, Allender et.al. [1] showed that Reachability for graphs embedded on the torus is logspace reducible to the planar case. Kynčl and Vyskočil [18] generalized this result to graphs embedded on a fixed surface of arbitrary genus. Das, Datta and Nimbhorkar [9] showed that Reachability for graphs with bounded treewidth is in logspace, when a tree decomposition is given. Elberfeld, Jacoby and Tantau show in [12] that a tree decomposition can be computed in logspace.

We study Reachability on extensions of planar graphs. Our main result is logspace reduction from Reachability for $K_{3,3}$-free and K_5-free graphs to planar Reachability. Thus, the current upper bound for planar Reachability, UL $\cap$ coUL, carries over to Reachability for $K_{3,3}$-free and K_5-free graphs. One motivation for our results is to improve the complexity upper bounds of certain reachability problems, from NL to UL in this case. The major open question is whether one can extend our results further such that we finally get a collapse of NL to UL.

Our technique is to decompose a given graph G into its triconnected components. We show that this can be done in logspace, even for general graphs. Then we exploit the properties of the triconnected components.

- In the case of a $K_{3,3}$-free graph G, Asano [4] showed that the triconnected components of G are either planar or the K_5.
- In the case of a K_5-free graph G, it follows from a theorem of Wagner [26] (cf. Khuller [17]) that there can be nonplanar triconnected components of G of two types only:
  - either they are isomorphic to the Möbius ladder M_8, (see Figure 6 on page 17),
  - or they can be decomposed into 4-connected components which are all planar.

  The M_8 contains a $K_{3,3}$ and is therefore nonplanar.

We also show that the decomposition into 4-connected components in the last item can be done in logspace.

Because we want to reduce the reachability problem to planar reachability, the obstacles we have at this point are the K_5-components in the $K_{3,3}$-free case and the M_8-components in the K_5-free case. We construct planar gadget graphs which we use to replace these nonplanar components. Then we recombine the components.

There are several restrictions for the gadgets that have to be taken care of. Clearly, the original reachability problem should not be altered by the replacement. To make the gadgets planar, some edges of the original graphs occur several times as copies. With the copied edges, we also copy subgraphs attached to the endpoints of the edges. Since the replacement is done recursively in the construction algorithm, the gadget has to be designed in a way that large subgraphs are not copied. Otherwise this would not work in logspace.

We also consider the problem of computing *distances* in a graph. Jacoby and Tantau [16] showed that for series-parallel graphs the distance problem is complete for L. Thierauf and Wagner [24] proved that the distance problem for planar graphs is in $\mathsf{UL} \cap \mathsf{coUL}$. We will see that our transformations from $K_{3,3}$-free or K_5-free graphs to planar graphs maintain not just reachability, but also the distances between vertices. Therefore it follows from our results that distances in $K_{3,3}$-free or K_5-free graphs can be computed in $\mathsf{UL} \cap \mathsf{coUL}$.

Another related problem is to compute *longest paths*. In general, it is NP-complete, for directed and undirected graphs. But it is NL-complete for directed acyclic graphs (DAG). For series-parallel graphs it is complete for L [16]. Limaye, Mahajan and Nimbhorkar [19] prove that longest paths in planar DAGs can be computed in $\mathsf{UL} \cap \mathsf{coUL}$. Our transformations from $K_{3,3}$-free or K_5-free graphs to planar graphs also maintain longest paths between vertices. This is easy to see in the case of $K_{3,3}$-free DAGs and requires some extra arguments in the case of K_5-free DAGs. Hence, longest paths in $K_{3,3}$-free or K_5-free DAGs can be computed in $\mathsf{UL} \cap \mathsf{coUL}$.

The paper is organized as follows. Section 2 provides definitions and notations. In Section 3 we show how to decompose a graph into its biconnected and trionnected components in logspace. In Section 4 and 5 we prove that reachability on $K_{3,3}$-free and K_5-free graphs reduces to reachability on planar graphs, respectively. In these sections we also show the results on distances and longest paths.

2 Definitions and Notations

Complexity classes. The class L is the class of languages accepted by deterministic logspace Turing machines and NL by nondeterministic logspace Turing machines. The class UL contains languages accepted by unambiguous nondeterministic logspace machines, i.e. there exists at most one accepting computation path. Whereas NL is known to be closed under complement, this is not known for UL. Therefore we define coUL as the class of complements of languages in UL.

The class L is closed under Turing reductions, i.e. $\mathsf{L}^{\mathsf{L}} = \mathsf{L}$. Similarly $\mathsf{L}^{\mathsf{UL} \cap \mathsf{coUL}} = \mathsf{UL} \cap \mathsf{coUL}$. The composition of two logspace computable functions is computable in logspace.

By $\leq_m^{\mathsf{L}}$ and $\leq_T^{\mathsf{L}}$ we denote logspace many-one and Turing reduction, respectively.

Graphs. A graph $G = (V,E)$ consists of a finite set of vertices $V(G) = V$ and edges $E(G) = E \subseteq V \times V$. For $U \subseteq V$ let $G - U$ be the *induced subgraph* of G on $V - U$. A graph G is called *undirected* if E is symmetric. An undirected graph G is *connected* if there is a path between any two vertices in G.

Let G be undirected and $S \subseteq V$ with $|S| = k$. We call S a *k-separating set*, if $G - S$ is not connected. For $u, v \in V$ we say that *S separates u from v in G*, if $u \in S$, $v \in S$, or u and v are in different components of $G - S$. For sets of vertices $V_1, V_2 \subseteq V$ we say that *S separates V_1 from V_2 in G*, if S separates every $v_1 \in V_1$ from every $v_2 \in V_2$.

A k-separating set is called *articulation point* (or *cut vertex*) for $k = 1$, *separating pair* for $k = 2$, and *separating triple* for $k = 3$.

A graph G is *k-connected* if it contains no $(k-1)$-separating set. Hence a 1-connected graph is simply a connected graph. A 2-connected graph is also called *biconnected*, a 3-connected graph is also called *triconnected*.

Let S be a k-separating set in a k-connected graph G. Let G' be a connected component in $G-S$. A *split graph* or a *split component of* S in G is the induced subgraph of G on vertices $V(G') \cup S$, where we add *virtual edges* between all pairs of vertices in S. Note that the vertices of a separating set S can occur in several split graphs of G.

A $K_{3,3}$*-free graph* is an undirected graph which does not contain a $K_{3,3}$ as a minor. A K_5*-free graph* is an undirected graph which does not contain a K_5 as a minor. In particular, planar graphs are $K_{3,3}$-free and K_5-free [26].

Reachability problems. Let $\mathcal{G}$ be a class of graphs. We consider the following problems restricted to $\mathcal{G}$.

$$\begin{aligned} \mathcal{G}\text{-Reachability} &= \{(G,s,t) \mid G \in \mathcal{G} \text{ contains a path from } s \text{ to } t\} \\ \mathcal{G}\text{-Distance} &= \{(G,s,t,k) \mid G \in \mathcal{G} \text{ contains a path from } s \text{ to } t \text{ of length } \leq k\} \\ \mathcal{G}\text{-Long-Path} &= \{(G,s,t,k) \mid G \in \mathcal{G} \text{ contains a simple path from } s \text{ to } t \text{ of length } \geq k\} \end{aligned}$$

As already mentioned in the introduction, the following results are known.

- Reachability is NL-complete,
- undirected Reachability is L-complete [22],
- planar Reachability is in UL $\cap$ coUL [7].

By the second item one can find out whether an undirected graph G is connected: cycle through all pairs of vertices of the graph and check reachability for each pair. Therefore graph connectivity is in L.

Recall that a set S of vertices is a separating set in G if $G-S$ is not connected. Hence we can check in logspace whether S is a separating set. For constant k, a logspace machine can cycle through all size k subsets of vertices and output the k-separating ones. Hence, in particular, all articulation points, separating pairs, and separating triples of a graph can be computed in logspace. This is what we will use later on.

3 Decomposition of a Graph into Component Trees

In this section we show how to split a graph G into connected, biconnected and triconnected components. Within this context, we always refer to the *undirected version of* G. That is, every edge in G is considered as an undirected edge, and we still call it G.

Since the reachability problem on undirected graphs is in logspace [22], we can check whether s and t are in the same connected component of G, and ignore all other components. Therefore we may w.l.o.g. assume that G is connected.

We further decompose G into biconnected and triconnected components. There is an extensive literature on graph decomposition, see for example [25, 14, 15, 5, 6, 21]. We give definitions that are adapted to a logspace computation of the decompositions.

As we already mentioned, logspace computable functions are closed under composition. Hence we may separately argue that the decomposition into biconnected components is in logspace and the decomposition of biconnected components into triconnected components is in logspace. Then it follows that also the whole process is in logspace.

3.1 The biconnected component tree

We decompose G into biconnected components by splitting G at all its articulation points.

Definition 3.1. Let $G = (V,E)$ be a connected graph. A *biconnected component of* G is a is a maximal biconnected subgraph of G.

Observe that an articulation point occurs in ≥ 2 components. The intersection of the vertices of two biconnected components is either empty or an articulation point.

Lemma 3.2. *The biconnected components of a connected graph G can be computed in logspace.*

Proof. As explained at the end of Section 2, we can find out in logspace whether two vertices u,v of G belong to the same biconnected component. Hence, for a given vertex v, we can compute in logspace all vertices u of G which are in the same biconnected component as v. We use this as a subroutine in our algorithm which outputs all biconnected components of G.

The main loop of the algorithm cycles over all vertices of G. Let v be the current vertex. If v is the first vertex in the loop, we compute all vertices that are in the same biconnected component as v. For all other v's we check whether v is in a biconnected component of some vertex $u < v$. In this case we have already output v in an earlier stage and proceed to the next v in the loop. Otherwise v is in a new component and we compute the vertices which are in the same biconnected component as v. □

We define a graph with the biconnected components as nodes.

Definition 3.3. The *biconnected component tree of* G is the following graph. There is a node for every biconnected component and for every articulation point of G. There is an edge between the node for biconnected component B and the node for an articulation point a, if a belongs to B.

As a convention in this paper, we use the term "node" for the nodes of the component trees and "vertex" for the vertices of the input graph G and its transformations.

In a slight abuse of notation, we also denote the node for component B in the tree by the same name, B, instead of for example v_B, and similar for articulation points. It should always be clear from the context what is meant.

Note that the biconnected component tree is in fact a tree: it is connected because G is connected, and it is acyclic because we deal with articulation points.

We define an order on the nodes of the biconnected component tree of G: The logspace algorithms that compute the articulation points and the biconnected components of G make their respective outputs in a certain order. We use this order for the tree nodes. I.e., as the root of the tree we choose the first articulation point. The order on the children of a node are defined by the order they appear in the construction algorithms. We will use this order when we navigate in the component tree.

A trivial case is when G has no articulation points. Then G is biconnected and the component tree consists of just one node. In this case we can directly proceed to Section 3.2. Hence, for the rest of this section we assume that there are articulation points in G.

By Lemma 3.2 we can compute the nodes of the component tree in logspace. We show that we can also traverse the tree in logspace. It is known that trees can be traversed in logspace, see [8, 20]. We show how to do this in case of component trees.

Lemma 3.4. *The biconnected component tree of a connected graph G can be traversed in logspace.*

Proof. The traversal proceeds as a depth-first search. We show how to navigate locally in the component tree, i.e., for a current node how to compute its *parent*, *first child*, and *next sibling*. We explore the tree starting at the root. Thereby we store the following information on the tape.

- We always store the root node, i.e., one vertex which is an articulation point.
- When the current node is articulation point a_0, we just store it.
- When the current node is a biconnected component B with parent articulation point a_0, then we store a_0 and an arbitrary vertex $v \neq a_0$ from B.

For the last item, note that we cannot afford to store all vertices of B. The vertex v that we store serves as a representative for B. As a choice for v take the first vertex of B that is computed by the construction algorithm of Lemma 3.2. Note that v and a_0 together with the root node identify B uniquely.

The traversal continues by exploring the subtrees at the articulations point in B, different from a_0. Let a_1 be the current articulation point in B. We compute a representative vertex for the first biconnected split component of a_1 different from B. Then we erase a_0 and the representative vertex for B from the tape and recursively traverse the subtrees at a_1.

When we return from the subtrees at a_1, we recompute a_0 and B, the parent of a_1. This is done by computing the path from the root node to B in the component tree. That is, we start at the root node and look for the child component that contains B via reachability queries. Then we iterate the search until we reach B, where we always store the current parent node.

The tree traversal continues with the next sibling of B in the tree. That is, we compute the next articulation point in B after a_1 with respect to the order on the articulation points. Then we delete a_1 from the work tape. If B does not have a next sibling, we return to the parent of B. □

Lemma 3.4 allows us to reduce the the reachability problem for connected graphs to the one for biconnected graphs.

Lemma 3.5. Reachability $\leq_T^{\mathsf{L}}$ *biconnected* Reachability. *The reduction maintains planarity, $K_{3,3}$-freeness, and K_5-freeness.*

Proof. Let B_s be a biconnected component of G that contains s. If s itself is an articulation point, there might be several components that contain s. In this case choose any such component. Let similarly B_t be a component for t. There is a unique simple path P from B_s to B_t in the biconnected component tree. Such a path can be computed in logspace because this is a reachability problem in an undirected graph, the biconnected component tree, which we can traverse by Lemma 3.4. Let $a_1, a_2, \ldots, a_k$ be the articulation points where P passes through, in this order.

The crucial observation now is, that a simple path p from s to t in G has to go through the articulation points $a_1, a_2, \ldots, a_k$ in the same order. Moreover, the part of p between a_i and a_{i+1} stays within the biconnected component defined by a_i and a_{i+1}: suppose that the path would deviate on the way and go through some other articulation point a to a neighboring component. But then p would have to go through a again and hence, p would not be simple. Therefore there is a path from s to t in G if, and only if, there are paths from a_i to a_{i+1} in the biconnected component between a_i and a_{i+1}, for $i = 1, \ldots, k-1$, and similarly, between s and a_1 and between a_k and t. □

As a consequence of the lemma, it suffices to consider biconnected graphs in the following.

3.2 The triconnected component tree

We further decompose the biconnected graph G into its *triconnected components*. In contrast to the decomposition presented in earlier versions of this paper, we found an easier way to argue later on, together with Datta and Nimbhorkar [11]. For completeness, and also because we need the structure of the decomposition later on, we present the new approach from [11] here.

An obvious approach to decompose a biconnected graph G into 3-connected components would be to split G at every separating pair. However, there are some subtleties to take care of. Consider for example a simple cycle. Every pair of vertices in the cycle, except the neighboring ones, constitute a separating pair. Hence, if we would split the cycle at every separating pair, pieces of the cycle would be in many components. To avoid this, we look for such cycle components and do not split them any further. The vertices of separating pair $\{a,b\}$ lie on a cycle if there are only two vertex disjoint paths between them. We decompose a biconnected graph only along separating pairs which are connected by at least three disjoint paths.

Definition 3.6. [11] Let $G = (V,E)$ be a biconnected graph. A separating pair $\{a,b\}$ is called 3-*connected* if there are three vertex-disjoint paths between a and b in G.

The *triconnected components of* G are the split graphs we obtain from G by splitting G successively along all 3-connected separating pairs, in any order. If a separating pair $\{a,b\}$ is connected by an edge in G, then we also define a 3-*bond* for $\{a,b\}$ as a triconnected component, i.e., a multigraph with two vertices $\{a,b\}$ and three edges between them.

In summary, we get three types of triconnected components of a biconnected graph: 3-connected components, cycle components, and 3-bonds. The task of the 3-bonds is only to represent edges of the graph which are replaced by virtual edges in the other components. That way, we do not need access to the input graph G anymore for further computation, all the information about G is available in the components. Note that the cycle components are not 3-connected. A special case is when G has separating pairs, but none of them is 3-connected. Then G is a simple cycle and constitutes one cycle component.

Definition 3.6 leads to the same triconnected components as defined by Hopcroft and Tarjan [15], but by a different construction: they first completely decompose the graph along *all* separating pairs and then merge triangles to larger cycles. They also show that the decomposition is unique, i.e., independent of the order of the separating pairs in the definition. This is also shown in [11].

Hopcroft and Tarjan [15] presented a linear-time algorithm to compute such a decomposition. Miller and Ramachandran [21] present a linear time algorithm for computing triconnected components which also has a parallel implementation on a CRCW-PRAM with $O(\log^2 n)$ parallel time and using a linear number of processors. By the next lemma, the decomposition can also be computed in logspace.

Lemma 3.7. [11] *The* 3*-connected separating pairs and the triconnected components of a biconnected graph G can be computed in logspace.*

Proof. We already explained that we can compute all separating pairs of G in logspace. Among those, we identify the 3-connected ones as follows. A separating pair $\{a,b\}$ in G is *not* 3-connected if there are exactly two split components of $\{a,b\}$ (without attaching virtual edges) and both are not biconnected. To see this, note that a split component C which is not biconnected has an articulation point c. All paths from a to b in C must go through c. Hence there are no two vertex disjoint paths from a to b.

To check the above conditions we have to find articulation points in the split components of $\{a,b\}$. This can be done in logspace with queries to reachability.

It remains to compute the vertices of a 3-connected component. Two vertices $u,v \in V$ belong to the same 3-connected component or cycle component, if no 3-connected separating pair separates u from v. This property can again be checked by solving several reachability problems. □

In the same way as we defined the biconnected component tree of a connected graph, we define the *triconnected component tree* of a biconnected graph.

Definition 3.8. The *triconnected component tree $\mathcal{T}$ of G* is the following graph. There is a node for each triconnected component and for each 3-connected separating pair of G. There is an edge in $\mathcal{T}$ between the node for triconnected component C and the node for a separating pair $\{a,b\}$, if a,b belong to C.

Note that graph $\mathcal{T}$ is connected, because G is biconnected, and acyclic. Hence $\mathcal{T}$ is a tree. For an example see Figure 1. We consider an arbitrary separating pair as the root node of $\mathcal{T}$. If G has no separating pair, i.e. G is 3-connected, then $\mathcal{T}$ consists of a single node.

By Lemma 3.7 we can compute the nodes of the component tree in logspace. We show that we can also traverse the tree in logspace.

Lemma 3.9. *The triconnected component tree of a biconnected graph G can be computed and traversed in logspace.*

Proof. The traversal of the tree works analogously as in Lemma 3.4 for the biconnected component tree. Instead of articulation points we have separating pairs now. Hence we store two vertices instead of one. In case of a 3-connected component or a cycle, we again store a representative vertex from the component, for example the first vertex from the component computed by the construction algorithm. The local navigation now can be done in the same way as for the biconnected component tree. □

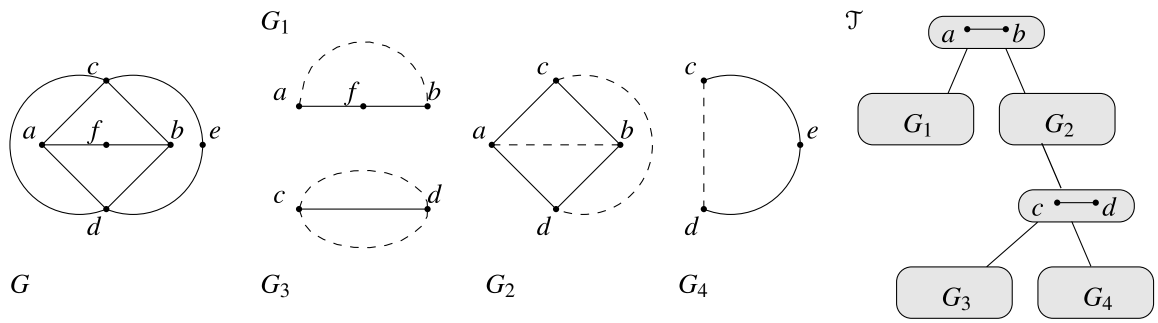


Figure 1: ([10]) The decomposition of a biconnected planar graph G. Its triconnected components are $G_1,\dots,G_4$ and the corresponding decomposition into the triconnected tree $\mathcal{T}$ of G. The separating pairs are $\{a,b\}$ and $\{c,d\}$. Since the separating pair $\{c,d\}$ is connected by an edge in G, we also get $\{c,d\}$ as 3-bond G_3. The virtual edges corresponding to the separating pairs are drawn with dashed lines.

We define the size of a triconnected component tree in the obvious way. For our logspace algorithms it will be important to detect *large children* in a tree.

Definition 3.10. Let $\mathcal{T}$ be a triconnected component tree of graph G. The *size of an individual component node* of $\mathcal{T}$ is the number of vertices of G in the component. The *size of* $\mathcal{T}$, denoted by $|\mathcal{T}|$, is the sum of the sizes of its component nodes.

Let $\mathcal{T}_C$ be a component tree rooted at some component C and let $\mathcal{T}_{C'}$ be a subtree of $\mathcal{T}$ rooted at a child C' of C. We call C' a *large child of* C, if $|\mathcal{T}_{C'}| > |\mathcal{T}_C|/2$.

Clearly a node can have at most one large child.

3.3 Partitioning the reachability problem

Let $G=(V,E)$ be a biconnected graph and $s,t \in V$. Let $\mathcal{T}_G$ be the triconnected component tree of G. Let S and T be 3-connected components that contain s and t, respectively. If s or t belong to a separating pair, they occur in several components. In this case choose an arbitrary such component. Consider the simple path from S to T in $\mathcal{T}_G$, say $S=C_1,\ C_2,\dots,\ C_\ell=T$, see Figure 2.

By the definition of the triconnected component tree, the nodes are alternating 3-connected component nodes and separating pair nodes. Let $C_i=\{a_i,b_i\}$ be a separating pair node, and C_{i-1} and C_{i+1} component nodes. Observe that a simple path p from s to t in G has to visit at least one vertex of each of these separating pairs. Once p has reached C_i, say via a_i, it will not go back to C_{i-1}: the only way back to C_{i-1} would be via b_i. Then p already visited both vertices of C_i and still should proceed via C_i. But this is not possible because p is a simple path. Similarly, if p has reached C_{i+1}, it will not go back to C_i. Path p might pass through the components indicated below the C_i's in Figure 2. But the components $C_1,\ C_2,\dots,\ C_\ell$ are visited in this order.

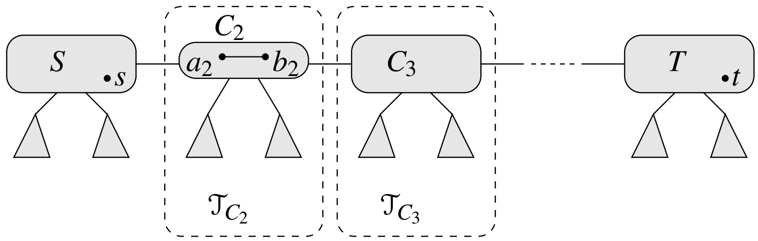


Figure 2: The triconnected component tree $\mathcal{T}_G$ partitioned according to a path from S to T, indicated by the dashed boxes. The solid boxes indicate component nodes and the triangles indicate subtrees. Components C_i are alternating separating pairs, $C_i = \{a_i, b_i\}$, or 3-connected.

For a node C_i, we define the tree $\mathcal{T}_{C_i}$ and its underlying graphs G_i as

$$\begin{aligned} \mathcal{T}_{C_i} &= \text{the subtree of } \mathcal{T}_G \text{ rooted at } C_i\text{, where the branches to } C_{i-1} \text{ and } C_{i+1} \text{ are cut off,} \\ G_i &= \text{the graph corresponding to } \mathcal{T}_{C_i}. \end{aligned}$$

We have just argued that the reachability problem in G can be partitioned into reachability problems in the G_i's.

Lemma 3.11. *Any simple path p from s to t in G can be written as a concatenation of paths, $p = p_1 \cdot p_2 \cdots p_\ell$, such that*

- *p_1 goes from s to a_2 or b_2 in G_1,*
- *if C_i is a component node, p_i is a path from a_{i-1} or b_{i-1} to a_{i+1} or b_{i+1} in G_i,*
- *if C_i is a separating pair node, p_i is a path from a_i to b_i in G_i, or vice versa, or a trivial path $p_i = (a_i)$ or $p_i = (b_i)$,*
- *p_ℓ is a path from $a_{\ell-1}$ or $b_{\ell-1}$ to t in G_ℓ.*

In the reachability problem for G_i, we search for a path from a_i or b_i to a_{i+1} or b_{i+1}, if C_i is a component node, and from a_i to b_i or vice versa if C_i is a separating pair node. Recall that each separating pair a, b is connected by a virtual edge. Hence we might have a virtual edge (a, b) in C_i on our path.

- If (a, b) is also a directed edge in G then we are fine. This edge is indicated by a 3-bond node as a child in the tree.
- Otherwise we have to check whether there is a path from a to b in G_i by traversing a child of C_i in $\mathcal{T}_{C_i}$.

  Note that in the child component the same situation may occur again. Suppose we reach vertex c of separating pair $\{c, d\}$ in a child component. If the path now goes further down into a split component of $\{c, d\}$, then the only way back to b goes via d. Hence, for the separating pairs $\{c, d\}$ inside the subtrees $\mathcal{T}_{C_i}$, we only want to find out whether there is a path from c to d. We do *not* need to consider four connectivity problems between two separating pairs as above between the root components of the $\mathcal{T}_{C_i}$'s.

If a component is planar, we can test reachability in $\mathsf{UL} \cap \mathsf{coUL}$ [7]. However, this is not known for the nonplanar components. Allender and Mahajan [3] showed that planarity can be tested in logspace. Hence we can find out which components are planar and which are not. In the next two sections we show for $K_{3,3}$-free and K_5-free graphs G how these nonplanar components may look like and how to replace them by planar components such that the original reachability problem stays the same. The reason for the partitioning of the reachability problem presented above is that we do a different replacement if the nonplanar component is the root C_i of $\mathcal{T}_{C_i}$ than if it is inside $\mathcal{T}_{C_i}$. The difference is due to the point described in the second item above.

4 Reachability in $K_{3,3}$-free Graphs

We give a logspace reduction from the reachability problem for biconnected $K_{3,3}$-free graphs to the reachability problem for planar graphs. The latter problem is known to be in $\mathsf{UL} \cap \mathsf{coUL}$ [7].

Theorem 4.1. *biconnected $K_{3,3}$-free* Reachability $\leq_m^{\mathsf{L}}$ *planar* Reachability.

We prove Theorem 4.1 in Section 4.1 and 4.2. Let G be the given biconnected $K_{3,3}$-free graph. Consider the decomposition of the underlying undirected version of G into triconnected components as described in Section 3. Asano [4] (see also Hall [13]) showed that every nonplanar component is precisely the K_5. Figure 3 shows an example.

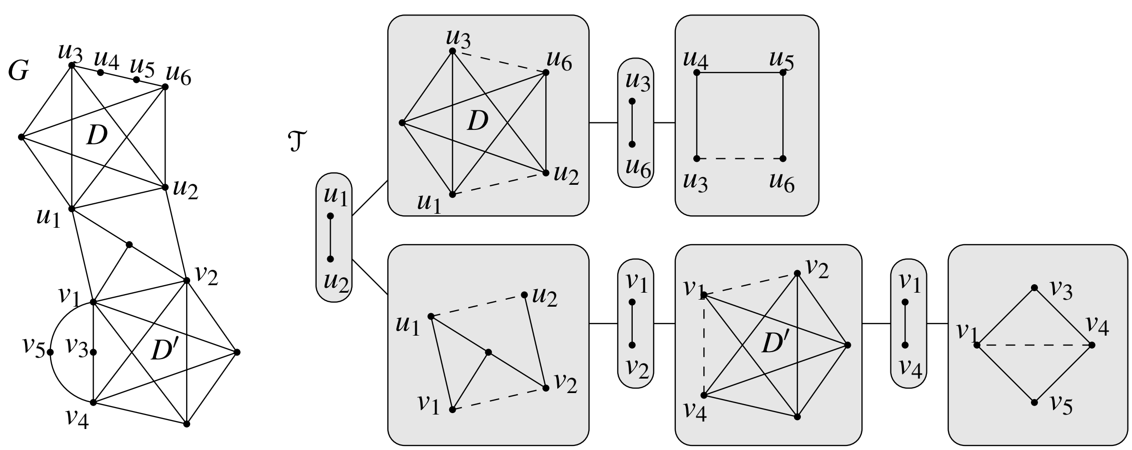


Figure 3: The $K_{3,3}$-free graph G contains two K_5-components K_1 and K_2 which appear as nodes in the triconnected component tree $\mathcal{T}$. To keep the figure simple, we do not show the 3-bonds and also did not further decompose the planar components.

Theorem 4.2. [4] *Each triconnected component of a $K_{3,3}$-free biconnected graph is either planar or exactly the graph K_5.*

The key step in the reduction is to replace the K_5-components by planar components such that the reachability properties are maintained with respect to the directed graph G.

4.1 Transforming a K_5-component into a planar component.

Recall the partitioning of the reachability problem shown in Section 3.3. Let $S = C_1, C_2, \ldots, C_\ell = T$ be the simple path from S to T in the triconnected component tree $\mathcal{T}$ of G. Let $\mathcal{T}_{C_i}$ be the subtree rooted at C_i and G_i be the subgraph of G corresponding to $\mathcal{T}_{C_i}$.

We start with the root C_i and traverse the tree in depth first manner. When we reach a K_5-component node K, then we replace it locally by a planar component such that the reachability properties do not change.

As explained in Section 3.3 and Lemma 3.11, we have to solve a reachability problem in every component. In case that K is the root of $\mathcal{T}_{C_i}$, then there are actually four reachability problems. We postpone this case to Section 4.2.

So let K be an inner K_5-component node in $\mathcal{T}_{C_i}$ with vertices $w_1, \ldots, w_5$. There is a parent separating pair, say $\{w_1, w_2\}$, and we search for a path from w_1 to w_2. The planar graph K' that replaces K is defined as shown in Figure 4. Node K might have a large child within $\mathcal{T}_{C_i}$. Figure 4 shows K' for the case when the large child is at (w_3, w_4), if there is one.

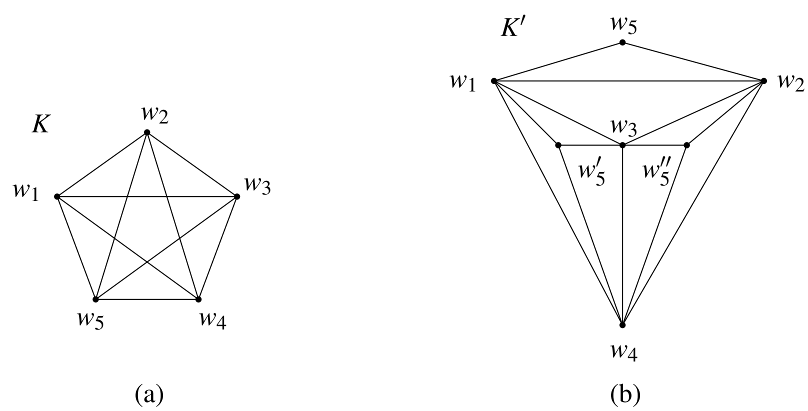


Figure 4: (a) A K_5-component node K.
(b) The planar component node K' constructed from K for the case that we search for a path from w_1 to w_2, and, if K has a large child, it is at (w_3, w_4). The two vertices w_5' and w_5'' are copies of w_5. For example, an edge (w_1, w_5) in K occurs twice in K', as (w_1, w_5) and (w_1, w_5'). In case (w_1, w_5) is a virtual edge in K, then both copies in K' are considered as virtual edges.
The edges of K and K' are drawn undirected to not overload the picture. The edges that come from graph G have the same direction as in G.

The following Lemma summarizes the important properties of K' that we will use later on.

Lemma 4.3. *Let K be a K_5-component node with vertices $w_1,\ldots,w_5$, where we search for a path from w_1 to w_2, and, if there is a large child, it is at (w_3,w_4). Let K' be the component constructed from K shown in Figure 4. Component K' has the following properties:*

- *K' is planar.*
- *Every path from w_1 to w_2 in K exists as well in K', possibly going through one of the copies w_5' or w_5'' instead of w_5.*
- *K' contains the edge (w_3,w_4) only once.*
- *Vertices w_1 and w_2 are on the outer face of K' in the embedding shown in Figure 4.*

The second item of the lemma implies the correctness of the construction: There is a path from w_1 to w_2 in K or vice versa, iff there is such a path in K'. The last item will be important when we reverse the decomposition process. Then the planar components will be sticked together at the separating pairs. For the resulting graph to be planar we need w_1 and w_2 on the outer face.

The price we pay for getting K' planar is that one vertex of K occurs three times as copies of the original vertex, this is w_5 in Figure 4. As a consequence, some of the original edges occur now twice in K', like (w_1,w_5) and (w_1,w_5'). Now we might run into a problem. Namely, at the virtual edges, our algorithm recursively explores the subtrees at this edge. If we have a copy of such an edge, we will explore the same subtree again when we come to the copy. This is fine as long as the size of the subtree is small i.e. a fraction of N, where N is the size of the subtree rooted at K in $\mathcal{T}_{C_i}$. However, there might be a large child, i.e. a child of size $> N/2$, see Definition 3.10 on page 9. In this case, we should *not* make copies of the edge in order to stay in logspace. The definition of K' given in Figure 4 refers to the case that we search a path from w_1 to w_2 in K and we have a large child at (w_3,w_4). The same construction works if there is no large child, and it can be easily adapted to the case that the large child is at another pair, e.g. at (w_2,w_4).

4.2 Replacing the K_5-components

Recall that G_i is the subgraph of G corresponding to component tree $\mathcal{T}_{C_i}$ according to the partitioning given in Section 3.3. Our goal now is to replace the K_5-components in $\mathcal{T}_{C_i}$ by the above planar gadget and then to reassemble the components to a planar graph G_i'. This is done by identifying the copies of the vertices of the separating pairs. By the last item of Lemma 4.3, the vertices of the separating pairs can be put at the outer face of a K'-component. Therefore the resulting graph is planar.

However, this simple replacement we do only inside the tree, for K_5-components K different from the root. Suppose path p from s to t reaches K at w_1 of separating pair $\{w_1,w_2\}$. Then we only want to know whether there is a path from w_1 to w_2, because this is the only way to get back to the root node. Here it is fine to stick components together at their copies of a separating pair $\{w_1,w_2\}$, because then we have the same paths available as in G. However, when the root of the tree, C_i, is itself a K_5-component, then the replacement is done differently.

Consider the case that K is the root of $\mathcal{T}_{C_i}$, i.e., $K = C_i$. Recall that C_i has the separating pairs $\{a_{i-1},b_{i-1}\}$ and $\{a_{i+1},b_{i+1}\}$ as neighbors on path P from S to T. So now we have to consider four reachability questions from a_{i-1} or b_{i-1} to a_{i+1} or b_{i+1}. For each of the four possibilities we create one copy of G_i, say $G_{i,1},\dots,G_{i,4}$. The corresponding trees $\mathcal{T}_{C_i,1},\dots,\mathcal{T}_{C_i,4}$ have copies $K_1,\dots,K_4$ of K as their root. For example, at K_1 we ask for a path from a_i to a_{i+1}. Hence we identify w_1 of K_1 with a_i and w_2 with a_{i+1}. Then we replace each K_i by a planar K_i' from Figure 4, adapted to the corresponding reachability problem. For the internal K_5-nodes, we do the simple replacement described above. This yields subgraphs $G'_{i,1},\dots,G'_{i,4}$. Graph G'_i is defined by identifying the copies of vertices $a_{i-1},b_{i-1},a_{i+1},b_{i+1}$ in $G'_{i,1},\dots,G'_{i,4}$, respectively. The drawing in Figure 5 shows that G'_i is planar.

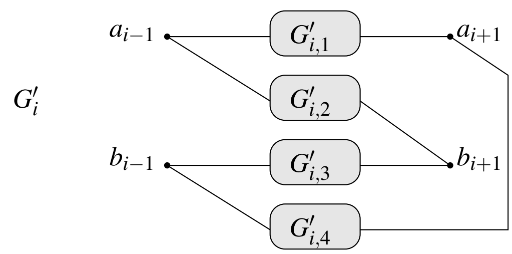


Figure 5: A schematic illustration of the construction when K is the root of $\mathcal{T}_{C_i}$. There are four reachability problems from the vertices of separating pairs $\{a_{i-1},b_{i-1}\}$ to $\{a_{i+1},b_{i+1}\}$. For each of the four possibilities we have one copy of G_i, say $G_{i,1},\dots,G_{i,4}$. The corresponding trees $\mathcal{T}_{C_i,1},\dots,\mathcal{T}_{C_i,4}$ have copies $K_1,\dots,K_4$ of K as their root. Then we replace each K_i by a planar K_i' from Figure 4, adapted to the corresponding reachability problem. This yields subgraphs $G'_{i,1},\dots,G'_{i,4}$. We join them at $a_{i-1},b_{i-1},a_{i+1},b_{i+1}$ as indicated. This defines G'_i.

Lemma 4.4. *Let G'_i be the subgraph that results from G_i after the replacement of the K_5-components. G'_i has the following properties.*

(i) G'_i is planar.

(ii) if C_i is a component node: there are paths from a_{i-1} or b_{i-1} to a_{i+1} or b_{i+1} in G_i if and only if there are such paths in G'_i.

(iii) if C_i is a separating pair node: there is a path from a_i to b_i in G_i, or vice versa, if and only if there are such paths in G'_i.

(iv) The size of G'_i is polynomial in the size of G_i.

Proof. By the discussion preceding the lemma it remains to show *(i)* and *(iii)*.

Ad *(i)*. G'_i is constructed by merging planar components at their common separating pairs. Note that in each planar component the separating pairs are connected by a virtual edge. Therefore the two vertices of a separating pair touch the same face. Moreover, by Lemma 4.3, there is a planar embedding such that the root separating pair of each component is at the outer face. Therefore we can put all components with the same root separating pair inside a face which touches the root separating pair in the parent component. Hence the merging process yields again a planar graph.

Ad *(iii)*. For G_i of size N, let $\mathcal{S}(N)$ be the size of G'_i. If a K_5-component K in G_i is replaced by a planar component K', some edges of K have several copies in K'. If such an edge corresponds to a separating pair, we also copy the subtree of that edge. Let k be the number of additional copies of edges in K' from K. We have the following recurrence for $\mathcal{S}(N)$,

$$\mathcal{S}(N) \leq k\,\mathcal{S}(N/2) + O(N)\,.$$

Recall that we do not make copies of a large child. The subgraphs we make copies of are of size $\leq N/2$. This leads to a polynomial bound on $S(N)$ for constant k: $S(N) = O(N^{\log k})$.

We give a bound on k: each of the four edges to w_5 in K have one extra copy in K', going to w_5, w'_5, or w''_5. Moreover, if K is at the root of the current subtree, we use the construction shown in Figure 5. Then we create four copies of the subtrees. Hence $k \leq 4 \cdot 4 = 16$. □

As a final step, we concatenate graphs G'_i at the separating pairs between them. The resulting graph G' is the output of our reduction. Graph G' is clearly planar and has polynomial size by Lemma 4.4 *(iii)*. It is also biconnected. We argue that G' can be constructed in logspace.

Lemma 4.5. *G' is planar and can be computed in logspace. There is a path from s to t in G if and only if there is such a path in G'.*

Proof. By Lemma 3.7 and 3.9 we can compute the triconnected component tree in logspace. We can also figure out which are the K_5-components because they have constant size. We can compute path P from S to T in the tree in logspace. Recall that the tree is undirected.

Then we replace the K_5-components by the planar gadget. Thereby we distinguish whether a K_5-component lies on path P or not and do the replacement accordingly as described above. From the resulting tree we construct G' by identifying the copies of separating pairs.

We mention some technical details of the algorithm.

- When a subtree is copied due to the replacement of a K_5-component K by K', we give new names to vertices in the copies of the subtree.

- A separating pair in component K can have up to 8 copies in K' (in case K is one of the root nodes). When the depth first traversal goes into recursion at a separating pair in K', we have to store at which copy of a separating pair we went into recursion. Because there are ≤ 8 copies, 3 bits suffice. We need such bits at each level of the recursion. In the depth traversal, whenever we reach a copied component for the first time, we relabel its vertices, keeping a counter on the work tape which starts by $n+1$ (assuming, that G has vertices with labels $1,\ldots,n$). Then we recursively traverse the children of the component.

At each stage in the tree, say at a node C, the sizes of the subtrees rooted at the children of C are $\leq 1/2$ the size of $\mathcal{T}_C$. Hence, there are $O(\log n)$ levels of recursion and the algorithm runs in logspace. □

This finishes the proof of Theorem 4.1. Together with Lemma 3.5 and the result of [7] we get:

Corollary 4.6. *$K_{3,3}$-free* Reachability *is in* $\mathsf{UL} \cap \mathsf{coUL}$.

4.3 Distance and longest paths in $K_{3,3}$-free graphs

For the distance problem and the longest path problem it suffices again to consider biconnected graphs, because we can pass only once through every articulation point on a simple path from s to t. Hence we can consider longest paths or distances in the biconnected components, and then sum up these lengths appropriately.

For a biconnected $K_{3,3}$-free graph G we use the same transformation to the planar graph G' as above in Lemma 4.5. The crucial point to observe is that simple paths in a K_5-component K of G have the same length as the corresponding paths in the planar component K' in G'. Hence, distances in G are the same as in G'. Moreover, if G is acyclic, then also G' is acyclic.

Lemma 4.7. *1. $K_{3,3}$-free* Distance $\leq_T^{\mathsf{L}}$ *planar* Distance.

2. $K_{3,3}$-free DAG Long-Path $\leq_T^{\mathsf{L}}$ *planar DAG* Long-Path.

Thierauf and Wagner [24] proved that computing the distance in planar directed graphs is in $\mathsf{UL} \cap \mathsf{coUL}$. Limaye, Mahajan and Nimbhorkar [19] proved that computing a longest path in planar DAGs is in $\mathsf{UL} \cap \mathsf{coUL}$.

Corollary 4.8. *Distances in $K_{3,3}$-free graphs and longest paths in $K_{3,3}$-free DAGs can be computed in* $\mathsf{UL} \cap \mathsf{coUL}$.

5 Reachability in K_5-free graphs

We give a logspace reduction from the reachability problem for directed K_5-free graphs to the reachability problem for directed planar graphs.

Theorem 5.1. *biconnected K_5-free* Reachability $\leq_m^{\mathsf{L}}$ *planar* Reachability.

We consider again the decomposition of a biconnected graph into 3-connected components. The crucial theorem for K_5-free graphs is due to Wagner [26]. Khuller [17] gives a formulation of Wagners theorem in terms of a clique-sum operation where graphs are joined at a common clique.

Definition 5.2. Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two undirected graphs such that the induced subgraphs of G_1 and G_2 on $V_1 \cap V_2$ both are cliques.

A graph $G = (V_1 \cup V_2, E)$ is a *clique-sum of* G_1 *and* G_2 if E agrees with E_1 on $V_1 - V_2$ and with E_2 on $V_2 - V_1$, E is arbitrary on $V_1 \cap V_2$, and there are no other edges in E. If $|V_1 \cap V_2| \leq k$, we also say that G is a *k-clique-sum*.

For a class $\mathcal{G}$ of graphs, $\langle \mathcal{G} \rangle_k$ is the closure of $\mathcal{G}$ under the k-clique-sum operation.

The Möbius ladder M_8, is shown in Figure 6. It is a 3-connected graph on 8 vertices which is nonplanar, because it contains a $K_{3,3}$. Wagner showed that the closure under 3-clique-sum of planar graphs and the M_8 is precisely the class of K_5-free graphs.

Theorem 5.3. [26] *Let $\mathcal{C}$ be the class of all planar graphs together with the Möbius ladder M_8. Then $\langle \mathcal{C} \rangle_3$ is the class of all graphs with no K_5-minor.*

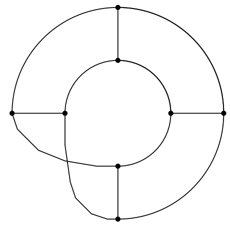


Figure 6: The Möbius ladder M_8.

We make two easy observations with respect to the above clique-sum operation.

- 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 M_8 is nonplanar and 3-connected, but not 4-connected. Furthermore, the M_8 cannot be part of a 3-clique-sum operation where all the tree vertices are chosen from the M_8, because the M_8 does not contain a triangle as induced subgraph.

By Theorem 5.3 and the two observations we get a characterization of all nonplanar 3-connected components of a K_5-free graph.

Corollary 5.4. (cf. [17]) *A 3-connected nonplanar component of a K_5-free biconnected graph is either the M_8 or its 4-connected components are all planar.*

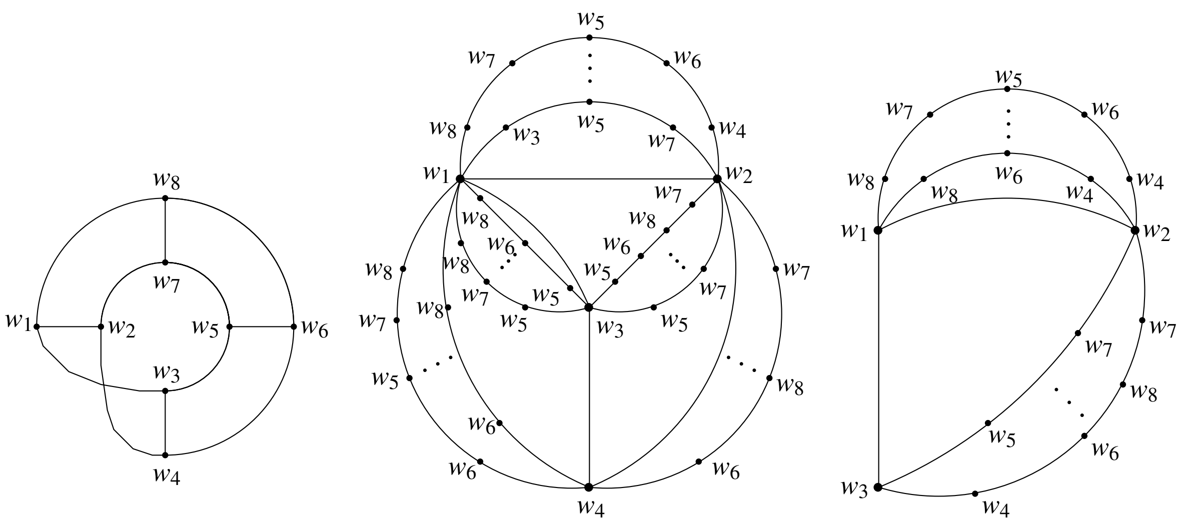
In the next two sections we construct a planar gadget to replace the M_8-components and show how to decompose the other nonplanar components into 4-connected planar components in logspace.

5.1 Transforming a M_8-component into a planar component

Recall again the partitioning of the reachability problem shown in Section 3.3. Let $S = C_1, C_2, \ldots, C_\ell = T$ be the simple path from S to T in the triconnected component tree $\mathcal{T}$ of G. Let $\mathcal{T}_{C_i}$ be the subtree rooted at C_i and G_i be the subgraph of G corresponding to $\mathcal{T}_{C_i}$. The C_i's are alternating separating pair nodes, $C_i = \{a_i, b_i\}$ in this case, and component nodes.

We traverse $\mathcal{T}_{C_i}$ in depth first manner. When we reach an M_8-component node, then we replace it locally by a planar component such that the reachability properties do not change.

Let M be an M_8-component node in $\mathcal{T}_{C_i}$ with vertices $w_1, w_2, \ldots, w_8$. If M is not the root of $\mathcal{T}_{C_i}$, then there is a parent separating pair, say $\{w_1, w_2\}$, and we search for a path from w_1 to w_2. The planar graph M' that replaces M is defined in Figure 7. Node M might have a large child within $\mathcal{T}_{C_i}$. The edge of a large child should not be copied. Figure 7 (b) and (c) show two cases where (w_3, w_4) and (w_1, w_3) correspond to a large child of M, respectively. If M is the root of $\mathcal{T}_{C_i}$, then we use the same construction as in Figure 5.



(a) (b) (c)

Figure 7: (a) Assume that we search for a path from w_1 to w_2 in the M_8-component M. Edges are drawn undirected for simplicity.
(b) The planar component M' is shown schematically for the case that (w_3, w_4) corresponds to a large child of M. For simplicity, all the copies of a vertex in M' have the same label in the picture. For every path from w_1 to w_2 that does *not* contain edge (w_3, w_4), M' contains a copy of the path, i.e. a copy of all vertices and edges on this path. This is indicated by the paths above the edge (w_1, w_2) in the picture. The remaining paths go along (w_3, w_4) or (w_4, w_3) in M. This edge should occur only once in M'. Therefore, these paths in M are subdivided into paths from w_1 to w_3 or to w_4, and from these vertices to w_2. This is indicated in the part below the edge (w_1, w_2) in the picture.
(c) M' in the case that (w_1, w_3) is the large child. The construction is essentially the same as in (b).

Lemma 5.5. *Let M be a M_8-component node with vertices $w_1, w_2, \ldots, w_8$, where we search for a path from w_1 to w_2, and, if there is a large child, it is at (w_3, w_4). Let M' be the component constructed from M shown in Figure 7 (b). Component M' has the following properties:*

- *M' is planar.*
- *Every path from w_1 to w_2 in M exists as well in M', possible going through the copies of the vertices instead.*
- *M' contains the edge (w_3, w_4) only once.*
- *Vertices w_1 and w_2 are both on the outer-face of M' in the embedding shown in Figure 7.*

The second item of the lemma implies the correctness of the construction: There is a path from w_1 to w_2 in M or vice versa, iff there is such a path in M'. We put things together. Analogously to Lemma 4.4 and 4.5, we get:

Lemma 5.6. *Let G_i' be the subgraph that results from G_i after the replacement of the M_8-components. G_i' has the following properties:*

(i) G_i' has no M_8 as a 3-connected component.

(ii) if C_i is a component node: there are paths from a_{i-1} or b_{i-1} to a_{i+1} or b_{i+1} in G_i if and only if there are such paths in G_i'.

(iii) if C_i is a separating pair node: there is a path from a_i to b_i in G_i, or vice versa, if and only if there are such paths in G_i'.

(iv) The size of G_i' is polynomial in the size of G_i.

(v) G_i' can be constructed in logspace.

The proof is the same as for the corresponding lemmas in Section 4. The constant k in the size bound for G_i' is a bit larger here, because we have more copies of vertices. The construction algorithm works also in the same way, we just have to replace a M_8-component here instead of a K_5-component.

5.2 The 4-connected component tree

We show how to further decompose the 3-connected components which are nonplanar and not the M_8. We start by identifying the separating triples that define a 4-connected component.

For a given separating triple τ_0, we would like to define the *maximal separating triples w.r.t.* τ_0 as the separating triples τ that are not separated from τ_0 by any other separating triple. These separating triples define one 4-connected component. A technical difficulty here is that the split components of two separating triples might overlap each other. If we would simply split a 3-connected component along all its separating triples, we might decompose some parts more than once. To avoid this, we first define the notion of a *candidate maximal separating triples*. The actual maximal separating triples to proceed with are then selected from the candidates.

Definition 5.7. Let C be a 3-connected component and τ_0 be a separating triple in C. Let C' be a split component of τ_0 in C. A separating triple $\tau \neq \tau_0$ is a *candidate maximal separating triple in C' w.r.t.* τ_0, if no separating triple $\tau' \notin \{\tau, \tau_0\}$ separates τ from τ_0 in C'.

Two candidate maximal separating triples τ_1, τ_2 w.r.t. τ_0 are called *crossing*, if there is a vertex $v \notin \tau_0 \cup \tau_1 \cup \tau_2$ in C' which is separated from τ_0 by τ_1 and by τ_2.

Figure 8 shows an example where crossing separating triples occur. The split component of separating triple $\tau_0 = \{a_0, b_0, c_0\}$ in Figure 8 (a) has separating triples $\tau_1 = \{a, b_0, c_0\}$ and $\tau_2 = \{a_0, b_0, c\}$ which overlap each other and τ_0. Figure 8 (b) and (c) show the split components we get when we split further along τ_1 and τ_2. The split components overlap each other in vertex b. Therefore τ_1 and τ_2 are crossing.

The next lemma shows that there are essentially two ways how crossing candidate maximal separating triples can occur. Figure 8 and 11 present the two possibilities.

Lemma 5.8. *Let τ_1 and τ_2 be two crossing candidate maximal separating triples w.r.t. τ_0 in a triconnected component C. Then $\tau_0 \subseteq \tau_1 \cup \tau_2$ and $|\tau_1 \cap \tau_2| \leq 1$.*

Furthermore, τ_1 and τ_2 each have exactly one split component besides the one with τ_0.

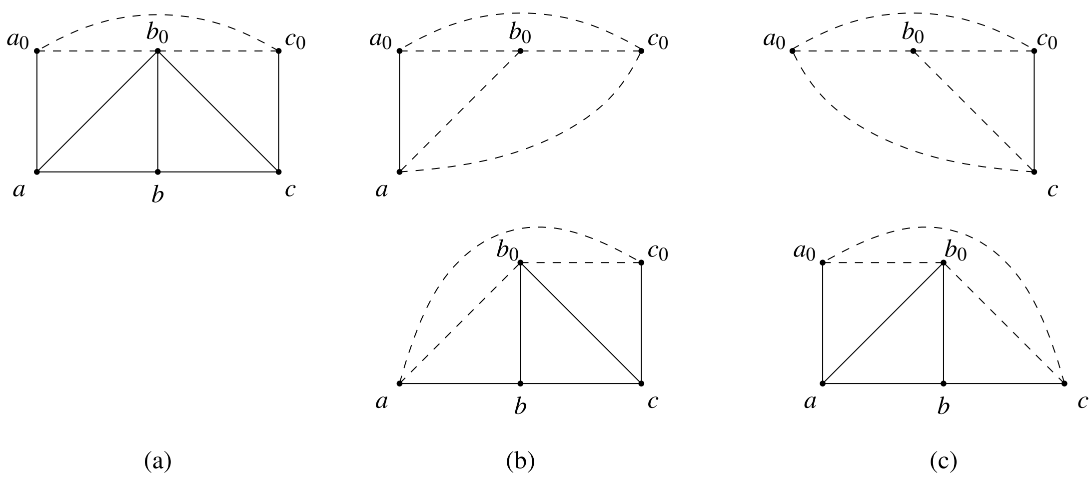


Figure 8: (a) A split component of separating triple $\tau_0 = \{a_0, b_0, c_0\}$. Two crossing candidate maximal separating triples are $\tau_1 = \{a, b_0, c_0\}$ and $\tau_2 = \{a_0, b_0, c\}$. We have $\tau_1 \cap \tau_2 = \{b_0\}$.
(b) The upper graph is the 4-connected split component of τ_0 and τ_1. The lower graph is the split component of τ_1 which has to be decomposed further.
(c) Same as in (b) but for separating triple τ_2 instead of τ_1.

Proof. Let $\tau_1 = \{a_1, b_1, c_1\}$ and $\tau_2 = \{a_2, b_2, c_2\}$. Let v be a vertex that is separated from τ_0 by τ_1 and by τ_2.

Assume first that $\tau_0 \not\subseteq \tau_1 \cup \tau_2$. Let $a_0 \in \tau_0 - (\tau_1 \cup \tau_2)$. Because component C is 3-connected, there are ≥ 3 vertex-disjoint paths from v to a_0 in C. Every such path must go through a vertex of τ_1 and of τ_2, because both triples separate v from τ_0.

Let p_1, p_2, p_3 be three vertex-disjoint paths from v to a_0. From each path p_i pick the vertex of $\tau_1 \cup \tau_2$ that is closest to a_0. Let τ be these three vertices. Let $\rho = (\tau_1 \cup \tau_2) - \tau$ be the remaining vertices. Since τ_1 and τ_2 might intersect each other we have $1 \leq |\rho| \leq 3$. Because p_1, p_2, p_3 are vertex-disjoint, there are only two cases how the paths go through τ_1 and τ_2: for $i = 1, 2, 3$,

1. either p_i passes through a vertex in $\tau_1 \cap \tau_2$, which belongs to τ then,
2. or p_i passes through a vertex in $\tau_1 - \tau_2$ and a vertex in $\tau_2 - \tau_1$. The vertex which p_i reaches first is in ρ, the other in τ.

In both cases, p_i does not pass through any other vertices of $\tau_1 \cup \tau_2$. Figure 9 illustrates the situation.

We claim that *every* path p from a vertex in ρ to a_0 passes through a vertex of τ: suppose that p manages to avoid τ. Let u be the vertex in ρ on p that is closest to a_0. One of the paths p_1, p_2, p_3 passes through u, say p_1. Let p'_1 be the initial part of p_1 from v to u. By definition of ρ, path p'_1 does not go through τ. We construct a path p' by concatenating p'_1 with the rear part of p from u to a_0. Then p' is a path from v to a_0 that passes through only one vertex of $\tau_1 \cup \tau_2$, namely u. Recall that u is in just one

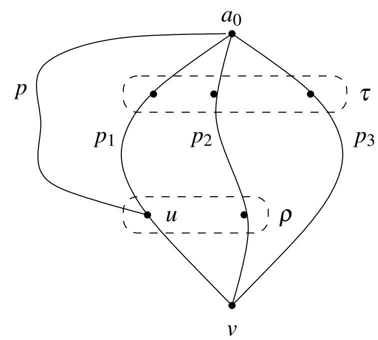


Figure 9: The vertex-disjoint paths p_1, p_2, p_3 from v to $a_0 \in \tau_0 - (\tau_1 \cup \tau_2)$. The vertices of $\tau_1 \cup \tau_2$ are partitioned into τ and ρ. In the example here, τ_1 and τ_2 have one vertex in common, the one on p_3, which is in τ then.
Suppose there is a path p from $u \in \rho$ that does not pass through a vertex of τ. Then the initial part of p_1 from v to u connected with p yields a path p' from v to $a_0 \in \tau_0$. Then either τ_1 or τ_2 does not separate v from τ_0, a contradiction. Hence there is no such path p.

of τ_1 and τ_2. Hence, p' passes through only one of τ_1 and τ_2, the one u belongs to. But then, because of p', the other triple would not separate v from τ_0, a contradiction.

We conclude that τ separates ρ from τ_0. Therefore τ is a separating triple that separates τ_1 from τ_0, if $\rho \cap \tau_1 \neq \emptyset$, or τ_2 from τ_0, if $\rho \cap \tau_2 \neq \emptyset$. But then τ_1 or τ_2 would not be maximal which is a contradiction. Therefore we must have $\tau_0 \subseteq \tau_1 \cup \tau_2$.

We show that $|\tau_1 \cap \tau_2| \leq 1$. Assume that τ_1 and τ_2 share two vertices, say $\tau_1 = \{a_1, b, c\}$ and $\tau_2 = \{a_2, b, c\}$. Because we already showed $\tau_0 \subseteq \tau_1 \cup \tau_2$, and we also have $\tau_1, \tau_2 \neq \tau_0$, exactly one of b and c is in τ_0, say b. I.e., we have $\tau_0 = \{a_1, a_2, b\}$.

We show that $\{b, c\}$ is a separating pair in this case. Namely, $\{b, c\}$ separates v from a_1 and a_2. Consider a simple path p from v to a_1. Because τ_2 separates v from τ_0, path p must go through a vertex of $\tau_2 = \{a_2, b, c\}$. Suppose that p does *not* pass through b or c. Then p must pass through a_2. Let path p' be the initial part of p from v to a_2. Note that p' does not pass through any of the vertices of $\tau_1 = \{a_1, b, c\}$. Hence τ_1 does not separate v from τ_0, a contradiction. Therefore every simple path from v to a_1 passes through b or c. The same holds for a_2 instead of a_1 by an analogous argument. Hence, $\{b, c\}$ is a separating pair. But this is a contradiction because there are no separating pairs in a 3-connected component. Therefore we must have $|\tau_1 \cap \tau_2| \leq 1$.

It remains to show that τ_1 and τ_2 each have just one other split component than the one with τ_0. The split component contains v in both cases. Assume that, say τ_1, has one more split component, and let w be a vertex in this component. In τ_1 there is a vertex that is in τ_0 and a vertex that is not in τ_0, say $a_0 \in \tau_1 \cap \tau_0$ and $c_1 \in \tau_1 - \tau_0$. Now there is a path p from v via c_1 and w to a_0. Note that p does not pass through τ_2. But then τ_2 does not separate v from τ_0 anymore which is a contradiction. Hence there is no other split component of τ_1 or τ_2.

Figure 10 illustrates the argument with the graph from Figure 8, where we added vertex w and put edges between w and each vertex of $\tau_1 = \{a_0, b, c\}$. Now w is in a second split component of τ_1 because there are no edges between w and a or v.

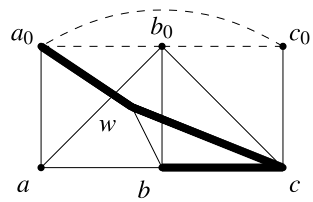


Figure 10: The graph from Figure 8 where we added w and put edges between w and each vertex of $\tau_1 = \{a_0, b, c\}$. Now τ_1 has two split components besides the one with τ_0. However, now $\tau_2 = \{a, b_0, c_0\}$ is no longer a separating triple because the bold path connects b to $a_0 \in \tau_0$ without passing through τ_2.

However, now τ_2 does not separate v from τ_0 anymore because the path (v, c, w, a_0) does not pass through τ_2. □

It might seem surprising that Lemma 5.8 leaves the possibility that crossing separating triples τ_1 and τ_2 are disjoint. However, Figure 11 shows that this case might actually occur.

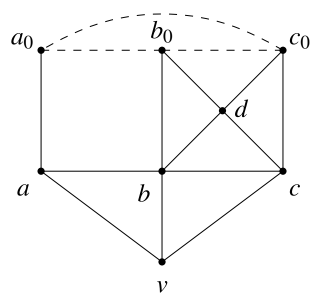


Figure 11: A split component of separating triple $\tau_0 = \{a_0, b_0, c_0\}$. Crossing candidate maximal separating triples are $\tau_1 = \{a_0, b, c\}$ and $\tau_2 = \{a, b_0, c_0\}$. Vertex v is separated from τ_0 by both, τ_1 and τ_2. Note that $\tau_1 \cap \tau_2 = \emptyset$.

A consequence of the lemma is that either there are no crossing candidate maximal separating triples w.r.t. τ_0, or precisely two, τ_1 and τ_2. In the latter case, τ_1 and τ_2 are also the only candidate maximal separating triples. Because each of them has just one split component besides the one with τ_0, it suffices to select one of τ_1 and τ_2 as a maximal separating triple to proceed with in the decomposition.

Definition 5.9. A *maximal separating triple* with respect to τ_0 is either a candidate maximal separating triple which is not crossing with any other candidate maximal separating triple or the lexicographical smaller one of two crossing candidate maximal separating triples.

Based on the notion of maximal separating triples, we can decompose a 3-connected component C into 4-connected components. The decomposition is an iterative process. We have to choose a *root* *separating triple* τ_0 to start with. We need such a triple because only then maximal separating triples are defined. I.e., the decomposition depends on τ_0.

The split components of τ_0 are further split. Consider a split component C' of τ_0. Let $\tau_1, \ldots, \tau_k$ be the maximal separating triples w.r.t. τ_0 in C'. Split C' at $\tau_1, \ldots, \tau_k$ (in any order). There will be one component that contains all triples $\tau_0, \tau_1, \ldots, \tau_k$. This component is 4-connected because it does not contain separating triples. The remaining components are 3-connected and are split components of $\tau_1, \ldots, \tau_k$, respectively. Hence we can iterate the process with $\tau_1, \ldots, \tau_k$ as root, respectively for each component, until all split components are 4-connected.

Recall that by the definition of split components given in Section 2, the vertices of each separating triple are pairwise connected by virtual edges. If there is already an edge in C between two vertices a, b of a separating triple, then we define a again a 3-bond for (a, b). Note that a, b can occur in more than one separating triple. In this case we define an extra 3-bond of (a, b) for every such separating triple.

The 4-*connected component tree of C w.r.t. to root separating triple* τ_0 has a node for τ_0, for all maximal separating triples that occur in the iterative decomposition of C starting with τ_0 as described above, the 4-connected components, and the 3-bonds. We have edges between 4-connected components nodes and their separating triple nodes, and between 3-bond nodes and their separating triple nodes.

This 4-connected component tree can be computed in logspace, since the tasks are similar to those of the biconnected and triconnected component trees.

Lemma 5.10. *The* 4-*connected component tree of a* 3-*connected graph C can be computed and traversed in logspace.*

Proof. The decomposition of C depends on the root separating triple we start with. To fix one such triple, we may take for example the first triple computed by the logspace algorithm which computes all the separating triple in C.

Construction and navigation works essentially in the same way as for the biconnected and triconnected component trees. We mention the points that have to be adapted.

For the traversal of the tree we always store the three vertices of the root separating triple. In case of a 4-connected component D, we store a representative vertex $v \in D - \tau_0$ and the vertices of the parent separating triple τ_0 of D. We can decide whether a further vertex u is in D: $u \in D$ if either u belongs to a maximal separating triple w.r.t. τ_0, or

- there is a path from u to v in $C - \tau$, for all separating triples τ,
- u and v are not separated from τ_0 by any other separating triple, and
- u and v are separated from the root separating triple by τ_0.

Recall that we can compute all separating triples of C. Similarly we can compute all maximal separating triples for the current triple τ_0. By Lemma 5.8 we can identify crossing triples and select the lexicographically smaller one according to Definition 5.9. Hence we can also compute all 4-connected components. As a representative vertex for D we choose the first vertex of D found by the construction algorithm.

We define the order on the children of a node as the order they are computed by the construction algorithm of the tree. To navigate in the tree, we need to compute the parent of a node. We do so by computing the path in the tree from the root to τ_0. □

5.3 Planar arrangement of the 4-connected components

It remains to recombine all the 4-connected components into one planar graph. To do so, we essentially reverse the above decomposition process. However, to obtain a planar graph, we make copies of the components and arrange the copies in a planar way. This has to be done carefully such that the size of the resulting graph is polynomially bounded in the size of the input graph. Also the reachability properties should not change.

Consider a 4-connected component tree $\mathcal{T}$ constructed from a nonplanar 3-connected component C. Let u, v be vertices of C such that we want to know whether there is a path from u to v in C. Let D_u and D_v be the component nodes in $\mathcal{T}$ which contain u and v, respectively. Consider D_u as the root of $\mathcal{T}$ and let P be a simple path from D_u to D_v in $\mathcal{T}$. We construct a planar arrangement of the components of $\mathcal{T}$.

We start by putting the planar component D_u in the new planar arrangement. Inductively assume that we have put some component D of $\mathcal{T}$, and let τ be some child separating triple node of D in $\mathcal{T}$. Furthermore, let the children of τ be the 4-connected component nodes $D_1, D_2, \ldots, D_k$. One of the children is put once in the planar arrangement. The other children are put three times. Figure 12 shows the construction.

The child that is put only once is selected as follows:

1. If a child D_i of τ is a vertex on path P from D_u to D_v in $\mathcal{T}_S$, then we select D_i.
2. If no child of τ is a node on path P but there is a large child D_j, then we select D_j.
3. If none of the first two cases occurs then we select an arbitrary component, say D_1.

Lemma 5.11. *Let C be a nonplanar 3-connected components which is not isomorphic to M_8. Let C' be the graph obtained from C by the above process. C' has the following properties:*

(i) C' is a planar.

(ii) There is a simple path from u to v in C if and only if there is such a path in C'.

(iii) The size of C' is polynomial in the size of C.

(iv) C' can be constructed in logspace.

Proof. Ad *(i)*. Because all components D_i are planar, we only have to argue that the arrangement shown in Figure 12 is planar. For each component D_i there is a planar embedding such that the vertices v_1, v_2, v_3 of τ are at the outer face. This follows for example from a proof of Fáry's theorem that every planar graph can be drawn with straight lines. Therefore we can put the new components $D_i^{(1)}, D_i^{(2)}, D_i^{(3)}$ as shown in Figure 12.

The construction is recursive. The child separating triples of the components D_1 and $D_i^{(1)}, D_i^{(2)}, D_i^{(3)}$, for $i = 2, \ldots, k$, form a triangle within each of these components. Therefore we can continue the construction within the face of each of these triangles.

Ad *(ii)*. Consider a path p from u to v in C such that u is in D and v is in C_1. If p passes through just one of v_1, v_2, v_3, then p directly exists also in C'. If p makes a detour, for example, if p goes from v_1 to v_2

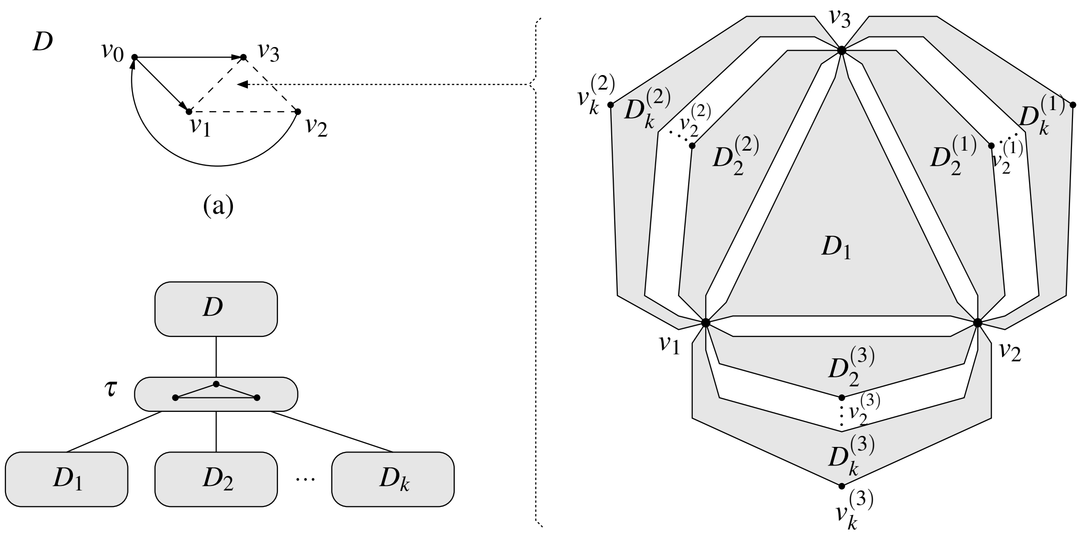


Figure 12: (a) An example of a planar 4-connected component D with separating triple $\tau = \{v_1, v_2, v_3\}$.
(b) A 4-connected component tree with root D, separating triple τ and its children, the 4-connected component nodes $D_1, D_2, \ldots, D_k$. (The children are not shown in (a).)
(c) The planar arrangement of $D_1, D_2, \ldots, D_k$ at separating triple τ. Say, D_1 is the component that occurs only once. For each remaining component D_i we put three components $D_i^{(1)}, D_i^{(2)}, D_i^{(3)}$, for $i = 2, \ldots, k$. These components are essentially copies of D_i. Clearly, we use new vertices for each of the copies, but with the following exception: in $D_i^{(j)}$ we replace v_j by a new vertex $v_j^{(i)}$, but maintain the other two vertices of τ in all components $D_i^{(j)}$, for $i = 2, \ldots, k$. For example in $D_2^{(1)}$ we replace v_1 from D_2 by $v_2^{(1)}$, but keep the vertices v_2, v_3 from D_2 with the same edge connections as in D_2 to the corresponding copies of vertices.
In the example in (a), the construction shown in (c) would be put into the dashed triangle v_1, v_2, v_3.

in component D_2 and also passes vertex v_3 in between, then there will be a path from v_1 to v_2 in $D_2^{(3)}$ that passes the copy $v_2^{(3)}$ instead of v_3. Clearly, we do not introduce any new paths from D to D_1.

Ad *(iii)*. Let N be the size of C. Assume first that the 4-connected component tree $\mathcal{T}$ of C has no large children. Then we get again a recursion formula for the size $\mathcal{S}(N)$ of C',

$$\mathcal{S}(N) = k\mathcal{S}(N/2) + O(N), \tag{5.1}$$

for some constant k. The components are copied ≤ 3 times in the construction above. If component C is a node on the path from S to T in the triconnected component tree of G (see Figure 2 on page 10), then we use the construction shown in Figure 5 on page 14. Hence we get another factor 4 on the number of copies in this case. Therefore $k \leq 3 \cdot 4 = 12$, and we get a polynomial size bound $\mathcal{S}(N) = O(N^{\log k})$.

It remains to consider the case when there are large children in $\mathcal{T}$. The problematic case is when a separating triple τ has a large child and some other child D_i of τ is on path P from D_u to D_v. Then D_i is put only once in our planar arrangement, and the large child is copied three times. The point now is that this situation does *not* occur recursively: if a component is not on path P, this also holds for all of its successors in the tree. Our selection rules for the component to be put only once will from then on choose the large child if there is one. Hence the somewhat larger size of C' we get here is still captured by the $O(N)$ term in recursion formula (5.1) for $S(N)$.

Ad *(iv)*. By Lemma 5.10, the 4-connected component tree $\mathcal{T}$ of C can be constructed in logspace. From $\mathcal{T}$, the construction of C' can be done along the lines of the proof of Lemma 4.5. □

It remains to merge the planar triconnected components to one planar graph. Because the separating pairs in the planar components are connected by a virtual edge, we can attach the components along this edge and the resulting graph is planar. Recall that there is a planar embedding of the components such that the parent separating pair is at the outer face. Hence the merging process yields indeed a planar graph that is also biconnected. This finishes the proof of Theorem 5.1.

Combining Theorem 5.1 and Lemma 3.5 with the result of [7] we get:

Corollary 5.12. *K_5-free* Reachability *is in* $\mathsf{UL} \cap \mathsf{coUL}$.

5.4 Distance and longest paths in K_5-free graphs

To compute distances, we search for shortest simple paths. Again it is easy to see that our graph transformations do not change the distances between the vertices. Also for the longest path problem we can guarantee the same length of a simple path in the resulting graph, but here we have to argue more carefully. Namely, we have to ensure that we do not get longer paths by the copies of the components introduced in the planar arrangement of the 4-connected components. We show that it is not possible for a path to pass through a vertex and its copy.

Lemma 5.13. *Let G be a DAG G and let G be the planar graph constructed from G.*

(i) Let v be vertex of G and let v' be a copy of v in G'. Then there is no path from v to v' in G'.

(ii) G' is a DAG.

Proof. Ad *(i)*. Let D be the 4-connected component that contains v and let $\{v_1, v_2, v_3\}$ be the separating triple that separates D from the rest of G. In G', assume that there is a path p from v in D to v' in D'. The components D and D' are connected by one vertex of v_1, v_2, v_3, say v_1. Hence p has the form $p = (v, u_1, \ldots, u_l, v_1, u'_{l+1}, \ldots, u'_k, v')$, where $u_1, \ldots, u_l \in D$ and $u'_{l+1}, \ldots, u'_k \in D'$. But then we have the cycle $(v, u_1, \ldots, u_l, v_1, u_{l+1}, \ldots, u_k, v)$ in D which is a contradiction because G is acyclic by assumption.

The proof for *(ii)* is similar. A cycle in G' that starts at v has to pass through v_1, v_2, and v_3 in some order by the construction in Figure 12 (c). This yields again a cycle in G. □

We argue that the length of longest paths are not changed by the transformations. Consider Figure 12. Let p be a longest path from v_1 to some vertex v in G. Assume that p goes through component D_2 and that v is in D_1. The interesting case is when p also passes through v_2 and v_3, say in this order. Then we have two ways to go in G':

1. we can go through component $D_2^{(2)}$, pass through $v_2^{(2)}$, and then go to D_1 via v_3.
2. we can go through component $D_2^{(3)}$ to v_2, and then to v_3 through component $D_2^{(1)}$.

The first case corresponds exactly to path p in G because after passing through $v_2^{(2)}$, the path cannot go through v_2 anymore by Lemma 5.13. In the second case the longest simple path from v_1 to v_2 in $D_2^{(3)}$ has the same length as the first part of p from v_1 to v_2 in D_2, because a part of a longest path is again a longest path. Also, the longest path from v_2 to v_3 in $D_2^{(1)}$ has the same length as the second part of p from v_2 to v_3 in C_2. Hence, the second possibility does not lead to a longer path. We conclude that the lengths of longest paths do not change by the reduction.

Theorem 5.14. *1. K_5-free* Distance $\leq_T^{\mathsf{L}}$ *planar* Distance.

2. K_5-free DAG Long-Path $\leq_T^{\mathsf{L}}$ *planar DAG* Long-Path.

As a consequence, we get the following corollary.

Corollary 5.15. *Distances in K_5-free graphs and longest paths in K_5-free DAGs can be computed in* UL $\cap$ coUL.

Conclusions

We showed a reduction from the reachability, the distance and the longest path problem on $K_{3,3}$-free graphs and on K_5-free graphs (DAGs) to the corresponding problem on planar graphs (DAGs), respectively. It would be interesting to extend the result to $K_{3,4}$-free graphs or to K_6-free graphs, for example or even to minor-closed graph classes.

The main open problem clearly is to bring planar reachability into logspace. By our reductions, this would carry over to $K_{3,3}$-free and K_5-free reachability.

Acknowledgments

We thank Samir Datta, Thanh Minh Hoang, Nutan Limaye, Prajakta Nimbhorkar, Mario Szegedy, Raghunath Tewari, Jacobo Torán for helpful discussions. The comments of the anonymous referees were extremely useful to improve the presentation of the paper.

References

[1] ERIC ALLENDER, DAVID MIX BARRINGTON, TANMOY CHAKRABORTY, SAMIR DATTA, AND SAMBUDDHA ROY: Planar and grid graph reachability problems. *Theory on Computing Systems*, 45(4):675–723, 2009. [doi:http://dx.doi.org/10.1007/s00224-009-9172-z] 2

[2] ERIC ALLENDER, SAMIR DATTA, AND SAMBUDDHA ROY: The directed planar reachability problem. In *Proc. 25th annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pp. 238–249, 2005. 2

[3] ERIC ALLENDER AND MEENA MAHAJAN: The complexity of planarity testing. *Information and Computation*, 189:117–134, 2004. 11

[4] TETSUO ASANO: An approach to the subgraph homeomorphism problem. *Theoretical Computer Science*, 38:249–267, 1985. 2, 11

[5] GUISEPPE DI BATTISTA AND ROBERTO TAMASSIA: Incremental planarity testing. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 436–441, 1989. 5

[6] GUISEPPE DI BATTISTA AND ROBERTO TAMASSIA: On-line maintenance of triconnected components with SPQR-trees. *Algorithmica*, 15(4):302–318, 1996. 5

[7] CHRIS BOURKE, RAGHUNATH TEWARI, AND N.V. VINODCHANDRAN: Directed planar reachability is in unambiguous log-space. *ACM Transactions on Computation Theory*, 1(1):1–17, 2009. 2, 4, 11, 15, 26

[8] STEPHEN A. COOK AND PIERRE MCKENZIE: Problems complete for deterministic logarithmic space. *Journal of Algorithms*, 8:385–394, 1987. 6

[9] BIRESWAR DAS, SAMIR DATTA, AND PRAJAKTA NIMBHORKAR: Log-space algorithms for paths and matchings in k-trees. In *Proceedings of the 27th International Symposium on Theoretical Aspects of Computer Science*, pp. 215–226, 2010. 2

[10] SAMIR DATTA, NUTAN LIMAYE, PRAJAKTA NIMBHORKAR, THOMAS THIERAUF, AND FABIAN WAGNER: Planar graph isomorphism is in log-space. In *Proceedings of the 13th Annual IEEE Conference on Computational Complexity (CCC)*, pp. 203–214. IEEE Computer Society, 2009. 9

[11] SAMIR DATTA, PRAJAKTA NIMBHORKAR, THOMAS THIERAUF, AND FABIAN WAGNER: Isomorphism for $K_{3,3}$-free and K_5-free graphs is in log-space. In *Proceedings of the 29th annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pp. 145–156, 2009. 7, 8

[12] MICHAEL ELBERFELD, ANDREAS JAKOBY, AND TILL TANTAU: Logspace versions of the theorems of Bodlaender and Courcelle. In *Proceedings of the 51st Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 143–152, 2010. 2

[13] DICK W. HALL: A note on primitive skew curves. *Bulletin of the American Mathematical Society*, 49(12):935–936, 1943. 11

[14] FRANK HARARY: *Graph Theory*. Addison-Wesley, 1969. 5

[15] JOHN E. HOPCROFT AND ROBERT E. TARJAN: Dividing a graph into triconnected components. *SIAM Journal on Computing*, 2(3):135–158, 1973. 5, 8

[16] ANDREAS JAKOBY AND TILL TANTAU: Logspace algorithms for computing shortest and longest paths in series-parallel graphs. In *27th International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 4855 of *Lecture Notes in Computer Science*, pp. 216–227. Springer, 2007. 2, 3

[17] SAMIR KHULLER: Parallel algorithms for K_5-minor free graphs. Technical Report TR88-909, Cornell University, Computer Science Department, 1988. 2, 16, 17

[18] JAN KYNČL AND TOMÁŠ VYSKOČIL: Logspace reduction of directed reachability for bounded genus graphs to the planar case. *ACM Transactions on Computation Theory (TOCT)*, 1(3):1–11, 2010. 2

[19] NUTAM LIMAYE, MEENA MAHAJAN, AND PRAJAKTA NIMBHORKAR: Longest paths in planar dags in unambiguous logspace. *Chicago Journal of Theoretical Computer Science*, 2010(8), 2010. 3, 16

[20] STEVEN LINDELL: A logspace algorithm for tree canonization (extended abstract). In *Proceedings of the 24th annual ACM Symposium on Theory of Computing (STOC)*, pp. 400–404, 1992. 6

[21] GARY L. MILLER AND VIJAI RAMACHANDRAN: A new graph triconnectivity algorithm and its parallelization. *Combinatorica*, 12:53–76, 1992. 5, 8

[22] OMER REINGOLD: Undirected connectivity in log-space. *Journal of the ACM*, 55(4):1–24, 2008. 2, 4

[23] KLAUS REINHARDT AND ERIC ALLENDER: Making nondeterminism unambiguous. *SIAM Journal of Computing*, 29(4):1118–1131, 2000. 2

[24] THOMAS THIERAUF AND FABIAN WAGNER: The isomorphism problem for planar 3-connected graphs is in unambiguous logspace. *Theory of Computing Systems*, 47(3):655–673, 2010. 3, 16

[25] WILLIAM T. TUTTE: *Connectivity in graphs*. University of Toronto Press, 1966. 5

[26] KLAUS WAGNER: Über eine Eigenschaft der ebenen Komplexe. In *Mathematical Annalen*, volume 114, 1937. 2, 4, 16

AUTHORS

Thomas Thierauf
professor
University of Aalen, Germany
thomas.thierauf@htw-aalen.de
http://image.informatik.htw-aalen.de/~thierauf

Fabian Wagner
University of Ulm,
fabian.wagner@uni-ulm.de
http://www.uni-ulm.de/in/theo/m/alumni/wagner.html