

# Depth-First Search in Directed Planar Graphs, Revisited

Eric Allender   

Rutgers University, USA

Archit Chauhan  

Chennai Mathematical Institute, India

Samir Datta  

Chennai Mathematical Institute, India

## Abstract

We present an algorithm for constructing a depth-first search tree in planar digraphs; the algorithm can be implemented in the complexity class  $AC^1(UL \cap co-UL)$ , which is contained in  $AC^2$ . Prior to this (for more than a quarter-century), the fastest uniform deterministic parallel algorithm for this problem had a runtime of  $O(\log^{10} n)$  (corresponding to the complexity class  $AC^{10} \subseteq NC^{11}$ ).

We also consider the problem of computing depth-first search trees in other classes of graphs, and obtain additional new upper bounds.

**2012 ACM Subject Classification** Complexity Classes, Parallel Algorithms

**Keywords and phrases** Depth-First Search, Planar Digraphs, Parallel Algorithms, Space-Bounded Complexity Classes

**Related Version** <https://ecc.weizmann.ac.il/report/2020/074/>

**Funding** *Eric Allender*: Supported in part by NSF Grants CCF-1909216 and CCF-1909683.

*Archit Chauhan*: Partially supported by a grant from Infosys foundation and TCS PhD fellowship.

*Samir Datta*: Partially supported by a grant from Infosys foundation and SERB-MATRICS grant MTR/2017/000480.

**Acknowledgements** We thank the anonymous referees for their helpful and insightful suggestions, which improved the presentation.

## Preface

Klaus-Jörn Lange has made fundamental contributions to the study of subclasses of NC (such as [20, 25]) and he also was one of the first to identify subtleties in the formulation of unambiguity in the logspace setting [13] and he contributed to our understanding of unambiguous computation [5, 30, 31, 32].

In our contribution to the celebration of Klaus-Jörn Lange’s work, we bring together these two research threads, in order to give a better understanding of the computational complexity of constructing depth-first search trees in planar digraphs.

## 1 Introduction

Depth-first search trees (DFS trees) constitute one of the most useful items in the algorithm designer’s toolkit, and for this reason they are a standard part of the undergraduate algorithmic curriculum around the world. When attention shifted to parallel algorithms in the 1980’s, the question arose of whether NC algorithms for DFS trees exist. An early negative result was that the problem of constructing the *lexicographically-least* DFS tree in a given digraph is complete for P [34]. But soon thereafter significant advances were made in developing parallel algorithms for DFS trees, culminating in the  $RNC^7$  algorithm of

42 Aggarwal, Anderson, and Kao [1]. This remains the fastest parallel algorithm for the problem  
 43 of constructing DFS trees in general graphs, in the probabilistic setting, or in the setting of  
 44 nonuniform circuit complexity. It remains unknown if this problem lies in (deterministic) NC  
 45 (and we do not solve that problem here).

46 More is known for various restricted classes of graphs. For directed acyclic graphs (DAGs),  
 47 the lexicographically-least DFS tree from a given vertex can be computed in  $AC^1$  [16]. (See  
 48 also [17, 9, 19, 33, 23, 22].) For undirected planar graphs, an  $AC^1$  algorithm for DFS trees  
 49 was presented by Hagerup [21]. For more general planar directed graphs Kao and Klein  
 50 presented an  $AC^{10}$  algorithm. Kao subsequently presented an  $AC^5$  algorithm for DFS in  
 51 *strongly-connected* planar digraphs [27]. In stating the complexity results for this prior work  
 52 in terms of complexity classes (such as  $AC^1, AC^{10}$ , etc.), we are ignoring an aspect that  
 53 was of particular interest to the authors of this earlier work: minimizing the number of  
 54 processors. This is because our focus is on classifying the complexity of constructing DFS  
 55 trees in terms of complexity classes. Thus, if we reduce the complexity of a problem from  
 56  $AC^{10}$  to  $AC^2$ , then we view this as a significant advance, even if the  $AC^2$  algorithm uses many  
 57 more processors (so long as the number of processors remains bounded by a polynomial).  
 58 Indeed, our algorithms rely on the logspace algorithm for undirected reachability [35], which  
 59 does not directly translate into a processor-efficient algorithm. We suspect that our approach  
 60 can be modified to yield a more processor-efficient  $AC^3$  algorithm, but we leave that for  
 61 others to investigate.

## 62 1.1 Our Contributions

63 First, we observe that, given a DAG  $G$ , computation of a DFS tree in  $G$  logspace-reduces to  
 64 the problem of reachability in  $G$ . Thus, for general DAGs, computation of a DFS tree lies  
 65 in NL, and for planar DAGs, the problem lies in  $UL \cap \text{co-UL}$  [12, 38]. For classes of graphs  
 66 where the reachability problem lies in L, so does the computation of DFS trees. One such  
 67 class of graphs is planar DAGs with a single source (see [2], where this class of graphs is  
 68 called SMPDs, for **S**ingle-source, **M**ultiple-sink, **P**lanar **D**AGs).

69 For undirected planar graphs, it was shown in [4] that the approach of Hagerup’s  $AC^1$   
 70 DFS algorithm [21] can be adapted in order to show that construction of a DFS tree in a  
 71 planar *undirected* graph logspace-reduces to computing the distance between two nodes in a  
 72 planar digraph. Since this latter problem lies in  $UL \cap \text{co-UL}$  (see Theorem 1), so does the  
 73 problem of DFS for planar *undirected* graphs.

74 Our main contribution in the current paper is to show that a more sophisticated application  
 75 of the ideas in [21] leads to an  $AC^1(UL \cap \text{co-UL})$  algorithm for construction of DFS trees in  
 76 planar *directed* graphs. (That is, we show DFS trees can be constructed by unbounded fan-in  
 77 log-depth circuits that have oracle gates for a set in  $UL \cap \text{co-UL}$ .)<sup>1</sup> Since  $UL \subseteq NL \subseteq SAC^1 \subseteq$   
 78  $AC^1$ , the  $AC^1(UL \cap \text{co-UL})$  algorithm can be implemented in  $AC^2$ . Thus this is a significant  
 79 improvement over the best previous parallel algorithm for this problem: the  $AC^{10}$  algorithm  
 80 of [28], which has stood for 28 years.

---

<sup>1</sup> An earlier version of this work claimed a stronger upper bound, but there was an error in one of the lemmas in that version [3].

## 2 Preliminaries

We assume that the reader is familiar with depth-first search trees (DFS trees), but we provide a few reminders here, to establish the conventions that we will follow.

Given any node  $r$  in a directed graph  $G$ , a depth-first traversal of  $G$  starting at  $r$  is a traversal of all of the nodes reachable in  $G$  from  $r$  obtained by starting with  $r$  as the only node on a stack, and repeating the following steps until the stack is empty: (1) Pop a node  $v$  off the stack and ignore it if it has already been visited. (2) Otherwise mark it as *visited*, and push all unvisited out-neighbours of  $v$  onto the stack. Different depth-first traversals of  $G$  result if the out-neighbours of  $v$  are placed onto the stack in different orders. Each depth-first traversal gives rise to a *depth-first search tree* (namely, the directed tree rooted at  $r$  where the children of each node  $v$  are the out-neighbours  $x$  of  $v$  having the property that, when  $x$  is first marked *visited*,  $x$  has no in-neighbour other than  $v$  that has been marked as *visited*). Of course, given a depth-first search tree  $T$ , it is possible to traverse  $T$  in an order that is not a depth-first traversal of  $G$ . Thus it is more correct to say that we are constructing a depth-first *traversal* of  $G$ , but we follow the established convention by abusing notation and referring to depth-first search trees (DFS trees) and depth-first traversals interchangeably throughout the paper.

In this paper a DFS tree is always a directed rooted tree as detailed above. On the other hand, when we call a digraph a *tree* (as opposed to a *DFS tree*) we mean only that the underlying *undirected* graph forms a connected acyclic graph. Similarly, if the underlying undirected graph is acyclic, the directed graph is said to be a *forest*, and when we refer to the  $k$ -connected components of  $G$ , we are referring to the subgraphs of  $G$  corresponding to the  $k$ -connected components of the underlying undirected graph.

A graph embedded in the plane with no edge crossings is called a *plane graph*. A graph is *planar* if it can be so embedded in the plane. Any (directed) cycle  $C$  in a plane graph divides the plane into two connected regions: the *interior* and the *exterior*. Any vertex not on  $C$  that is embedded in the interior region is said to be *enclosed* by  $C$ . We shall have opportunity to speak of colored graphs; when we say that  $v$  is *immediately enclosed* by the colored cycle  $C$ , it means that  $v$  is enclosed by  $C$  and there is no other colored cycle  $C'$  enclosing  $v$  whose interior is a subset of the interior of  $C$ . A subgraph  $H$  is *strictly enclosed* by  $C$  if no edge of  $H$  lies on  $C$  and every edge of  $H$  (except possibly its endpoints) is embedded in the interior of  $C$ .

We further assume that the reader is familiar with the standard complexity classes  $L$ ,  $NL$  and  $P$  (see e.g. the text [8]). We will also make frequent reference to the logspace-uniform circuit complexity classes  $NC^k$  and  $AC^k$ .  $NC^k$  is the class of problems for which there is a logspace-uniform family of circuits  $\{C_n\}$  consisting of AND, OR, and NOT gates, where the AND and OR gates have fan-in two and each circuit  $C_n$  has depth  $O(\log^k n)$ . (The logspace-uniformity condition implies that each  $C_n$  has only  $n^{O(1)}$  gates.)  $AC^k$  is defined similarly, although the AND and OR gates are allowed unbounded fan-in. An equivalent characterization of  $AC^k$  is in terms of concurrent-read concurrent-write PRAMs with running time  $O(\log^k n)$ , using  $n^{O(1)}$  processors. For more background on these circuit complexity classes, see, e.g., the text [41].

A nondeterministic Turing machine is said to be *unambiguous* if, on every input  $x$ , there is at most one accepting computation path. If we consider logspace-bounded nondeterministic Turing machines, then unambiguous machines yield the class  $UL$ . A set  $A$  is in  $co-UL$  if and only if its complement lies in  $UL$ .

The construction of DFS trees is most naturally viewed as a *function* that takes a graph

128  $G$  and a vertex  $v$  as input, and produces as output an encoding of a DFS tree in  $G$  rooted at  
 129  $v$ . But the complexity classes mentioned above are all defined as sets of *languages*, instead of  
 130 as sets of *functions*. Since our goal is to place DFS tree construction into the appropriate  
 131 complexity classes, it is necessary to discuss how the complexity of functions fits into the  
 132 framework of complexity classes.

133 When  $\mathcal{C}$  is one of  $\{\mathbf{L}, \mathbf{P}\}$ , it is fairly obvious what is meant by “ $f$  is computable in  $\mathcal{C}$ ”; the  
 134 classes of logspace-computable functions and polynomial-time-computable functions should  
 135 be familiar to the reader. However, the reader might be less clear as to what is meant by  
 136 “ $f$  is computable in  $\mathbf{NL}$ ”. As it turns out, essentially all of the reasonable possibilities are  
 137 equivalent. Let us denote by  $\mathbf{FNL}$  the class of functions that are computable in  $\mathbf{NL}$ ; it is  
 138 shown in [24] each of the three following conditions is equivalent to “ $f \in \mathbf{FNL}$ ”.

- 139 1.  $f$  is computed by a logspace machine with an oracle from  $\mathbf{NL}$ .
- 140 2.  $f$  is computed by a logspace-uniform  $\mathbf{NC}^1$  circuit family with oracle gates for a language  
 141 in  $\mathbf{NL}$ .
- 142 3.  $f(x)$  has length bounded by a polynomial in  $|x|$ , and the set  $\{(x, i, b) : \text{the } i^{\text{th}} \text{ bit of } f(x)$   
 143  $\text{is } b\}$  is in  $\mathbf{NL}$ .

144 Rather than use the unfamiliar notation “ $\mathbf{FNL}$ ”, we will abuse notation slightly and refer to  
 145 certain functions as being “computable in  $\mathbf{NL}$ ”.

146 The proof of the equivalence above relies on the fact that  $\mathbf{NL}$  is closed under complement.  
 147 Thus it is far less clear what it should mean to say that a function is “computable in  $\mathbf{UL}$ ”  
 148 since it remains an open question if  $\mathbf{UL}$  is closed under complement (although it is widely  
 149 conjectured that  $\mathbf{UL} = \mathbf{NL}$ ) [36, 7]). However the proof from [24] carries over immediately to  
 150 the class  $\mathbf{UL} \cap \mathbf{co-UL}$ . That is, the following conditions are equivalent:

- 151 1.  $f$  is computed by a logspace machine with an oracle from  $\mathbf{UL} \cap \mathbf{co-UL}$ .
- 152 2.  $f$  is computed by a logspace-uniform  $\mathbf{NC}^1$  circuit family with oracle gates for a language  
 153 in  $\mathbf{UL} \cap \mathbf{co-UL}$ .
- 154 3.  $f(x)$  has length bounded by a polynomial in  $|x|$ , and the set  $\{(x, i, b) : \text{the } i^{\text{th}} \text{ bit of } f(x)$   
 155  $\text{is } b\}$  is in  $\mathbf{UL} \cap \mathbf{co-UL}$ .

156 Thus, if any of those conditions hold, we will say that “ $f$  is computable in  $\mathbf{UL} \cap \mathbf{co-UL}$ ”.

157 The important fact that the composition of two logspace-computable functions is also  
 158 logspace-computable (see, e.g., [8]) carries over with an identical proof to the functions  
 159 computable in  $\mathbf{L}^C$  for any oracle  $C$ . Thus the class of functions computable in  $\mathbf{UL} \cap \mathbf{co-UL}$  is  
 160 also closed under composition. We make implicit use of this fact frequently when presenting  
 161 our algorithms. For example, we may say that a colored labeling of a graph  $G$  is computable  
 162 in  $\mathbf{UL} \cap \mathbf{co-UL}$ , and that, given such a colored labeling, a decomposition of the graph into  
 163 layers is also computable in logspace, and furthermore, that – given such a decomposition of  
 164  $G$  into layers – an additional coloring of the smaller graphs is computable in  $\mathbf{UL} \cap \mathbf{co-UL}$ , etc.  
 165 The reader need not worry that a logspace-bounded machine does not have adequate space  
 166 to store these intermediate representations; the fact that the final result is also computable in  
 167  $\mathbf{UL} \cap \mathbf{co-UL}$  follows from closure under composition. In effect, the bits of these intermediate  
 168 representations are re-computed each time we need to refer to them.

169 The following theorem, due to [39], gives an important example of a function that is  
 170 computable in  $\mathbf{UL} \cap \mathbf{co-UL}$ .

171 ► **Theorem 1.** [39] *The function that takes as input a directed planar graph  $G$  and two*  
 172 *vertices  $x$  and  $y$ , and produces as output the length of the shortest path from  $x$  to  $y$ , lies in*  
 173  *$\mathbf{UL} \cap \mathbf{co-UL}$ .*

174 **Proof.** Thierauf and Wagner [39, Section 4] show that the techniques of [12, 36, 2] can be

175 combined to show that distance in planar graphs can be computed in  $UL \cap co-UL$ , by reducing  
 176 the computation of distance to the planar reachability problem.

177 More precisely, Thierauf and Wagner observe that, given a planar graph  $G$ , the argument  
 178 in [2] shows how to produce a grid graph  $G'$  with certain edges labeled as “distinguished”,  
 179 with the property that every path  $p$  between two vertices in  $G$  can be associated with a  
 180 unique path  $p'$  in  $G'$ , where furthermore the length of the path  $p$  is equal to the number of  
 181 “distinguished” edges in  $p'$ . (Essentially, edges in  $G$  are mapped to paths in  $G'$ , and some of  
 182 the edges in  $G'$  are marked as corresponding to “real” edges in  $G$ .) They then show that a  
 183 modification of the weight function from [12] has the property that, given the weight of a  
 184 path in  $G'$ , one can easily determine the number of “distinguished” edges in the path, and  
 185 thereby determine the distance between two vertices in  $G$ . ◀

186 Finally, we will consider  $AC^k$  circuits augmented with oracle gates for an oracle in  
 187  $UL \cap co-UL$ , which we denote by  $AC^k(UL \cap co-UL)$ .

### 188 **3 DFS in DAGs Logspace-Reduces to Reachability**

189 In this section, we observe that constructing the lexicographically-least DFS tree in a (not-  
 190 necessarily planar) DAG  $G$  can be done in logspace given an oracle for reachability in  $G$ .  
 191 But first, let us define what we mean by the lexicographically-least DFS tree in  $G$ :

192 ▶ **Definition 2.** *Let  $G$  be a DAG, with some ordering on the neighbours of each vertex.*  
 193 *(For example, with adjacency lists, we can consider the ordering in which the neighbours are*  
 194 *presented in the list. But we will also need to consider different orderings.) For any such*  
*ordering, the lexicographic-least DFS traversal of  $G$  is the traversal done by the Algorithm 1.*

**Input:**  $(G, v)$

**Output:** Sequence of edges in DFS tree

visited[ $v$ ]  $\leftarrow$  1

visited[ $w$ ]  $\leftarrow$  0 for all  $w \neq v$

**for** every out neighbour  $w$  of  $v$ , in the given order **do**

**if** visited[ $w$ ] = 0 **then**

        print( $v, w$ )

        DFS( $G, w$ )

**end**

**end**

195 **Algorithm 1** Static DFS routine

196 That is, the lexicographically-least DFS tree is merely a DFS tree, but with the (very  
 197 natural) condition that the children of every vertex are explored in the given order. Im-  
 198 portantly, when we apply this procedure as part of our algorithm for DFS in planar graphs,  
 199 the ordering on the neighbours of  $v$  will be determined *dynamically*. (Note that, in the  
 200 algorithm that defines the lexicographically-least DFS traversal, no reference is made to  
 201 the ordering of the neighbours of  $v$  until it is visited; thus it causes no problems if this  
 202 ordering is not determined until that time.) Also, we will need to apply our algorithm to  
 203 directed acyclic *multigraphs* (i.e., graphs with parallel edges between vertices) where there  
 204 is a logspace-computable function  $f(v, e)$  that computes the ordering of the neighbours of  
 205 vertex  $v$ , assuming that  $v$  is entered using edge  $e$  – where  $e$  can also be “null” if  $v$  is the  
 206 root of the traversal. (That is, if the DFS tree visits vertex  $v$  from vertex  $x$ , and there are

## 6 Depth-First Search in Directed Planar Graphs, Revisited

207 several parallel edges from  $x$  to  $v$ , then the ordering of the neighbours of  $v$  may be different,  
 208 depending on which edge is followed from  $x$  to  $v$ .)<sup>2</sup>

209 As is observed in [16], the unique path from  $s$  to another vertex  $v$  in the lexicographically-  
 210 least DFS tree in  $G$  rooted at  $s$  is the lexicographically-least path in  $G$  from  $s$  to  $v$ .<sup>3</sup>

211 Now consider the following simple algorithm for constructing the lexicographically-least  
 212 path in a DAG  $G$  from  $s$  to  $v$ , shown in Algorithm 2:

**Input:**  $(G, s, v, f)$   
**Output:** Lexicographically-least path from  $s$  to  $v$  under  $f$   
 $current \leftarrow s; e \leftarrow null;$   
**while** ( $current \neq v$ ) **do**  
      $child \leftarrow$  first child of  $current$  (in the order given by  $f(current, e)$ )  
     **while** ( $REACH(child, v) \neq TRUE$ ) **do**  
          $child \leftarrow$  next child of  $current$  (in the order given by  $f(current, e)$ )  
     **end**  
      $e \leftarrow$  a selected edge from  $current$  to  $child$ ; output  $e$   
      $current \leftarrow child;$   
**end**

■ **Algorithm 2** DAG DFS routine

213 The correctness of this algorithm is essentially shown by the proof of Theorem 11 of [16].

214 The algorithm for computing the lexicographically-least DFS tree rooted at  $s$  can thus be  
 215 presented as the composition of two functions  $g$  and  $h$ , where  $g(G, s) = (G, s, L)$ , where  $L$  is  
 216 a list, containing the lexicographically-least path from  $s$  to each vertex  $v$ . Note that the set  
 217 of edges in the DFS tree in  $G$  rooted at  $s$  is exactly the set of edges that occur in the list  $L$   
 218 in  $g(G, s) = (G, s, L)$ . Then  $h(G, s, L)$  is just the result of removing from  $G$  each edge that  
 219 does not appear in  $L$ . The function  $h$  is computable in logspace, whereas  $g$  is computable in  
 220 logspace with an oracle for reachability in  $G$ .

221 As discussed in Section 2, a DFS tree is not only a list of edges; one must also know the  
 222 order in which to explore the children of a node. Given a node  $v$  with children  $x$  and  $y$ ,  
 223 in order to determine whether  $x$  should be visited prior to  $y$ , one can simply compute the  
 224 lexicographically-least path from  $s$  to  $x$  and from  $s$  to  $y$ , and compare.

<sup>2</sup> Let us give additional motivation for having a dynamically-computed ordering on the neighbours of  $v$ . We will be considering a DAG whose vertices consist of strongly-connected components (SCCs) of the original graph  $G$ . We will have already pre-computed *several* DFS trees of *each* SCC: one rooted at *each* node in the SCC. Our final DFS tree will consist of (a) one DFS tree for each SCC (where the root of the DFS tree for SCC  $C$  is some node  $r_C \in C$ ) along with (b) a selected edge  $(v_D, r_C)$  connecting any two SCCs  $D$  and  $C$  that are adjacent in the DFS tree of the DAG. But of course, to fully specify the DFS tree, we also need to have an ordering on the neighbours of each vertex. In practice, we will be using the (precomputed) DFS tree of  $D$  (rooted at  $r_D$ ) to determine the order of neighbours of vertex  $D$  in the DAG (whose vertices are SCCs). The “lexicographically least” property of our DFS tree of the DAG depends only on the ordering of the neighbours (and not on the selection of the specific edge between vertices in the directed acyclic multigraph).

<sup>3</sup> In case a more detailed definition is necessary, here is what is meant by “the lexicographically-least path from  $s$  to  $v$ ”. Let  $p$  and  $p'$  be two paths from  $s$  to  $v$ . If  $p$  is shorter than  $p'$ , then  $p$  precedes  $p'$  in the lexicographic ordering. If  $p$  and  $p'$  have the same length and are not equal, then they each start with  $s$  and agree up through some vertex  $x$ , and first differ at the next vertex. Let us say that  $p$  has the edge  $(x, w)$  and  $p'$  has the edge  $(x, w')$ . The vertex  $x$  is entered via some edge  $e$  (where, if  $x = s$ , then  $e$  is the null edge). The neighbours of  $x$  are ordered according to  $f(x, e)$ . If  $w$  precedes  $w'$  in the ordering  $f(x, e)$ , then  $p$  precedes  $p'$  in the lexicographic ordering.

225 Since reachability in DAGs is a canonical complete problem for NL, we obtain the following  
226 corollary:

227 ► **Corollary 3.** *Construction of lexicographically-least DFS trees for DAGs lies in NL.*

228 Similarly, since reachability in planar directed (not-necessarily acyclic) graphs lies in  
229  $UL \cap \text{co-UL}$  [12, 38], we obtain:

230 ► **Corollary 4.** *Construction of lexicographically-least DFS trees for planar DAGs lies in*  
231  $UL \cap \text{co-UL}$ .

232 A planar DAG  $G$  is said to be an SMPD if it contains at most one vertex of indegree  
233 zero. Reachability in SMPDs is known to lie in  $L$  [2].

234 ► **Corollary 5.** *Construction of lexicographically-least DFS trees for SMPDs lies in  $L$ .*

## 235 4 Overview of the Algorithm

236 The main algorithmic insight that led us to the algorithm in this paper was a generalization  
237 of the layering algorithm that Hagerup developed for *undirected* graphs [21]. We show that  
238 this approach can be modified to yield a useful decomposition of *directed* graphs, where  
239 the layers of the graph have a restricted structure that can be exploited. More specifically,  
240 the strongly-connected components of each layer are what we call *meshes*; we exploit the  
241 properties of meshes to construct paths (which will end up being paths in the DFS trees we  
242 construct) whose removal partitions the graph into significantly smaller strongly-connected  
243 components.

244 The high-level structure of the algorithm is thus:

- 245 1. Construct a planar embedding of  $G$ .
- 246 2. Partition the graph  $G$  into layers (each of which is surrounded by a directed cycle).
- 247 3. Identify one such cycle  $C$  that has properties that will allow us to partition the graph  
248 into smaller weakly-connected components.
- 249 4. Depending on which properties  $C$  satisfies, create a path  $p$  from the exterior face either  
250 to a vertex on  $C$  or to one of the meshes that reside in the layer just inside  $C$ . Removal  
251 of  $p$  partitions  $G$  into weakly-connected components, where each strongly-connected  
252 component therein is smaller than  $G$  by a constant factor.
- 253 5. Let the vertices on this path  $p$  be  $v_1, v_2, \dots, v_k$ . The DFS tree will start with the path  $p$ ,  
254 and append DFS trees for subgraphs  $G_1, G_2, \dots, G_k$  to this path, where  $G_i$  consists of  
255 all of the vertices that are reachable from  $v_i$  that are not reachable from  $v_j$  for any  $j > i$ .  
256 (This is obviously a tree, and it will follow that it is a DFS tree.) Further, decompose  
257 each  $G_i$  into a DAG of strongly-connected components. Build a DFS tree of that DAG,  
258 and then work on building DFS trees of the remaining (smaller) strongly-connected  
259 components.
- 260 6. Each of the steps above can be accomplished in  $UL \cap \text{co-UL}$ , which means that there is  
261 an  $AC^0$  circuit with oracle gates from  $UL \cap \text{co-UL}$  that takes  $G$  as input and produces  
262 the list of much smaller graphs  $G_1, \dots, G_k$ , as well as the path  $p$  that forms the spine  
263 of the DFS tree. We now recursively apply this procedure (in parallel) to each of these  
264 smaller graphs. The construction is complete after  $O(\log n)$  phases, yielding the desired  
265  $AC^1(UL \cap \text{co-UL})$  circuit family.

266 In the exposition below, we first layer the graph in terms of clockwise cycles (which we  
267 will henceforth call red cycles), and obtain a decomposition of the original graph into smaller

268 pieces. We then apply a nested layering in terms of counterclockwise cycles (which we will  
 269 henceforth call blue cycles); ultimately we decompose the graph into units that are structured  
 270 as a DAG, which we can then process using the tools from Section 3. The more detailed  
 271 presentation follows.

#### 272 4.1 Degree Reduction and Expansion

273 ► **Definition 6.** (of  $\text{Exp}^\circ(G)$  and  $\text{Exp}^\circ(G)$ ) Let  $G$  be a plane digraph. The “expanded”  
 274 digraph  $\text{Exp}^\circ(G)$  (respectively,  $\text{Exp}^\circ(G)$ ) is formed by replacing each vertex  $v$  of total degree  
 275  $d(v) > 3$  by a clockwise (respectively, counterclockwise) cycle  $C_v$  on  $d(v)$  vertices, where  
 276 the  $d(v)$  edges incident on  $v$  now connect to the  $d(v)$  vertices on  $C_v$  (so that each of those  
 277 vertices now has degree 3), respecting the cyclic ordering of edges around  $v$ .

278 We will also find it useful to refer to the process of converting  $\text{Exp}^\circ(G)$  (or  $\text{Exp}^\circ(G)$ )  
 279 back to  $G$ , by *contracting* each expanded cycle  $C_v$  back to  $v$ .

280  $\text{Exp}^\circ(G)$  and  $\text{Exp}^\circ(G)$  each have maximum degree bounded by 3; i.e., they are *subcubic*.  
 281 Next we define the clockwise (and counterclockwise) dual for such a graph and also a notion  
 282 of distance.

283 Recall that for an undirected plane graph  $H$ , the dual (multigraph)  $H^*$  is formed by  
 284 placing, for every edge  $e \in E(H)$ , a dual edge  $e^*$  between the face(s) on either side of  $e$  (see  
 285 Section 4.6 from [18] for more details). Faces  $f$  of  $H$  and the vertices  $f^*$  of  $H^*$  correspond  
 286 to each other as do vertices  $v$  of  $H$  and faces  $v^*$  of  $H^*$ . There is also a well-studied notion  
 287 of duality for *directed* plane graphs. The graph that is called the *dual of a directed graph*  
 288 in sources such as [10, 29, 11, 26] corresponds to the edges of weight one in what we define  
 289 below as the *clockwise dual of  $G$* ; for technical reasons we also include additional edges (in  
 290 the reverse direction) of weight zero, and we also make use of a *counterclockwise dual*:

291 ► **Definition 7.** (of Duals  $G^\circ$  and  $G^\circ$ ) Let  $G$  be a plane digraph. Then the clockwise dual  
 292  $G^\circ$  (respectively, counterclockwise dual  $G^\circ$ ) is a weighted bidirected version of the undirected  
 293 dual of the underlying undirected graph of  $G$ . If  $e$  is an edge in  $G$  with faces  $f$  and  $g$  to the  
 294 left and right, respectively (in the direction of travel on  $e$ ), then there is an edge with weight  
 295 one in  $G^\circ$  that is oriented from  $f^*$  to  $g^*$  (thus corresponding to rotating  $e$  90 degrees in a  
 296 clockwise direction). The edge in the other direction, from  $g^*$  to  $f^*$  receives weight zero. (The  
 297 weights in  $G^\circ$  are the opposite, with the weight one edge resulting from a counterclockwise  
 298 rotation, and the other direction having weight zero.). We inherit the definition of dual  
 299 vertices and faces from the underlying undirected dual.

300 ► **Definition 8.** Let  $G$  be a plane subcubic graph, and let  $f$  and  $g$  be faces of  $G$ . Define  
 301  $d^\circ(f, g)$  to be the weight of the minimal-weight path from  $f^*$  to  $g^*$  in  $G^\circ$ . We define  $d^\circ(f, g)$   
 302 similarly.

303 ► **Definition 9.** For a plane subcubic digraph  $G$ , let  $f_0$  be the external face. Define the type  
 304  $\text{type}^\circ(f)$  (respectively,  $\text{type}^\circ(f)$ ) of a face to be the singleton set  $\{d^\circ(f_0, f)\}$  (respectively,  
 305  $\{d^\circ(f_0, f)\}$ ). Generalise this to edges  $e$  by defining  $\text{type}^\circ(e)$  (respectively  $\text{type}^\circ(e)$ ) as the  
 306 set consisting of the union of the  $\text{type}^\circ$  (respectively,  $\text{type}^\circ$ ) of the two faces adjacent to  $e$ .  
 307 Also, for a vertex  $v$ , define  $\text{type}^\circ(v)$  (respectively  $\text{type}^\circ(v)$ ) to be the union, over all faces  $f$   
 308 incident on  $v$ , of  $\text{type}^\circ(f)$  (respectively,  $\text{type}^\circ(f)$ ).

309 It is easy to see by definition of the duals that the types of adjacent faces can differ by  
 310 at most one, and hence no vertex can be adjacent to faces with three distinct types. We  
 311 formalise this in the lemma below:

312 ► **Lemma 10.** *In every subcubic graph  $G$ , the cardinality  $|\text{type}^\circ(x)|, |\text{type}^\circ(x)|$  where  $x$  is*  
 313 *a face, edge or a vertex is at least one and at most 2 and in the latter case consists of*  
 314 *consecutive non-negative integers.*

315 *Further, if  $v \in V(G)$  is such that  $|\text{type}^\circ(v)| = 2$ , then there exist unique  $u, w \in V(G)$ ,*  
 316 *such that  $(u, v), (v, w) \in E(G)$  and  $|\text{type}^\circ(u, v)| = |\text{type}^\circ(v, w)| = 2$ .*

317 **Proof.** We first observe that if  $(f_1, f_2)$  is a dual edge with weight 1, then by the triangle  
 318 inequality we have,  $d^\circ(f_0, f_1) \leq d^\circ(f_0, f_2) \leq d^\circ(f_0, f_1) + 1$ . Now, since each vertex  $v \in V(G)$   
 319 of a subcubic graph is incident on at most 3 faces the only case in which  $|\text{type}^\circ(v)| > 2$   
 320 corresponds to three distinct faces  $f_1, f_2, f_3$  being incident on a vertex. But here the  
 321 undirected dual edges form a triangle such that in the directed dual the edges with weight 1  
 322 are oriented either as a cycle or acyclically. In the former case by three applications of the  
 323 above inequality, we get that  $d^\circ(f_0, f_1) \leq d^\circ(f_0, f_2) \leq d^\circ(f_0, f_3) \leq d^\circ(f_0, f_1)$ , hence all 3  
 324 distances are the same. Therefore  $|\text{type}^\circ(v)| = 1$ .

325 In the latter case, suppose the edges of weight 1 are  $(f_1, f_2), (f_2, f_3), (f_1, f_3)$ , then by the  
 326 above inequality again we get:  $d^\circ(f_0, f_1) \leq d^\circ(f_0, f_2), d^\circ(f_0, f_3) \leq d^\circ(f_0, f_1) + 1$ . Thus, both  
 327  $d^\circ(f_0, f_2), d^\circ(f_0, f_3)$  are sandwiched between two consecutive values  $d^\circ(f_0, f_1), d^\circ(f_0, f_1) +$   
 328  $1$ . Hence  $d^\circ(f_0, f_1), d^\circ(f_0, f_2), d^\circ(f_0, f_3)$  must take at most two distinct values, and  
 329 thus  $|\text{type}^\circ(v)| \leq 2$ . Moreover either  $\text{type}^\circ(f_1) \neq \text{type}^\circ(f_2) = \text{type}^\circ(f_3)$  or  $\text{type}^\circ(f_1) =$   
 330  $\text{type}^\circ(f_2) \neq \text{type}^\circ(f_3)$ . Let  $e_1, e_2, e_3$  be such that,  $e_1^\circ = (f_2, f_3), e_2^\circ = (f_1, f_3), e_3^\circ =$   
 331  $(f_1, f_2)$ . Then the two cases correspond to  $|\text{type}^\circ(e_1)| = |\text{type}^\circ(e_2)| = 2, |\text{type}^\circ(e_3)| = 1$  and  
 332 to  $|\text{type}^\circ(e_1)| = 1, |\text{type}^\circ(e_2)| = |\text{type}^\circ(e_3)| = 2$ , respectively. Noticing that  $e_1, e_3$  are both  
 333 incoming or both outgoing edges of  $v$  completes the proof for the clockwise case. The proof  
 334 for the counterclockwise case is formally identical. ◀

335 ► **Definition 11.** *For a plane subcubic graph  $G$  as above, define  $\text{red}(G)$  to be a colored version*  
 336 *of  $G$ , where vertices and edges with a type of cardinality two in  $G^\circ$  are colored red, and all*  
 337 *other vertices and edges are white. Similarly, define  $\text{blue}(G)$  to be the colored version of  $G$ ,*  
 338 *where vertices and edges with a type of cardinality two in  $G^\circ$  are colored blue, and all other*  
 339 *vertices and edges are white.*

340 We will see later how to apply both the duals in  $G$  to get red and blue layerings of a  
 341 given input graph.

342 We enumerate some properties of  $\text{red}(G)$  and  $\text{blue}(G)$ , where  $G$  is subcubic:

- 343 ► **Lemma 12.** 1. *Red vertices and edges in  $\text{red}(G)$  form disjoint clockwise cycles.*  
 344 2. *No clockwise cycle in  $\text{red}(G)$  consists of only white edges (and hence white vertices).*  
 345 *Similar properties hold for  $\text{blue}(G)$ .*

346 **Proof.** 1. Firstly, note that a red edge must have red endpoints, as they are adjacent to the  
 347 same faces that the edge between them is adjacent to. It is immediate from Lemma 10  
 348 that if  $v$  is a red vertex, it has exactly one red incoming edge and one red outgoing edge,  
 349 proving that they form disjoint cycles. Now consider a red cycle  $C$ . The type of each edge  
 350 of  $C$  must be the same, since if there are two consecutive edges in  $C$  of different types,  
 351 it would make the common vertex adjacent to at least three vertices of different types  
 352 contradicting Lemma 10. This means that the distance in  $G^\circ$  of each face bordering the  
 353 “outside” of  $C$  from the external face is one less than the distance of each face bordering  
 354 the “inside” of  $C$ . But in any *counterclockwise* cycle, the distance in  $G^\circ$  from the external  
 355 face to both sides of  $C$  are the same (by the way distances are defined in  $G^\circ$ ). Thus  $C$  is  
 356 clockwise.

357 2. Suppose  $C$  is a clockwise cycle. Consider the shortest path in  $G^\circ$  from the external face  
 358 to a face enclosed by  $C$ . From the Jordan curve theorem (Theorem 4.1.1 [18]), it must  
 359 cross the cycle  $C$ . The edge dual to the crossing must be red.

360

361 The definitions above, which apply only to subcubic plane graphs, can now be extended  
 362 to a general plane graph  $G$ , by considering the subcubic graphs  $\text{Exp}^\circ(G)$  (and  $\text{Exp}^\circ(G)$ ).  
 363 But first, we must make a simple observation about  $\text{red}(\text{Exp}^\circ(G))$  (respectively about  
 364  $\text{blue}(\text{Exp}^\circ(G))$ ).

365 ► **Lemma 13.** *Let  $v \in V(G)$  be a vertex of degree more than 3. Let  $C_v$  be the corresponding*  
 366 *expanded cycle in  $\text{Exp}^\circ(G)$ . Suppose at least one edge of  $C_v$  is white in  $\text{red}(\text{Exp}^\circ(G))$ . Then*  
 367 *there is a unique red cycle  $C$  that shares edges with  $C_v$ .*

368 **Proof.** First we note that  $C_v$  does not contain anything inside it since it is an expanded  
 369 cycle. By Lemma 12 we know that  $C_v$  has at least one red edge. Suppose it shares one or  
 370 more edges with a red cycle  $R_1$ . Since both cycles are clockwise and  $C_v$  has nothing inside,  
 371 the cycle  $R_1$  must enclose  $C_v$ . Now suppose there is another red cycle  $R_2$  that shares one or  
 372 more edges with  $C_v$ . Then  $R_2$  must also enclose  $C_v$ . But two cycles cannot enclose a cycle  
 373 whilst sharing edges with it without touching each other, which contradicts the above lemma  
 374 that all red cycles in a subcubic graph are vertex disjoint. ◀

375 The last two lemmas allow us to consistently contract the red cycles in  $\text{red}(\text{Exp}^\circ(G))$ , in  
 376 order to obtain a colored version of  $G$  which we call  $\text{Col}^\circ(G)$ . We make this more precise in  
 377 the following:

378 ► **Definition 14.** *The colored graph  $\text{Col}^\circ(G)$  (respectively,  $\text{Col}^\circ(G)$ ) is obtained by labeling*  
 379 *a vertex  $v \in V(G)$  having degree more than 3 as red iff the cycle  $C_v$  in  $\text{red}(\text{Exp}^\circ(G))$  has at*  
 380 *least one red edge and at least one white edge. Otherwise the color of  $v$  is white<sup>4</sup>. All the edges*  
 381 *of  $G$ , and all of the vertices of  $G$  having degree  $\leq 3$  inherit their colors from  $\text{red}(\text{Exp}^\circ(G))$ .*  
 382 *The coloring of  $\text{Col}^\circ(G)$  is similar.*

383 We can now characterize the colorings in the graph  $\text{Col}^\circ(G)$ :

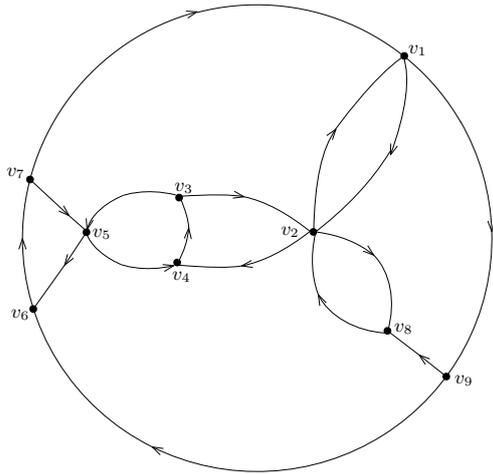
384 ► **Lemma 15.** *The following hold:*

- 385 1. *A red cycle in  $\text{Col}^\circ(G)$  is not connected via a red edge to any vertex in its interior.<sup>5</sup>*  
 386 2. *Every 2-connected component of the red subgraph of  $\text{Col}^\circ(G)$  is a simple clockwise cycle.*

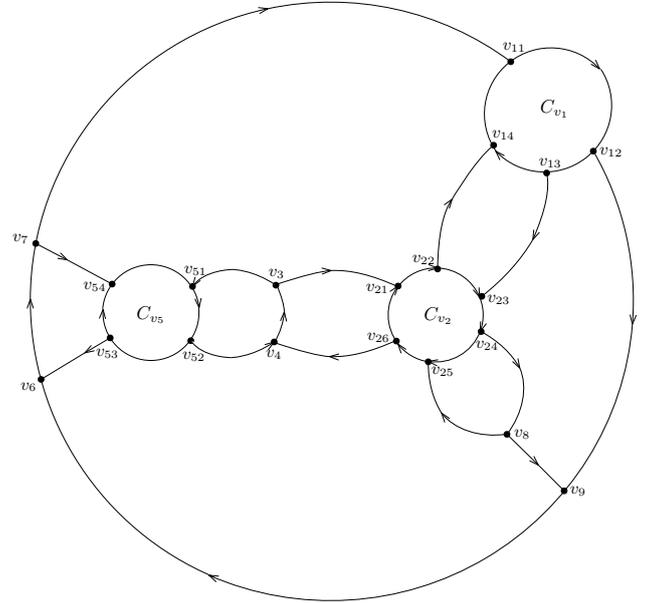
387 **Proof.** Both parts of the lemma follow, if we can establish that the red subgraph of  $\text{Col}^\circ(G)$   
 388 consists of a collection of connected components, each of which is a remnant of exactly  
 389 one red cycle in  $\text{red}(\text{Exp}^\circ(G))$ , where furthermore, each connected component consists of  
 390 a collection of red cycles that intersect at cut vertices (as illustrated in Figure 5). Recall  
 391 that  $\text{Col}^\circ(G)$  results from taking the subcubic graph  $\text{Exp}^\circ(G)$  and contracting each cycle  
 392  $C_v$  where  $v$  is a vertex in  $G$  of degree  $> 3$ . The red subgraph of  $\text{red}(\text{Exp}^\circ(G))$  consists of  
 393 disjoint cycles, by Lemma 12. Contracting any cycle  $C_v$  in  $\text{red}(\text{Exp}^\circ(G))$  does not increase  
 394 the number of red connected components. Thus each red connected component of  $\text{Col}^\circ(G)$  is

<sup>4</sup> This may seem counterintuitive. If  $C_v$  is not entirely red, then  $v$  participates in some red cycle containing edges not in  $C_v$ . Whereas if  $C_v$  is all red, then  $v$  is not connected to other red parts of  $G$ , and thus we color it white.

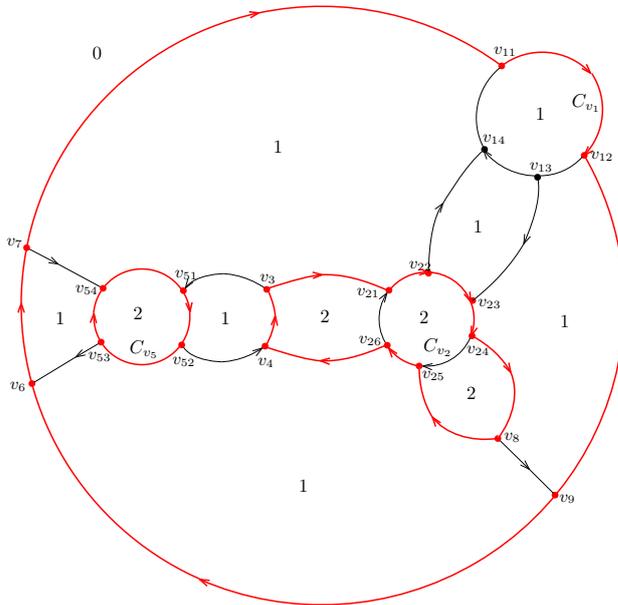
<sup>5</sup> The *interior* of a cycle is the subgraph of  $G$  induced on the vertices that are embedded inside  $C$ , but not on  $C$ .



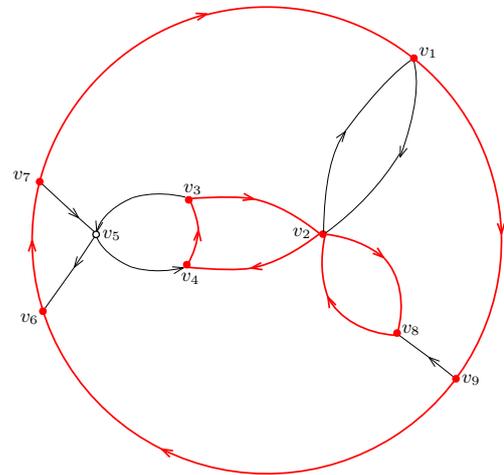
■ **Figure 1** An example of a directed graph  $G$ .



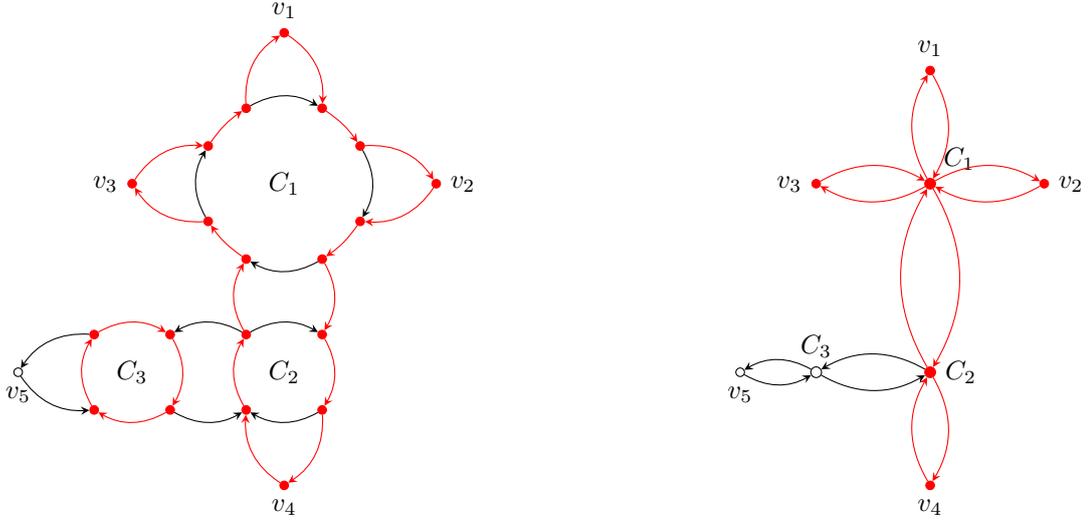
■ **Figure 2** The graph  $\text{Exp}^{\circ}(G)$ .



■ **Figure 3** The graph  $\text{red}(\text{Exp}^{\circ}(G))$ , along with types of the faces.



■ **Figure 4** The graph  $\text{Col}^{\circ}(G)$ . Notice that vertex  $v_5$  was expanded into a red cycle,  $C_{v_5}$ , but is a white vertex in  $\text{Col}^{\circ}(G)$  because all of its edges were red in  $\text{red}(\text{Exp}^{\circ}(G))$ .



■ **Figure 5** An example of contracting expanded cycles. The figure on right shows the graph  $G$  after contracting the expanded cycles  $C_1, C_2, C_3$  in  $\text{Exp}^\circ(G)$ .

395 a remnant of exactly one red cycle in  $\text{Exp}^\circ(G)$ . If  $C_v$  contains all red vertices, then  $v$  is white  
 396 in  $\text{Col}^\circ(G)$  and thus is not part of any red subgraph. If  $C_v$  contains both red and white  
 397 vertices, then  $C_v$  consists of alternating red subpaths and white subpaths, and by Lemma 13  
 398 the red subpaths are all part of the same cycle; let us call it  $R$ . On contracting  $C_v$ ,  $R$   
 399 is transformed into a collection of clockwise red cycles (let's call them  $R_1, R_2, \dots$ ) sharing a  
 400 common cut-vertex  $v$ . Furthermore, for any other  $C_x$  that contains edges from  $R$ , after  $C_v$  is  
 401 contracted,  $C_x$  now shares edges with exactly one of the cycles  $R_i$ . (This is because  $C_x$  is  
 402 embedded inside  $R$ . If it is possible to start at a vertex in  $C_v \cap R$ , and travel to  $C_x$  and then  
 403 back to  $C_v$ , it follows that  $C_x$  is embedded in the closed region between  $R$  and  $C_v$ . When  $C_v$   
 404 is contracted, that segment of  $R$  becomes one of the cycles  $R_i$ , and  $C_x$  is embedded inside  
 405 it.) Thus, when  $C_x$  is contracted,  $R_i$  in turn is transformed into a collection of cycles with  $x$   
 406 as a cut vertex. Inductively, this establishes the claim that, in turn, completes the proof of  
 407 the lemma.

408

◀

409 Although the above lemmas have been proved for the clockwise dual, they also hold for  
 410 counterclockwise dual with red replaced by blue.

## 411 4.2 Layering the Colored Graphs

412 ► **Definition 16.** Let  $x \in V(\text{Col}^\circ(G)) \cup E(\text{Col}^\circ(G))$ . Let  $\ell^\circ(x)$  be one more than the  
 413 minimum integer that occurs in  $\text{type}^\circ(x')$ , for each  $x' \in V(\text{Exp}^\circ(G)) \cup E(\text{Exp}^\circ(G))$  that is  
 414 contracted to  $x$ . Further let  $\mathcal{L}^k(\text{Col}^\circ(G)) = \{x \in V(\text{Col}^\circ(G)) \cup E(\text{Col}^\circ(G)) : \ell^\circ(x) = k\}$ .  
 415 Similarly, define  $\ell^\circ(x)$  and  $\mathcal{L}^k(\text{Col}^\circ(G))$ . We call  $\mathcal{L}^k(\text{Col}^\circ(G))$  the  $k^{\text{th}}$  layer of the graph.

416 See Figure 6 for an example. It is easy to see the following from Lemma 15:

417 ► **Proposition 17.** For every  $x \in V(\text{Col}^\circ(G)) \cup E(\text{Col}^\circ(G))$  the quantity  $\ell^\circ(x)$  is one more  
 418 than the number of red cycles that strictly enclose  $x$  in  $\text{Col}^\circ(G)$ . All the vertices and edges  
 419 of a red cycle of  $\text{Col}^\circ(G)$  lie in the same layer  $\mathcal{L}^{k+1}(\text{Col}^\circ(G))$  for the enclosure depth  $k$  of  
 420 the cycle.

421 We had already noted above that the red subgraph of  $G$  had simple clockwise cycles as  
 422 its 2-connected components. We note a few more lemmas about the structure of a layer of  $G$ :

423 ► **Lemma 18.** *We have:*

424 1. *A red cycle in a layer  $\mathcal{L}^{k+1}(\text{Col}^\circ(G))$  does not contain any vertex/edge of the same layer*  
 425 *inside it.*

426 2. *Any clockwise cycle in a layer consists of only red vertices and edges.*

427 *Dually, a blue cycle in a layer does not contain any vertex or edge of the same layer inside it.*

428 ► **Remark 19.** Notice that the conclusion in the second part of the lemma fails to hold if we  
 429 allow cycles spanning more than one layer.

430 **Proof.** The first part is a direct consequence of Proposition 17. For the second part we mimic  
 431 the proof of the second part of Lemma 12. Consider a clockwise cycle  $C \subseteq \mathcal{L}^{k+1}(\text{Col}^\circ(G))$  that  
 432 contains a white edge  $e$ . Every face adjacent to  $C$  from the outside must have  $\text{type}^\circ = \{k\}$   
 433 because  $C$  is contained in layer  $k + 1$ . Then the  $\text{type}^\circ$  of the faces on either side of  $e$  is the  
 434 same and therefore must be  $\{k\}$ . Let  $f$  be a face enclosed by  $C$  that has  $\text{type}^\circ(f) = \{k\}$ .  
 435 Thus it must be adjacent to a face of  $\text{type}^\circ = \{k - 1\}$ . But this contradicts that every face  
 436 inside and adjacent to  $C$  must have  $\text{type}^\circ = \{k'\}$  for  $k' \geq k$ . ◀

437 The lemmas above show that the strongly-connected components of the red subgraph of a  
 438 layer consist of red cycles touching each other without nesting, in a tree like structure. This  
 439 prompts the following definition:

440 ► **Definition 20.** *For a red cycle  $R \subseteq \mathcal{L}^k(\text{Col}^\circ(G))$  we denote by  $G_R$ , the graph induced by*  
 441 *vertices of  $\mathcal{L}^{k+1}(\text{Col}^\circ(G))$  enclosed by  $R$ .*

442 Now we combine Definitions 14 and 16:

443 ► **Definition 21.** *Each vertex or edge  $x \in V(G) \cup E(G)$  gets a red layer number  $k + 1$  if*  
 444 *it belongs to  $\mathcal{L}^{k+1}(\text{Col}^\circ(G))$  and a blue layer number  $l + 1$ , if it belongs to  $\mathcal{L}^{l+1}(\text{Col}^\circ(G_R))$*   
 445 *where  $R \subseteq \mathcal{L}^k(\text{Col}^\circ(G))$  is the red cycle immediately enclosing  $x$ . In this case, we say that  $x$*   
 446 *belongs to sublayer  $\mathcal{L}^{k+1, l+1}(\text{Col}(G))$ .*

447 *Moreover this defines the colored graph  $\text{Col}(G)$  by giving  $x$  the color red if it is red in*  
 448  *$\text{Col}^\circ(G)$ , and also giving  $x$  the color blue if it is blue in  $\text{Col}^\circ(G_R)$ . (Notice it could be both*  
 449 *red and blue). The vertex  $x$  is white if it is white in both  $\text{Col}^\circ(G_R)$  and  $\text{Col}^\circ(G)$ .*

450 By Proposition 17, we can also say that a sublayer  $\mathcal{L}^{k+1, l+1}(\text{Col}(G))$  thus consists of  
 451 edges/vertices that are strictly enclosed inside  $k$  red cycles and inside  $l$  blue cycles that are  
 452 contained *inside* the red cycle that immediately encloses them.

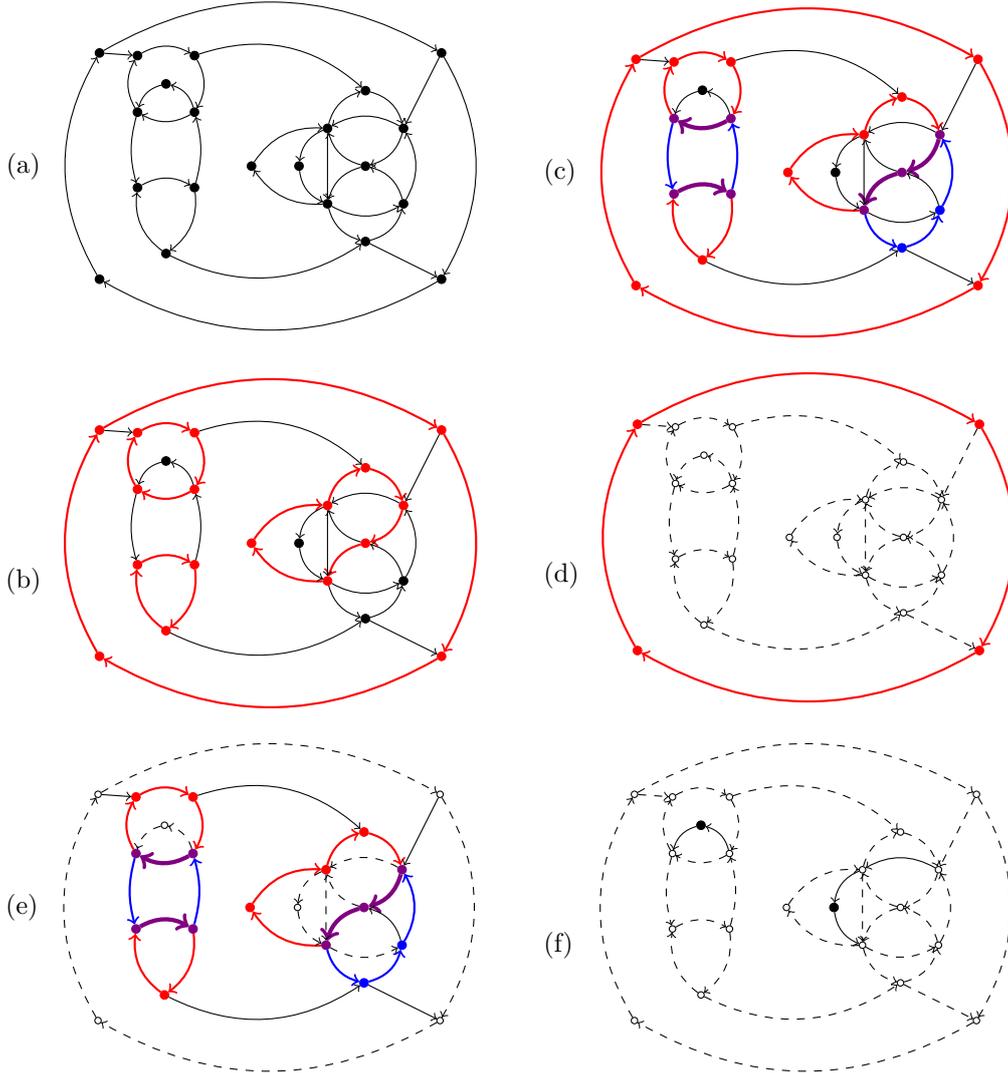
453 We now present some observations and lemmas regarding the structure of a sublayer.

454 Since every edge/vertex in  $\mathcal{L}^{k+1, l+1}(\text{Col}(G))$  has the same red *and* blue layer number, it  
 455 is clear that there can be no nesting of colored cycles. Also we have:

456 ► **Lemma 22.** *Every clockwise cycle in a sublayer  $\mathcal{L}^{k+1, l+1}(\text{Col}(G))$  consists of all red edges*  
 457 *and vertices and any every counterclockwise cycle in the sublayer consists of all blue vertices*  
 458 *and edges. (Some edges/vertices of the cycle can be both red as well as blue)*

459 **Proof.** This is a direct consequence of Lemma 18 applied to the sublayer  $\mathcal{L}^{k+1, l+1}(\text{Col}(G))$ ,  
 460 which is a (counterclockwise) layer in graph  $G_R$  for some red cycle  $R$ . ◀

461 Thus we can refer to clockwise cycles and counterclockwise cycles as red and blue cycles  
 462 respectively.



■ **Figure 6** Figure (a) is a graph  $G$ . Figure (b) is the graph  $\text{Col}^\circ(G)$ . We omit the cycle expansion and contraction procedure here. Figure (c) shows  $\text{Col}(G)$ , which we get from  $G$  after applying blue labellings to each red layer we obtained in the previous figure. The vertices and edges colored purple are those that are red as well as blue. Figure (d) represents the sublayer  $\mathcal{L}^{1,1}$ . The dashed edges and empty vertices are not part of the layer. Figure (e) represents the sublayer  $\mathcal{L}^{2,1}$ . Figure (f) represents the sublayer  $\mathcal{L}^{3,1}$ .

463 ► **Definition 23.** For a red or blue colored cycle  $C$  of layer  $\mathcal{L}^{k,l}(\text{Col}(G))$ , we denote by  $G_C$   
 464 the graph induced by vertices of  $\mathcal{L}^{k',l'}(\text{Col}(G))$  enclosed by  $C$ , where  $\{k',l'\}$  is  $\{k+1,1\}$  or  
 465  $\{k,l+1\}$  according to whether  $C$  is a red or a blue cycle respectively.

466 Note that:

467 ► **Proposition 24.** Two cycles of the same color in  $\mathcal{L}^{k+1,l+1}(G)$  cannot share an edge.

468 This is since neither is enclosed by the other as they belong to the same layer, and as they  
 469 also have the same orientation. Cycles of different colors can share edges but we note:

470 ► **Lemma 25.** *Two cycles of a sublayer  $\mathcal{L}^{k+1, l+1}(\text{Col}(G))$  can only share one contiguous*  
 471 *segment of edges.*

472 **Proof.** Let a red cycle  $R$  and a blue cycle  $B$  in a sublayer share two different contiguous  
 473 segments of edges, from  $x$  to  $u$  and from  $v$  to  $y$ , where the the path  $R(u, v)$  in  $R$  and the path  
 474  $B(u, v)$  in  $B$  share no edges. Notice that the graph  $(R \setminus R(u, v)) \cup B(u, v)$  is also a clockwise  
 475 cycle that encloses the edges of  $R(u, v)$ , contradicting the first part of Lemma 18. ◀

476 We consider the strongly-connected components of a sublayer and note the following lemmas  
 477 regarding them:

478 ► **Lemma 26.** *The trivial strongly-connected components of a sublayer (those that consist of*  
 479 *a single vertex) are white vertices. Let  $H$  be a non-trivial strongly-connected component of a*  
 480 *sublayer, and let  $o$  be the external face of  $H$ . Then*

- 481 1. *Every vertex/edge in  $H$  is blue or red (possibly both).*
- 482 2. *The boundary of every face of  $H$ , except possibly  $o$ , is a directed cycle.*
- 483 3. *Every face of  $H$  other than  $o$  has at least one edge adjacent to  $o$ .*

484 **Proof.** 1. In a non-trivial strongly-connected graph every vertex and edge lies on a cycle  
 485 and therefore by Lemma 22 must be colored red or blue (or both).

486 2. Suppose there is a face  $f$  the boundary of which is not a directed cycle. Look at a  
 487 directed dual (say clockwise) of  $H$ . This dual must be a DAG since the primal is strongly-  
 488 connected. The vertex  $f^*$  in the dual corresponding to face  $f$  of  $H$  has in-degree at least  
 489 one and out-degree at least one since it has boundary edges of both orientations, hence  
 490 the edges adjacent to  $f^*$  do not form a directed cut of the dual.

491 Let  $o^*$  denote the dual vertex corresponding to the outer face  $o$  of  $H$ . In order to prove  
 492 the claim, it is sufficient to show the existence of a directed cut  $C^*$  that separates  $f^*$   
 493 and  $o^*$ , since it would imply by cut cycle duality that there is a directed cycle  $C$  in  $H$   
 494 that encloses the face  $f$  w.r.t the outer face. Since the boundary of  $f$  is not a directed  
 495 cycle, this means that  $C$  must strictly enclose at least one edge of the boundary of  $f$ ,  
 496 contradicting Lemma 18. To see that the cut exists, consider a topological sort ordering  
 497 of the dual (it is a DAG). Let the number of a dual vertex  $v^*$  in the ordering be denoted  
 498 by  $n(v^*)$ . W.l.o.g, let  $n(f^*) < n(o^*)$ . Consider the partition of the dual vertices:

$$499 \quad A = \{v^* \mid n(v^*) \leq n(f^*)\}, B = \{v^* \mid n(v^*) > n(f^*)\}$$

500 By definition of topological sort, all edges across this partition must be directed from  $A$   
 501 to  $B$ , hence it is a directed cut, and therefore it must also contain a subset which is a  
 502 minimal directed cut. But clearly the minimal cut is not the set of edges adjacent to  $f^*$   
 503 since it has both out and in-degree at least one, hence proving the claim. Hence every  
 504 face in  $H$  must be a directed (hence colored) cycle (by Lemma 22).

505 3. We observed from the proof above that no vertex in the dual of  $H$ , except possibly the  
 506 vertex  $o^*$  corresponding to the outer face of  $H$ , can have both in-degree and out-degree  
 507 more than zero (i.e. every dual vertex except  $o^*$  is a source or a sink). Therefore if any  
 508 dual vertex  $f^*$  has a directed path to  $o^*$  or vice versa, then the path must be an edge  
 509 and we are done. Suppose there is no directed path from  $f^*$  to  $o^*$  and w.l.o.g. let  $f^*$  be  
 510 a source. Consider the trivial directed cut  $C_1$ :

$$511 \quad A = \{f^*\}, B = V(H) \setminus A$$

512 This is a cut since there are no edges from  $B$  to  $A$ , and this cut clearly corresponds to  
 513 the directed cycle which is the boundary of face  $f$  in  $H$ .

514 Now consider the cut  $C_2$ :

$$A' = \{v^* \mid v^* \text{ is reachable from } f^*\}, B' = V(H) \setminus A'$$

Clearly this is a  $f^*$ - $o^*$  cut with no edge from a vertex in  $A'$  to a vertex in  $B'$  and  $o^* \in B'$ . But this  $f^*$ - $o^*$  cut is different from  $C_1$  since  $f^*$  is a source vertex and hence  $A'$  has at least one more vertex than just  $f^*$ . Hence this corresponds to a directed cycle in  $H$  that strictly encloses at least some edge of  $f$ , and we again get a contradiction of Lemma 18.

◀

The strongly-connected components of a sublayer hence consist of intersecting red and blue facial cycles, with every face having at least one boundary edge adjacent to the outer face of the component.

► **Definition 27.** We call the strongly-connected components of a sublayer  $\mathcal{L}^{k,l}$  **meshes**.

## 5 Mesh Properties

► **Definition 28.** Given a subgraph  $H$  of  $G$  embedded in the plane, we define the closure of  $H$ , denoted by  $\tilde{H}$ , to be the induced graph on the vertices of  $H$  together with the vertices of  $G$  that lie in the interior of faces of  $H$  (except for the outer face of  $H$ ).

For convenience, we call a face of a graph that is not the outer face an *internal face*.

From Lemmas 22 and 26, we have a bijection: every face of a mesh, except possibly its outer face, is a directed cycle, and every directed cycle in a mesh is the boundary of a face of the mesh.

► **Definition 29.** Let  $0 < \alpha < 1$ . An  $\alpha$  separator of a digraph  $H$  that is a subgraph of a digraph  $G$  is a set of vertices of  $H$  whose removal from  $H$  separates  $\tilde{H}$  into subgraphs, where no strongly-connected component has size greater than  $\alpha|G|$ . An  $(\alpha, r)$  path separator is a sequence of vertices  $\langle v_1, \dots, v_n \rangle$ , that is an  $\alpha$  separator and also is a directed path. Here  $r = v_1$  is called the root of the path separator. We will have occasion to omit either or both of  $\alpha, r$  when they are clear from the context.

► **Definition 30.** Let  $G$  be a graph and let  $M$  be a mesh in a sublayer of  $G$ . For an internal face  $f$  of  $M$ , we define its weight, denoted by  $w(f)$ , to be  $|V(f)|$ . Let  $w(H)$  where  $H$  is a subgraph of  $M$  be defined as  $|V(\tilde{H})|$ .

► **Definition 31.** For a mesh  $M$ , we call a vertex that is adjacent to the outer face of  $M$  an external vertex, and a vertex that is not adjacent to the outer face an internal vertex. We call vertices of degree more than two junction vertices.

If  $p = \langle v_1, v_2, \dots, v_k \rangle$  is a directed path, for  $k \geq 1$ , such that  $v_2, \dots, v_{k-1}$  are all vertices of degree two, but  $v_1, v_k$  have degree more than two<sup>6</sup>, then we call  $p$  a segment. We call  $v_k$  the out junction neighbour of  $v_1$  and  $v_1$  the in junction neighbour of  $v_k$ .

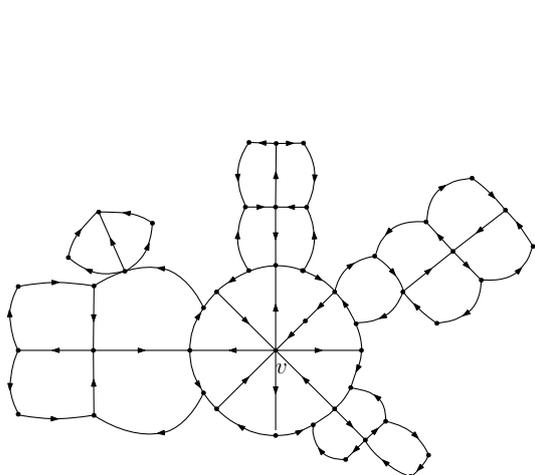
We call a segment with all edges adjacent to the outer face an external segment, and a segment with no edge adjacent to the outer face an internal segment. If the end points of an internal segment are both internal vertices also, we call the segment an i-i-segment.

The rest of this section is devoted to a proof of the following, which asserts that we can construct a path separator in a mesh, assuming that no internal face of the mesh is too large.

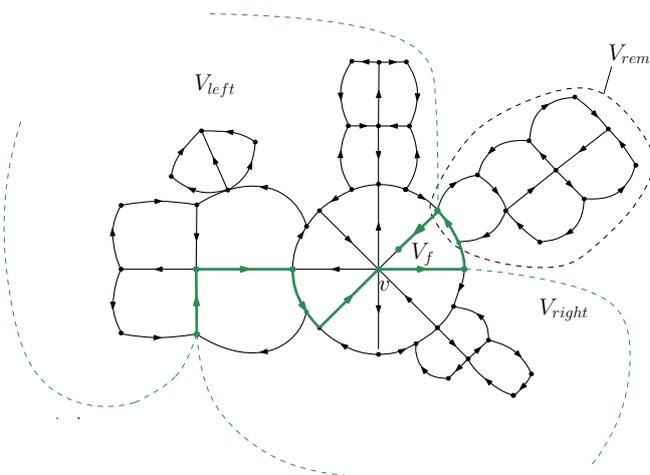
553

---

<sup>6</sup> Notice that here we explicitly allow  $k = 1$  so that  $v_1 = v_k$ .



■ **Figure 7** An example of a mesh



■ **Figure 8** An example of a path separator. The vertex  $v$  is a central node, and the green path is a separator.

554 ▶ **Lemma 32.** Suppose  $w(f) < w(G)/12$  holds for every internal face  $f$  of a mesh  $M$  that is  
 555 a subgraph of  $G$ . Then from any external vertex  $r$  of  $M$ , we can find (in  $UL \cap co-UL$ ) an  $\frac{11}{12}$   
 556 path separator of  $M$ , starting at  $r$ .

557 The high level idea is that using a clique sum decomposition of 2, 3-cliques (see Figure 12)  
 558 we find a “central” vertex  $v$  in the mesh  $M$ , such that we can find a path from the external  
 559 vertex  $r$  to  $v$ , and then extend the path around one of the faces adjacent to  $v$  to get the path  
 560 separator (all faces are directed cycles by Lemma 26). Because every face touches the outer  
 561 face and the weight of every face is small by the hypothesis of the lemma, we can always find a  
 562 face adjacent to  $v$  to encircle such that removing the path leaves no large (weakly) connected  
 563 component. The vertices of  $M$  with degree two (in-degree 1 and out-degree 1 because  $M$  is  
 564 strongly-connected) are not important since they can be seen as just “subdivision” vertices.  
 565 Now we will look at the structure of a mesh around an internal junction vertex, and the way  
 566 the rest of the mesh is attached to that structure. Also, we state here that we will abuse the  
 567 notion of 3-connected components by ignoring the non-junction vertices for convenience.

568 ▶ **Lemma 33.** If  $v$  is an internal junction vertex of a mesh and  $e_1, \dots, e_k$  are the edges  
 569 adjacent to  $v$  in the cyclic order of embedding, then the edges alternate in directions i.e. if  $e_1$   
 570 is outgoing from  $v$ , then  $e_2$  is incoming to  $v$  and  $e_3$  is outgoing and so on. Consequently,  $v$   
 571 has even degree (at least 4).

572 **Proof.** Let  $e_i, e_{i+1}$  be two edges adjacent to  $v$ , that are also adjacent in the cyclic order of  
 573 the drawing. Since they are adjacent in the drawing, they must enclose between them, a  
 574 region, and hence a face, which is not the outer face. But, by Lemma 26, the boundary of  
 575 every non-outer face in a mesh is a directed cycle, hence  $v, e_i, e_{i+1}$  lie on a directed cycle,  
 576 with both edges adjacent to  $v$ . Hence one of them must be an out edge from  $v$ , and the other  
 577 incident towards  $v$ . ◀

578 ▶ **Definition 34.** Let  $v$  be an internal junction vertex of degree  $2d$  in a mesh  $M$ , and let  
 579 its junction neighbours be  $(u_1, w_1, u_2, w_2, \dots, u_d, w_d)$  in clockwise order starting from edge  
 580  $\langle u_1, v \rangle$  (the  $w_i$ 's are out neighbours, and  $u_i$ 's the in neighbours, since junction neighbours  
 581 alternate).

582 Every adjacent pair of edges incident to  $v$  borders a face that is not the outer face. Let  
 583  $f_{u,v,w}$  denote the face bordered by  $v$  and the junction neighbours  $u$  and  $w$  of  $v$  which are  
 584 adjacent in cyclic order around  $v$ . The boundary of  $f_{u,v,w}$  can be written as three disjoint  
 585 parts (except for endpoints),  $\text{segment}(u,v) + \text{segment}(v,w) + \text{petal}_{w,u}$ , where the third part  
 586 denotes a simple path from  $w$  to  $u$  along the face boundary. We will use the notation  $\text{petal}_{w,u}$   
 587 to denote the corresponding boundary for any face  $f_{u,v,w}$  adjacent to  $v$ . We define  $\text{flower}(v)$   
 588 as  $\bigcup \{\text{vertices on the boundary of faces adjacent to } v\}$  (see Figure 9).

589 We note the following property of petals.

590 ► **Proposition 35.** For all adjacent junction neighbour pairs  $w_i, u_j$  of an internal vertex  $v$ ,  
 591  $\text{petal}_{w_i, u_j}$  are disjoint, except possibly the end points.

592 **Proof.** Petals of two faces must be internally disjoint because the corresponding faces share  
 593 the vertex  $v$  and two faces cannot have a non-contiguous intersection, by Lemma 25. ◀

594 For an internal junction vertex  $v$ , the union of the petals around  $\text{flower}(v)$  thus form  
 595 an undirected cycle around  $v$ , with at least four alternations in directions. Now we define  
 596 bridges of the cycle, which (roughly) are components of  $M$  we get after removing  $\text{flower}(v)$ ,  
 597 leaving the points of attachment intact. We use the formal definition of bridges from [40]:

598 ► **Definition 36.** For a subgraph  $H$  of  $M$ , a vertex of attachment of  $H$  is a vertex of  $H$  that  
 599 is incident with some edge of  $M$  not belonging to  $H$ . Let  $J$  be an undirected cycle of  $M$ . We  
 600 define a bridge of  $J$  in  $M$  as a subgraph  $B$  of  $M$  with the following properties:

- 601 1. each vertex of attachment of  $B$  is a vertex of  $J$ .
- 602 2.  $B$  is not a subgraph of  $J$ .
- 603 3. no proper subgraph of  $B$  has both the above properties.

604 We denote by 2-bridge, bridges with exactly two vertices of attachment to the specified cycle,  
 605 and by 3-bridge, bridges with three or more vertices of attachment.

606 Note that for the cycle formed by petals of  $\text{flower}(v)$ , the vertex  $v$  along with paths leading  
 607 to/ coming from  $\text{flower}(v)$  also form a bridge, but we call that a trivial bridge and do not  
 608 take it into consideration.

609 ► **Lemma 37. 1.** The vertices of attachment of a 2-bridge of  $\text{flower}(v)$  must both lie on one  
 610 petal of  $\text{flower}(v)$ .

611 2. The vertices of attachment of a 3-bridge of  $\text{flower}(P)$  can lie on one, or at most two,  
 612 adjacent petals. Moreover, in the latter case the junction neighbour of  $v$  common to both  
 613 petals must be a vertex of attachment of the 3-bridge.

614 3. For an internal vertex  $v$ , and an external vertex  $r$  of  $M$ , let  $p = \langle r, \dots, u_1, v \rangle$  be a  
 615 simple path from  $r$  to  $v$ , where  $u_1$  is an in junction neighbour of  $v$ . Let the other  
 616 junction neighbours of  $v$  be named as in Definition 34 in cyclic order from  $u_1$ . For  $j \in$   
 617  $\{i, i+1\}$ , consider an extended path of  $p$ ,  $p_{w_i, u_j} = \langle r, \dots, u_1, v, w_i \rangle + \text{petal}_{w_i, u_j} + \langle u_j, \dots, v \rangle$ ,  
 618 excluding the last edge incident to  $v$  in the sequence. That is,  $p_{w_i, u_j}$  goes from  $r$  to  $v$ ,  
 619 then to an out junction neighbour  $w_i$ , and then wraps around  $f_{u_j, v, w_i}$  by taking  $\text{petal}_{w_i, u_j}$   
 620 and then the segment back towards  $v$  from  $u_j$ . If there is a bridge of  $\text{flower}(v)$  of which  
 621  $u_1$  is a point of attachment and which also includes the edge of  $p$  incoming to  $u_1$ , we  
 622 denote it by  $B_{in}$ . The set  $V(\widetilde{M}) \setminus V(p_{w_i, u_j})$  can be partitioned into four disconnected  
 623 parts, called  $V_{left}$ ,  $V_{right}$ ,  $V_f$ , and  $V_{rem}$ , such that:

$$\begin{aligned}
 V_{left} = & (\{ \widetilde{f}_{u_1, v, w_1} \cup \widetilde{f}_{u_2, v, w_1} \cup \widetilde{f}_{u_2, v, w_2} \dots \cup \widetilde{f}_{u_i, v, w_{i-1}} \} \cup \{ \widetilde{f}_{u_i, v, w_i} \text{ if } j = i + 1 \}) \\
 & \cup \{ \text{vertices in the closure of bridges attached to the petals of these faces, excluding } B_{in} \} \\
 & \cup \{ \text{the "left" part of } B_{in} \text{ (see Figure 13)} \} \setminus V(p_{w_i, u_j})
 \end{aligned}$$

$$V_{right} = (\{\tilde{f}_{u_i, v, w_{i+1}} \cup \tilde{f}_{u_{i+2}, v, w_{i+1}} \dots \cup \tilde{f}_{u_d, v, w_d}\} \cup \{\tilde{f}_{u_{i+1}, v, w_i} \text{ if } j = i\} \\ \cup \{\text{vertices in the closure of bridges attached to petals petals these faces, excluding } B_{in}\} \\ \cup \{\text{the "right" part of } B_{in} \text{ (see Figure 13)}\} \setminus V(p_{w_i, u_j})$$

$$V_f = \tilde{f}_{u_j, v, w_i} \setminus V(p_{w_i, u_j})$$

$$V_{rem} = \left( \bigcup \{\text{vertices in the closure of all bridges that have vertices} \\ \text{of attachment only in } \text{petal}_{w_i, u_j}\} \right) \setminus V(p_{w_i, u_j}).$$

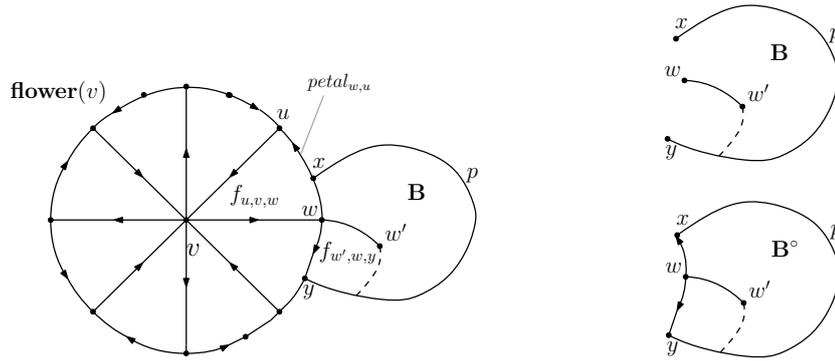
There is no undirected path between any vertex of one of these four sets to any vertex of another. The path  $p_{w_i, u_i}$  is therefore a path separator that gives these components.

- Proof. 1.** Let  $x, y$  be the two vertices of attachment of the 2-bridge  $B$  on  $\text{flower}(v)$ . Since bridges are connected graphs without the edges of the corresponding cycle (by the third property of Definition 36), there must be an undirected path,  $p$  in the bridge connecting  $x, y$ , without using any edge of  $\text{flower}(v)$ . If  $x$  and  $y$  were *not* on the same petal, then this path along with the other petals in  $\text{flower}(v)$ , must clearly enclose a junction neighbour of  $v$ , say  $w$  (see Figure 9). Thus  $w$  is not adjacent to the outer face. Now since  $w$  is an internal junction vertex, and two of its adjacent faces are also adjacent to  $v$ , look at another face  $f$  adjacent to  $w$  and not adjacent to  $v$ . (Internal junction vertices have at least four adjacent faces.) The boundary of this face cannot touch  $B$  since that would make it a part of  $B$  and consequently  $w$  would be a vertex of attachment of  $B$  to  $\text{flower}(v)$ . Therefore the boundary of  $f$  is enclosed within the paths  $p$  and the part of  $\text{flower}(v)$  that is also enclosed by  $p$ . Therefore  $f$  has no external edge, contradicting Lemma 26.
- 2.** Let  $x_1, x_2, \dots, x_k$  be the vertices of attachment of the bridge  $B$  on  $\text{flower}(v)$ , in the cyclic order of boundary of  $\text{flower}(v)$ . Clearly if the vertices of attachment lie on more than two petals of  $v$ , then at least one petal will be completely enclosed by  $B$ , which is not possible since every petal must have at least one external edge. Let us say they lie on two adjacent petals, and the junction neighbour common to both of them is  $w$ . By the same argument as above,  $w$  must have an edge other than those of adjacent petals of  $v$ , that connect it to  $B$ . Therefore  $w$  must be a vertex of attachment of  $B$  to  $\text{flower}(v)$ .
- 3.** First we note that  $\text{petal}_{w_i, u_j}$  will have an external vertex in it since the boundary of every face has at least one external vertex (Lemma 26), and segments  $(u_j, v)$  and  $(v, w_i)$  are internal. Let  $z$  be an external vertex on  $\text{petal}_{w_i, u_j}$ . The path  $p$  starts at external vertex  $r$ , comes to  $u_1, v, w_i$ , and reaches external vertex  $z$  on its way back to  $v$ . It will clearly divide  $\tilde{M}$  into at least two parts by the Jordan Curve theorem. Since  $p_{w_i, u_j}$  is just a wrap around the face  $f_{u_j, v, w_i}$  after  $z$ , it is clear that removing  $p$  puts all of the vertices of  $\tilde{f}_{u, v, w}$  in one disconnected region, while  $w_1, u_2, \dots, w_{i-1}$  and everything connected to them lie in another region, which we call  $V_{left}$ , and  $w_{i+1}, u_{i+2}, \dots, w_d$  and everything connected to them lie in yet another ( $V_{right}$ ).

We introduce another notation for an extension of a bridge:

► **Definition 38.** For a bridge  $B$  of  $\text{flower}(v)$ , we define  $B^\circ$  as  $B$  along with segments of  $\text{flower}(v)$  that lie between consecutive vertices of attachment of  $B$ . We call this the closed bridge of  $B$ .

Now we will give definitions/lemmas regarding the “internal structure” of meshes, that will be useful to define the “center” of a mesh.



■ **Figure 9** A vertex  $v$  and  $\text{flower}(v)$ .  $B$  is a bridge with two points of attachment  $x, y$  on two different petals of  $\text{flower}(v)$ . On the right are drawn the bridge  $B$  itself, and its closed version  $B^\circ$ . The only way the boundary of  $f_{w',w,y}$  can have an external edge is if it touches  $B$ , making  $w$  a point of attachment of  $B$  also.

667 ► **Definition 39.** For a mesh  $M$ , we call its internal-skeleton, denoted by  $I(M)$ , the induced  
668 subgraph on the vertices of  $i$ - $i$ -segments of  $M$ . (See Figure 11.)

669 ► **Lemma 40. 1.** For a mesh  $M$ , the graph  $I(M)$  is a forest.

670 2. If  $H$  is a 3-connected induced subgraph of  $M$  (ignoring subdivision vertices), then  $I(H)$  is  
671 a tree.

672 **Proof. 1.** Suppose there were an undirected cycle in  $M$  of all internal segments, then this  
673 cycle must enclose a face whose boundaries are also all internal segments. This contradicts  
674 Lemma 26 as it states that every face must have at least one external edge, and hence  
675 an external segment. Hence there can be no cycle (directed or undirected) consisting of  
676 all internal segments, and consequently, no cycle (directed or undirected) of all internal  
677 vertices.

678 2. Let  $H$  be a 3-connected induced subgraph of  $M$ . By definition,  $I(H)$  is obtained from  $M$   
679 by removing all external edges and external non-junction vertices. Suppose  $I(H)$  is not a  
680 tree, and hence consists of two or more disconnected trees. Let  $T_1$  and  $T_2$  be any two  
681 trees in  $I(H)$ . Let  $x$  be a vertex in  $T_1$  and  $y$  be a vertex in  $T_2$ . Since  $H$  is 3-connected,  
682 there must be at least three disjoint paths (undirected) between  $x$  and  $y$ . Clearly in a  
683 plane graph, if there are three disjoint paths between two vertices, one of the paths must  
684 be strictly enclosed in the closed region formed by other two. Therefore there must a  
685 path between  $x$  and  $y$  that is strictly enclosed inside the boundary of  $H$ , and hence does  
686 not contain any edge or vertex adjacent to the outer face of  $H$ . Hence  $x$  and  $y$  cannot  
687 become disconnected after removing external edges and external non-junction vertices  
688 leading to a contradiction that  $I(H)$  is disconnected. Therefore  $I(H)$  must be a tree.  
689 ◀

690 We will next give a procedure to define a “center” of a mesh.

691 ► **Definition 41.** For a mesh  $M$ , let  $T_M$  denote the tree obtained by the 1,2-clique sum  
692 decomposition of  $M$ . The nodes of  $T_M$  are of two types, clique nodes (cut vertices or separating  
693 pairs), and piece nodes, which are either 3-connected parts or cycles. Every piece node is  
694 adjacent to a clique node and vice-versa. (See [15, Section 3] for background about this  
695 decomposition in the special case when the graph is 2-connected. For general planar graphs,

696 we can first identify the cut vertices and find the block cut tree. For every clique node of a  
 697 cut vertex  $v$  that is attached to a piece node of block  $B$  containing a 3-connected separating  
 698 pair, we replace the block  $B$  by its triconnected decomposition tree,  $T_B$ , and attach the clique  
 699 node of  $v$  to a piece node of the triconnected block of  $T_B$  that contains  $v$ ).

700 ► **Proposition 42.** *The tree  $T_M$  defined above is a tree decomposition.*

701 **Proof.** It is easy to see that every vertex, as well as every edge of the graph occurs in at  
 702 least one piece node. To see the coherence property, we observe that the only vertices that  
 703 occur in more than one node are those that are part of a 3-connected separating pair or a  
 704 cut vertex. If  $v$  is such a node and is not a cut vertex then it occurs in a subtree of the  
 705 biconnected block it belonged to after the block cut decomposition (since the triconnected  
 706 decomposition is a tree decomposition). If it is a cut vertex, then in our construction, we  
 707 have joined the subtrees in the triconnected decomposition trees to the clique node of  $v$ ,  
 708 which again gives a subtree. ◀

709 We will now use a modified version of the tree vertex separator theorem, to show that  
 710 vertices of one of the nodes of  $T_M$  form a  $\frac{1}{2}$ -separator of  $M$ . We use the following fact from  
 711 the proof of [14, Lemma 7.19].

712 ► **Proposition 43.** *Let  $T_G$  be a tree decomposition of a graph  $G$ . The vertices of one of the*  
 713 *bags of  $T_G$  from a  $\frac{1}{2}$  separator of  $G$ .*

714 Now we define the center of a mesh.

715 ► **Definition 44.** *Consider the  $\frac{1}{2}$  separator node of  $T_M$  as described in Proposition 43. If it*  
 716 *is a separating pair, a cut vertex, or a face cycle, we call that subgraph the center of  $M$ .*

717 *If it is a 3-connected node  $P$ , look at its internal skeleton  $I(P)$ . We construct a new graph*  
 718  *$I'(P)$  which is isomorphic to  $I(P)$  but has edges directed differently. Let  $u, v$  be two adjacent*  
 719 *internal junction vertices of  $M$ . To give direction to a  $\text{segment}(u, v)$  in  $I'(P)$ , we consider*  
 720 *the unique bridge  $B$  of  $\text{flower}(u)$  that contains  $v$  as a point of attachment; we denote the*  
 721 *closed bridge of  $B$  by  $B_u^\circ(v)$ .  $B_v^\circ(u)$  is defined analogously. We orient  $(u, v)$  in the direction*  
 722 *of the heavier of  $B_u^\circ(v)$  and  $B_v^\circ(u)$  (breaking ties arbitrarily), where the weights of  $B_u^\circ(v)$ ,  $B_v^\circ(u)$*   
 723 *are  $|\widetilde{B_u^\circ(v)}|$  and  $|\widetilde{B_v^\circ(u)}|$ , respectively.*

724 *The center of  $M$  is defined to be  $\text{flower}(v)$  in this case, where  $v$  is the sink node of  $I'(P)$ .*

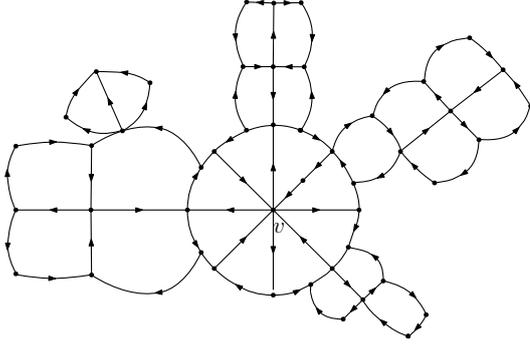
725 We show why  $I'(P)$  cannot have more than one sink.

726 ► **Lemma 45.** *The tree  $I'(P)$  defined above will have exactly one sink vertex.*

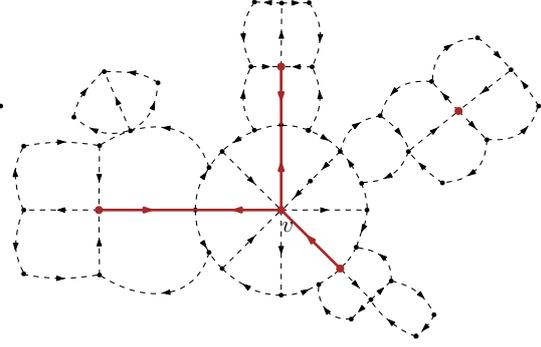
727 Notice, while the underlying undirected graph of  $I'(P)$  is a tree, a sink is defined with respect  
 728 to the orientations specified in the previous definition.

729 **Proof.** Suppose  $I'(P)$  has two junction vertices  $x$  and  $y$  that are sinks. They cannot be  
 730 adjacent, so consider the unique undirected path in  $I'(P)$  between  $x$  and  $y$ . There must be a  
 731 source  $z$  on the path. Let neighbours of  $z$  be  $x', y'$ , lying on the path from  $x$  to  $z$  and from  $z$   
 732 to  $y$  respectively.

733 Let  $B_z^\circ(x')$  and  $B_z^\circ(y')$  denote the bridges of  $\text{flower}(z)$  with points of attachments  $x'$  and  $y'$   
 734 respectively. Then by the orientations of the edges we have:  $|\widetilde{B_z^\circ(x')}| \geq |\widetilde{B_{x'}^\circ(z)}|$  which gives  
 735  $|\widetilde{B_z^\circ(x')}| > |\widetilde{B_z^\circ(y')}|$  since  $B_z^\circ(y')$  is clearly a proper subgraph of  $B_{x'}^\circ(z)$  and  $|\widetilde{B_z^\circ(y')}| \geq |\widetilde{B_{y'}^\circ(z)}|$   
 736 which gives  $|\widetilde{B_z^\circ(y')}| > |\widetilde{B_z^\circ(x')}|$  which is clearly a contradiction. ◀



■ **Figure 10** An example of a mesh



■ **Figure 11** The internal skeleton of the mesh. One of its components is a single node.

737 ► **Lemma 46.** *If the center of  $M$  is  $\text{flower}(v)$ , and  $w$  is an out neighbour of  $v$ , then  $w(\mathbb{B}_v^\circ(w)) \leq$   
 738  $\frac{1}{2}(w(\widetilde{M}) - w(V_{rem}(w, u)))$ , where  $u$  is either of the two in neighbours of  $v$  that are adjacent  
 739 to  $w$  around  $\text{flower}(v)$ , and  $V_{rem}(w, u)$  denotes bridges with all vertices of attachment in  
 740  $\text{petal}_{w, u}$ .*

741 **Proof.** Since the center is  $\text{flower}(v)$ , we have that  $w(\mathbb{B}_v^\circ(w)) \leq w(\mathbb{B}_w^\circ(v))$ . But  $V_{rem}(u, w)$   
 742 has empty intersection with each of  $\mathbb{B}_v^\circ(w)$  and  $\mathbb{B}_w^\circ(v)$ . Thus  $w(\mathbb{B}_v^\circ(w)) + w(\mathbb{B}_w^\circ(v)) \leq$   
 743  $w(\widetilde{M}) - w(V_{rem}(u, w))$ . The lemma follows. ◀

744 ► **Lemma 47. 1.** *If the center of  $M$  is not of the form  $\text{flower}(v)$  where  $v$  is an internal node  
 745 of a 3-connected component, then removing it from  $\widetilde{M}$  disconnects  $\widetilde{M}$  into weakly-connected  
 746 components, each with weight less than  $\frac{1}{2}w(\widetilde{M})$ .*  
 747 **2.** *If the center of  $M$  is  $\text{flower}(v)$  for an internal node  $v$  of a 3-connected component  $P$ , then  
 748 on removing  $\text{flower}(v)$  from  $\widetilde{M}$ , no weakly-connected component has weight more than  
 749  $\frac{1}{2}w(\widetilde{M})$ .*

750 **Proof. 1.** This follows from the vertex separator lemma for trees with weighted vertices.

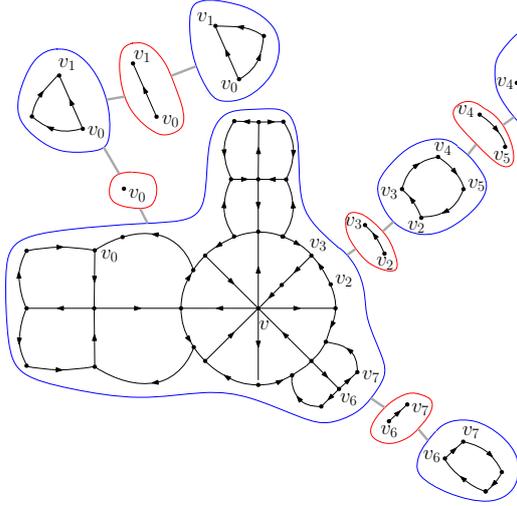
751 **2.** This follows from the  $v$  being the sink node of  $I'(P)$ .

752

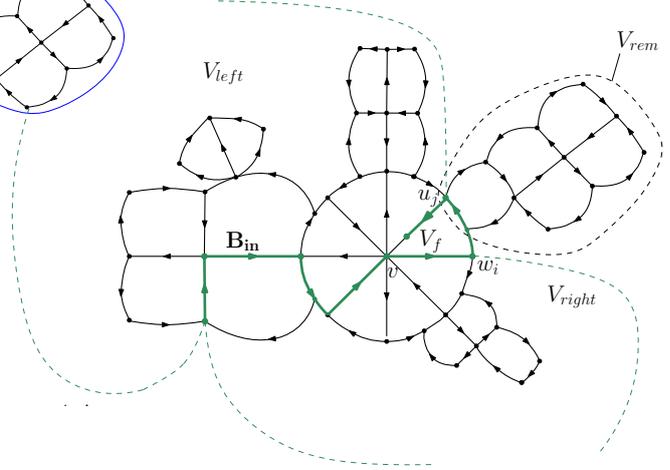
753 ► **Lemma 48.** *For every possible path  $p_{w_i, u_j}$  around  $v$  as defined in Lemma 37,  $V_{rem}$  consists  
 754 of a disjoint union of weakly-connected components, each of which has weight  $\leq \frac{1}{2}(w(M))$ .*

755 **Proof.** A (weakly-connected) component of  $V_{rem}$  is a bridge, attached to  $\text{petal}_{w_i, u_i}$  or to  
 756  $\text{petal}_{w_i, u_{i+1}}$  via its vertices of attachment. In the clique sum decomposition, these vertices of  
 757 attachment will always contain a 1 or 2 separating clique, since if a bridge is attached to a  
 758 petal via three or more nodes, the first and the last vertices of attachment form a separating  
 759 pair that separates the bridge from  $\text{flower}(v)$ . Hence it is a branch remaining in  $T_M$  after  
 760 removing the 3-connected piece node that is central in  $T_M$ . Since every branch after removal  
 761 of the central piece of  $T_M$  has weight  $\leq \frac{1}{2}(w(M))$ , every (weakly) connected component of  
 762  $V_{rem}$  has weight  $\leq \frac{1}{2}(w(M))$ . ◀

763 For a path  $p_{w_i, u_j}$  (where  $j \in \{i, i+1\}$ ) we sometimes use the notation  $V_{rem}(w_i, u_j)$  to  
 764 specify the petal where the bridges of  $V_{rem}$  are attached.



■ **Figure 12** The tree decomposition of the mesh using 1,2-clique sums. The nodes encircled red are clique separator nodes.



■ **Figure 13** An example of a path separator. The vertex  $v$  is a central node, and the green path is a separator.

## 5.1 Mesh Separator Algorithm

Now we give the algorithm to find an  $(\alpha, r)$  path separator in a mesh  $M(G)$ , with  $r \in V(M)$ , assuming the hypothesis of Lemma 32. Recall from Definition 29 an  $(\alpha, r)$  path separator is a directed path starting at (the “root”)  $r$  that is also an  $\alpha$  separator.

1. Find the decomposition tree,  $T_M$  of  $M$  with 2-cliques and 1-cliques as the separating sets.
2. Find the center of the mesh  $M$ . It will either be a cut vertex, a separating pair, a cycle, or flower( $v$ ) for some internal vertex  $v$ .
3. If it is a cut vertex, we just find a path from the root  $r$  to it. If it is a separating pair  $(u, v)$ , both the vertices must lie on a same face, which is a directed cycle. In both this case, and also the case in which the center is a cycle, find a path from the root to any vertex of the face that touches it the first time, and then extend the path by encircling the cycle.
4. If it is flower( $v$ ) for some internal vertex  $v$ , find a path  $p = \langle r, \dots, u_1, v \rangle$  to  $v$ . Let the junction neighbours of  $v$  in clockwise order starting from  $(u_1, v)$ , be  $w_1, u_2, w_2, \dots, w_d$ , with the  $w$ 's being out junction neighbours and the  $u$ 's being in junction neighbours. Starting clockwise from segment  $\langle u, v \rangle$ , find the first index  $i$  and  $j \in \{i, i + 1\}$  s.t. after removing the extended path  $p_{w_i, u_j}$ , (defined in Lemma 37) the remaining strongly-connected components are smaller than  $\frac{11}{12}w(G)$ .

The algorithm above can clearly be implemented in logspace with an oracle for planar reachability, and thus it can be implemented in  $\text{UL} \cap \text{co-UL}$ .

It remains to show that the “first  $i$ ” mentioned in the final step actually exists.

► **Lemma 49.** *If the center of  $M$  is flower( $v$ ) for some internal vertex  $v$ , then there will always exist an adjacent face  $f_{u_i, v, w_i}$  s.t. the path  $p_{w_i, u_i}$  is an  $\frac{11}{12}$ -separator.*

**Proof.** We have the following two cases:

1. For some  $i$  and  $j \in \{i, i + 1\}$ ,  $w(V_{rem}(w_i, u_j)) \geq \frac{1}{2}w(M)$ .

Then by Lemma 48,  $p_{w_i, u_j}$  separates  $V_{rem}(w_i, u_j)$  from the rest of the graph, and also every weakly-connected component in  $V_{rem}(w_i, u_j)$  has weight  $\leq \frac{1}{2}w(M)$ . Hence every weakly-connected component in  $M$  after removing  $p_{w_i, u_j}$  has weight  $\leq \frac{1}{2}w(M)$ .

793 2. For every  $p_{w_i, u_j}$ ,  $w(V_{rem}(w_i, u_j)) \leq \frac{1}{2}w(M)$ .  
794 We know that for any index  $i$  and  $j \in \{i, i+1\}$ , if  $f = f_{u_j, v, w_i}$ , then  $w(f) \leq w(G)/12$  by  
795 the hypothesis of Lemma 32. Starting clockwise from  $p_{u_1, w_1}$ , at first  $V_{left}$  is small, and  
796 on shifting from  $p_{w_i, u_i}$  to  $p_{w_i, u_{i+1}}$  or from  $p_{w_i, u_{i+1}}$  to  $p_{w_{i+1}, u_{i+1}}$ , the increase in  $V_{left}$  is  
797 bounded above by  $w(f) + w(V_{rem}(w_i, u_j)) + w(\widetilde{B}_v^o(w_i))$ . Recall that  
798 a.  $w(f) \leq w(G)/12$  (by the hypothesis of Lemma 32).  
799 b.  $w(V_{rem}(w_i, u_j)) \leq \frac{1}{2}w(M)$  (by hypothesis for this case).  
800 c.  $w(\widetilde{B}_v^o(w_i)) \leq \frac{1}{2}(w(M) - w(V_{rem}(w_i, u_j)))$  (by Lemma 46).  
801 Thus the addition to  $V_{left}$  in each iteration is  $\leq \frac{1}{12}w(G) + w(V_{rem}(w_i, u_j)) + \frac{1}{2}(w(M) -$   
802  $\frac{1}{2}w(V_{rem}(w_i, u_j)))$ , which is equal to  $\frac{1}{12}w(G) + \frac{1}{2}w(V_{rem}(w_i, u_i)) + \frac{1}{2}w(M) \leq \frac{1}{12}wG +$   
803  $\frac{3}{4}w(M)$ . Thus we can stop the first time  $w(V_{left})$  is greater than  $w(G)/12$ . So, we have  
804  $w(V_{left}) \leq \frac{2}{12}w(G) + \frac{3}{4}w(M) \leq \frac{11}{12}w(G)$ , and  $w(V_{right}) \leq \frac{11}{12}w(M)$ , and  $w(f) \leq \frac{1}{12}w(M)$ ,  
805 and  $w(V_{rem}) \leq \frac{1}{2}w(M)$ . Thus we have an upper bound of  $\frac{11}{12}w(G)$  on all the disconnected  
806 components. Hence  $p_{x_i, w_i}$  is a  $\frac{11}{12}$  path separator.

807

◀

## 808 6 Path Separator in a Planar Digraph

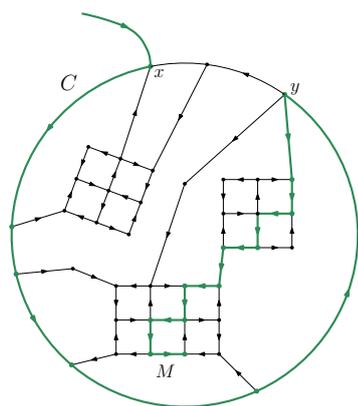
809 Having seen how to construct a path separator in a **mesh**, we now show how to use that to  
810 construct an  $(\frac{11}{12}, r)$  path separator in any planar digraph.

- 811 1. Given a graph  $G$ , first embed the graph so that the root  $r$  lies on the outer face. Through  
812 the root, draw a virtual directed cycle  $C_0$  that encloses the entire graph, and orient it,  
813 say clockwise. Find the layering described in Section 4 and output it on a transducer.  
814 Cycle  $C_0$  will be colored red and will be in the sublayer  $\mathcal{L}^{0,0}$ .  
815 2. In the laminar family of red/blue cycles, find the cycle  $C$  s.t.  $w(C)$  is more than  $|G|/12$ ,  
816 but no colored cycle  $C'$  in the interior of  $C$  has the same property. Such a cycle will  
817 clearly exist (it could be the virtual cycle  $C_0$ ). Let the sublayer of  $C$  be  $\mathcal{L}^{k,l}$ .  
818 3. Find a path  $p$  from the root  $r$  to any vertex  $r_C$  of the cycle  $C$  such that no other vertex  
819 of  $C$  is in the path. As seen above in Lemma 26, the graph in the interior of  $C$  and  
820 belonging to the immediately next sublayer ( $\mathcal{L}^{k+1,l}$  if  $C$  is clockwise and  $\mathcal{L}^{k,l+1}$  if  $C$  is  
821 counter-clockwise) is a DAG of meshes. There are two cases possible:  
822 a. The graph  $\tilde{C}$  has no strongly-connected components of weight larger than  $|G|/12$ . In  
823 this case we simply extend the path  $p$  from  $r_C$  by encircling the cycle  $C$  till the last  
824 vertex and stop.  
825 b. The graph  $\tilde{C}$  has a strongly-connected component of weight more than  $|G|/12$ . In  
826 this case, we extend  $p$  from  $r_C$  by encircling  $C$  till the last vertex  $u$  on  $C$  that can  
827 reach any such component  $M_C$ . Then extend the path from  $u$  to any vertex of  $M_C$   
828 and apply the mesh separator lemma (Lemma 32) to obtain the desired separator.  
829 (Observe that  $M_C$  satisfies the hypothesis of Lemma 32.)

830 ► **Lemma 50.** *The path  $p$  obtained by the above procedure is an  $\frac{11}{12}$  separator.*

831 **Proof.** We look at the two cases from step 3 in the algorithm:

- 832 1. In this case it is clear that the interior and exterior of cycle  $C$  are disconnected by  $p$ .  
833 The exterior of  $C$  has size  $\leq \frac{11}{12}|G|$  (by definition of  $C$ ), and in its interior every strongly-  
834 connected component has weight at most  $|G|/12$ . Thus this satisfies the definition of an  
835  $\frac{11}{12}$  separator.



■ **Figure 14** The cycle  $C$  is a cycle satisfying the property stated in step 2 of the algorithm. The mesh  $M$  in the next sublayer is heavy, so we find a path from the last vertex on  $C$  that can reach  $M$  (in this case  $y$ ), and then apply the algorithm of previous section on  $M$ .

- 836 2. We took the last edge in  $C$  from  $r_C$  that can reach the mesh  $M_C$ , and extended the path  
 837 to  $M_C$ . Thus after removing  $p$ , one weakly-connected component consists of the exterior  
 838 of  $G$ , along with (possibly) some vertices in the interior of  $C$  that cannot reach any “large”  
 839 mesh in the interior. Since  $M_C$  has weight greater than  $\frac{1}{12}|G|$ , no strongly-connected  
 840 component embedded outside of  $M_C$  can have weight more than  $\frac{11}{12}|G|$ . Also, after  
 841 removing path  $p$ , Lemma 32 guarantees that no other strongly-connected component will  
 842 have weight more than  $\frac{11}{12}|G|$ . Thus this is an  $\frac{11}{12}$  separator.

843 Hence overall we can guarantee an  $\frac{11}{12}$  path separator in  $G$ . ◀

## 844 7 Building a DFS Tree Using Path Separators

845 Given a graph  $G$ , one can determine in logspace if  $G$  is planar, and then compute a planar  
 846 embedding [6, 35]. Thus it will suffice to give a recursive divide and conquer algorithm  
 847 for DFS, assuming that  $G$  is presented embedded in the plane, and that we are given a root  
 848 vertex  $r$  on the outer face.

849 A single phase of the algorithm starts with  $G$  and  $r$ , and creates a sequence of subgraphs,  
 850 each of size at most  $\frac{11}{12}$  the size of  $G$ . The algorithm then computes DFS trees for each  
 851 of those graphs (recursively), and the results of (some of) the graphs are sewn together to  
 852 obtain a DFS tree for  $G$ . Each phase can be computed in  $\text{AC}^0(\text{UL} \cap \text{co-UL})$ , and hence the  
 853 entire algorithm can be implemented in  $\text{AC}^1(\text{UL} \cap \text{co-UL})$ .

854 We now describe a single phase in more detail.

- 855 1. Given a planar drawing of  $G$  and a root vertex on the outer face  $r$ , find an  $\frac{11}{12}$  path  
 856 separator  $p = \langle r, v_1, v_2, \dots, v_k \rangle$ , as described in Section 6. Path  $p$  is included in the DFS  
 857 tree.
- 858 2. Let  $R(v)$  denote the set of vertices of  $G$  reachable from  $v$ . Now for every vertex  $v_i$  in  
 859  $p$  compute in parallel:  $R'(v_i) = R(v) \setminus (\bigcup_{j=i+1}^k R(v_j))$ . Our DFS will correspond to first  
 860 traveling along  $p$  to  $v_k$ , doing DFS on  $R(v_k)$ , and then while backtracking on  $p$ , do DFS  
 861 on  $R'(v_i)$  for  $i$  from  $k-1$  down to 1. Given  $G$ , the encodings of  $p$  and  $R'(v_i)$  can all be  
 862 computed in  $\text{AC}^0(\text{UL} \cap \text{co-UL})$ .

- 863 3. For any  $v_i$ ,  $R'(v_i)$  can be written as a DAG of SCCs (strongly-connected components),  
864 where each SCC is smaller than  $\frac{1}{12}|G|$ . In  $AC^0(\text{UL} \cap \text{co-UL})$  we can compute this DAG  
865 and we can compute an encoding of the tuple  $(i, M, v)$  where  $M$  is an SCC in  $R'(v_i)$  and  
866  $v$  is a vertex in  $M$ . Recursively, in parallel, we compute a DFS tree of  $M$  for each tuple  
867  $(i, M, v)$ , using  $v$  as the root. Now we need to show how to sew together (some of) these  
868 DFS trees, to form a DFS tree for  $G$  with root  $r$ . Namely, for each  $i$ , for each  $M \in R'(v_i)$ ,  
869 we will select exactly one  $v$  such that the DFS tree for  $G$  will incorporate the DFS tree  
870 computed for  $(i, M, v)$ , as described next.
- 871 4. Given a triple  $(i, M, v)$ , let  $x_0, x_1, \dots, x_s$  be the order in which the vertices of  $M$  appear  
872 in a DFS traversal where the root  $x_0 = v$ . If  $v$  is such that the DFS tree for  $(i, M, v)$   
873 is incorporated into the DFS tree that we are constructing for  $G$ , then our DFS will  
874 correspond to first following the edges from  $x_0$  that lead to other SCCs in  $R'(v_i)$ . (No  
875 vertex reachable in this way can reach any  $x_j$ , or else that vertex would also be in  $M$ .)  
876 And then we will move on to  $x_1$  and repeat the process, etc. Thus let  $R''_{i,M,v}(x_j) =$   
877  $((R(x_j) \cap R'(v_i)) \setminus M) \setminus (\bigcup_{k < j} R(x_k))$ .
- 878 Our DFS tree for  $G$  is composed by using Algorithm 2 of Section 3, on the multigraph that  
879 has a vertex for each SCC in the DAG of SCCs that makes up any  $R''_{i,M',v}(x_j)$ . Crucially,  
880 the ordering on the edges that leave any node  $M''$  in this multigraph is determined by  
881 the order in which the vertices of  $M''$  are visited in the DFS tree of  $M''$ .
- 882 Let us see in more detail how to use the DFS trees that we computed for each  $(i, M, v)$ ,  
883 by considering how to process the DAG of SCCs in some  $R''_{i,M',v}(x_j)$ . Every SCC in this  
884 DAG is reachable from  $x_j$ . We will be using Algorithm 2 from Section 3 to compute  
885 the lexicographically-least path from  $x_j$  to any SCC  $M''$  in  $R''_{i,M',v}(x_j)$ . We can use any  
886 ordering for the edges that leave  $x_j$  (such as the order in which the edges are presented).  
887 For the other SCCs in the DAG, the ordering must be chosen more carefully. Let us say  
888 that the first edge that leaves  $x_j$  that lies on some path to a node in  $M''$  is  $(x_j, y)$ ; this  
889 edge will be in our DFS tree for  $G$ . The node  $y$  is in some SCC  $N$  in  $R''_{i,M',v}(x_j)$ . A DFS  
890 tree  $T_{i,N,y}$  was computed for  $(i, N, y)$ ; the order in which the nodes of  $T_{i,N,y}$  are visited  
891 imposes an order on the edges that leave  $N$  in the acyclic multigraph. That is the order  
892 that is used, in applying Algorithm 2.
- 893 More generally, when executing the **while** loop in Algorithm 2, if the variable *current*  
894 currently is set to some SCC  $M_1$ , and  $M_2$  is the first SCC adjacent to  $M_1$  (using the  
895 ordering on the edges of  $M_1$ ) that lies on a path to  $M''$ , and this is because there is an  
896 edge  $(w, z)$  where  $w$  is the first node in the traversal of  $M_1$  that is adjacent to any node  
897 of  $M_2$ , then on the next pass through the **while** loop, the ordering on the edges leaving  
898  $M_2$  is determined by the traversal order of the DFS tree that was computed for  $(i, M_2, z)$ .  
899 Let us denote this node  $z$  by  $v_{M_2}$ ; the edge  $(w, v_{M_2})$  will be in the DFS tree for  $G$ .
- 900 5. The final DFS tree for  $G$  thus consists of the path  $p = \langle r, v_1, v_2, \dots, v_k \rangle$  along with the  
901 DFS trees that were computed for each  $(i, M, v_M)$  (for the unique vertex  $v_M$  identified in  
902 the preceding step).

## 903 8 Conclusions and Open Problems

904 Although we give an improved upper bound for the problem of finding DFS trees in planar  
905 digraphs, we do not completely resolve the question of this problem's complexity. Computing  
906 DFS trees in planar graphs is clearly at least as hard as the reachability problem in planar  
907 graphs, and we know of no better lower bound for this problem.

908 In any class of graphs, computing *breadth*-first search trees is no harder than computing

909 distance in the graph. Reachability always reduces to the problem of computing distance,  
 910 but the complexity of these problems can differ. (Reachability in undirected graphs lies in  
 911 logspace [35], whereas computing distance in undirected graphs is complete for NL [37].) For  
 912 directed planar graphs, we have noted that both these problems lie in  $UL \cap co-UL$  (Theorem 1).  
 913 Thus we can also ask whether breadth-first search trees are easier to compute in planar  
 914 directed graphs, than DFS trees.

915 Note that, for *undirected* planar graphs, both breadth-first and depth-first search trees  
 916 reduce to computing distance in *directed* planar graphs [4]. We know of no better lower bound  
 917 for computing DFS trees in undirected planar graphs than the corresponding reachability  
 918 problem.

919 Of course, the outstanding open question in this area is to resolve the complexity of  
 920 computing DFS trees in general (directed or undirected) graphs. The  $RNC^7$  algorithm of [1]  
 921 is unlikely to be optimal. It would be of interest to improve the complexity even in terms of  
 922 nonuniform circuit complexity classes.

## 923 ——— References ———

- 924 1 Alok Aggarwal, Richard J. Anderson, and Ming-Yang Kao. Parallel depth-first search in  
 925 general directed graphs. *SIAM J. Comput.*, 19(2):397–409, 1990. doi:10.1137/0219025.
- 926 2 Eric Allender, David A. Mix Barrington, Tanmoy Chakraborty, Samir Datta, and Sambuddha  
 927 Roy. Planar and grid graph reachability problems. *Theory of Computing Systems*, 45(4):675–  
 928 723, 2009. doi:10.1007/s00224-009-9172-z.
- 929 3 Eric Allender, Archit Chauhan, and Samir Datta. Depth-first search in directed graphs,  
 930 revisited. Technical Report TR20-074, Electronic Colloquium on Computational Complexity  
 931 (ECCC), 2020.
- 932 4 Eric Allender, Archit Chauhan, Samir Datta, and Anish Mukherjee. Planarity, exclusivity,  
 933 and unambiguity. *Electronic Colloquium on Computational Complexity (ECCC)*, 26:39, 2019.
- 934 5 Eric Allender and Klaus-Jörn Lange.  $RSPACE(\log n) \subseteq DSPACE(\log^2 n / \log \log n)$ . *Theory*  
 935 *of Comput. Syst.*, 31(5):539–550, 1998. doi:10.1007/s002240000102.
- 936 6 Eric Allender and Meena Mahajan. The complexity of planarity testing. *Inf. Comput.*,  
 937 189:117–134, 2004.
- 938 7 Eric Allender, Klaus Reinhardt, and Shiyu Zhou. Isolation, matching, and counting: Uniform  
 939 and nonuniform upper bounds. *Journal of Computer and System Sciences*, 59(2):164–181,  
 940 1999.
- 941 8 Sanjeev Arora and Boaz Barak. *Computational Complexity, a modern approach*. Cambridge  
 942 University Press, 2009.
- 943 9 Tetsuo Asano, Taisuke Izumi, Masashi Kiyomi, Matsuo Konagaya, Hirotaka Ono, Yota Otachi,  
 944 Pascal Schweitzer, Jun Tarui, and Ryuhei Uehara. Depth-first search using  $O(n)$  bits. In  
 945 Hee-Kap Ahn and Chan-Su Shin, editors, *Proc. 25th International Symposium on Algorithms*  
 946 *and Computation (ISAAC)*, volume 8889 of *Lecture Notes in Computer Science*, pages 553–564.  
 947 Springer, 2014. doi:10.1007/978-3-319-13075-0\_44.
- 948 10 Giuseppe Di Battista, Peter Eades, Roberto Tamassiao, and Ioannis G. Tollis. *Graph drawing:*  
 949 *algorithms for the visualization of graphs*. Prentice Hall, 1998.
- 950 11 Glencora Borradaile and Philip N. Klein. An  $O(n \log n)$  algorithm for maximum *st*-flow in a  
 951 directed planar graph. *J. ACM*, 56(2):9:1–9:30, 2009. doi:10.1145/1502793.1502798.
- 952 12 Chris Bourke, Raghunath Tewari, and N. V. Vinodchandran. Directed planar reachability is  
 953 in unambiguous log-space. *TOCT*, 1(1):4:1–4:17, 2009. URL: <http://doi.acm.org/10.1145/1490270.1490274>, doi:10.1145/1490270.1490274.
- 954 13 Gerhard Buntrock, Birgit Jenner, Klaus-Jörn Lange, and Peter Rossmanith. Unambiguity and  
 955 fewness for logarithmic space. In Lothar Budach, editor, *Proc. 8th Symposium on Fundamentals*  
 956

- 957 of *Computation Theory (FCT)*, volume 529 of *Lecture Notes in Computer Science*, pages  
958 168–179. Springer, 1991. doi:10.1007/3-540-54458-5\_61.
- 959 **14** Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin  
960 Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer Publishing  
961 Company, Incorporated, 1st edition, 2015.
- 962 **15** Samir Datta, Nutan Limaye, Prajakta Nimbhorkar, Thomas Thierauf, and Fabian Wagner.  
963 Planar graph isomorphism is in log-space. In *Proceedings of the 24th Annual IEEE Conference*  
964 *on Computational Complexity (CCC)*, pages 203–214, 2009. doi:10.1109/CCC.2009.16.
- 965 **16** Pilar de la Torre and Clyde P. Kruskal. Fast parallel algorithms for all-sources lexicographic  
966 search and path-algebra problems. *J. Algorithms*, 19(1):1–24, 1995. doi:10.1006/jagm.1995.  
967 1025.
- 968 **17** Pilar de la Torre and Clyde P. Kruskal. Polynomially improved efficiency for fast parallel  
969 single-source lexicographic depth-first search, breadth-first search, and topological-first search.  
970 *Theory Comput. Syst.*, 34(4):275–298, 2001. doi:10.1007/s00224-001-1008-4.
- 971 **18** Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate texts in mathematics*. Springer,  
972 2016.
- 973 **19** Amr Elmasry, Torben Hagerup, and Frank Kammer. Space-efficient basic graph algorithms.  
974 In *Proc. 32nd International Symposium on Theoretical Aspects of Computer Science (STACS)*,  
975 volume 30 of *LIPICs*, pages 288–301. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015.  
976 doi:10.4230/LIPICs.STACS.2015.288.
- 977 **20** Henning Fernau, Klaus-Jörn Lange, and Klaus Reinhardt. Advocating ownership. In Vijay  
978 Chandru and V. Vinay, editors, *16th Foundations of Software Technology and Theoretical*  
979 *Computer Science (FSTTCS)*, volume 1180 of *Lecture Notes in Computer Science*, pages  
980 286–297. Springer, 1996. doi:10.1007/3-540-62034-6\_57.
- 981 **21** Torben Hagerup. Planar depth-first search in  $O(\log n)$  parallel time. *SIAM J. Comput.*,  
982 19(4):678–704, June 1990. URL: <http://dx.doi.org/10.1137/0219047>, doi:10.1137/  
983 0219047.
- 984 **22** Torben Hagerup. Space-efficient DFS and applications to connectivity problems: Simpler,  
985 leaner, faster. *Algorithmica*, 82(4):1033–1056, 2020. doi:10.1007/s00453-019-00629-x.
- 986 **23** Taisuke Izumi and Yota Otachi. Sublinear-space lexicographic depth-first search for bounded  
987 treewidth graphs and planar graphs. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli,  
988 editors, *Proc. 47th International Colloquium on Automata, Languages and Programming*  
989 *(ICALP)*, volume 168 of *LIPICs*, pages 67:1–67:17. Schloss Dagstuhl - Leibniz-Zentrum für  
990 Informatik, 2020. doi:10.4230/LIPICs.ICALP.2020.67.
- 991 **24** B. Jenner and B. Kirsig. Alternierung und Logarithmischer Platz. Dissertation, Universität  
992 Hamburg, 1989.
- 993 **25** Birgit Jenner, Bernd Kirsig, and Klaus-Jörn Lange. The logarithmic alternation hierarchy  
994 collapses:  $A\sigma_2^1 = a\pi_2^1$ . *Inf. Comput.*, 80(3):269–287, 1989. doi:10.1016/0890-5401(89)  
995 90012-6.
- 996 **26** Ming-Yang Kao. Linear-processor NC algorithms for planar directed graphs I: strongly  
997 connected components. *SIAM J. Comput.*, 22(3):431–459, 1993. doi:10.1137/0222032.
- 998 **27** Ming-Yang Kao. Planar strong connectivity helps in parallel depth-first search. *SIAM J.*  
999 *Comput.*, 24(1):46–62, 1995. doi:10.1137/S0097539792227077.
- 1000 **28** Ming-Yang Kao and Philip N. Klein. Towards overcoming the transitive-closure bottleneck:  
1001 Efficient parallel algorithms for planar digraphs. *Journal of Computer and System Sciences*,  
1002 47(3):459–500, 1993. doi:10.1016/0022-0000(93)90042-U.
- 1003 **29** Haim Kaplan and Yahav Nussbaum. Maximum flow in directed planar graphs with vertex  
1004 capacities. *Algorithmica*, 61(1):174–189, 2011. doi:10.1007/s00453-010-9436-7.
- 1005 **30** Klaus-Jörn Lange. Unambiguity of circuits. *Theor. Comput. Sci.*, 107(1):77–94, 1993. URL:  
1006 [https://doi.org/10.1016/0304-3975\(93\)90255-R](https://doi.org/10.1016/0304-3975(93)90255-R), doi:10.1016/0304-3975(93)90255-R.
- 1007 **31** Klaus-Jörn Lange. An unambiguous class possessing a complete set. In Rüdiger Reischuk  
1008 and Michel Morvan, editors, *14th Annual Symposium on Theoretical Aspects of Computer*

- 1009 (STACS), volume 1200 of *Lecture Notes in Computer Science*, pages 339–350. Springer, 1997.  
1010 doi:10.1007/BFb0023471.
- 1011 32 Klaus-Jörn Lange and Peter Rossmanith. Characterizing unambiguous augmented pushdown  
1012 automata by circuits. In Branislav Rován, editor, *Proc. Mathematical Foundations of Computer  
1013 Science (MFCS)*, volume 452 of *Lecture Notes in Computer Science*, pages 399–406. Springer,  
1014 1990. doi:10.1007/BFb0029635.
- 1015 33 Maxim Naumov, Alysson Vrieling, and Michael Garland. Parallel depth-first search for directed  
1016 acyclic graphs. In *Proc. 7th Workshop on Irregular Applications: Architectures and Algorithms*,  
1017 pages 4:1–4:8, 2017. doi:10.1145/3149704.3149764.
- 1018 34 John H. Reif. Depth-first search is inherently sequential. *Inf. Process. Lett.*, 20(5):229–234,  
1019 1985. doi:10.1016/0020-0190(85)90024-9.
- 1020 35 Omer Reingold. Undirected connectivity in log-space. *J. ACM*, 55(4), 2008.
- 1021 36 Klaus Reinhardt and Eric Allender. Making nondeterminism unambiguous. *SIAM J. Comput.*,  
1022 29(4):1118–1131, 2000. URL: <https://doi.org/10.1137/S0097539798339041>, doi:10.1137/  
1023 S0097539798339041.
- 1024 37 Till Tantau. Logspace optimization problems and their approximability properties. *Theory  
1025 Comput. Syst.*, 41(2):327–350, 2007. doi:10.1007/s00224-007-2011-1.
- 1026 38 Raghunath Tewari and N. V. Vinodchandran. Green’s theorem and isolation in planar  
1027 graphs. *Inf. Comput.*, 215:1–7, 2012. URL: <https://doi.org/10.1016/j.ic.2012.03.002>,  
1028 doi:10.1016/j.ic.2012.03.002.
- 1029 39 Thomas Thierauf and Fabian Wagner. The isomorphism problem for planar 3-connected  
1030 graphs is in unambiguous logspace. *Theory Comput. Syst.*, 47(3):655–673, 2010. doi:10.1007/  
1031 s00224-009-9188-4.
- 1032 40 W. T. Tutte. Separation of vertices by a circuit. *Discrete Mathematics*, 12(2):173–184, 1975.
- 1033 41 H. Vollmer. *Introduction to Circuit Complexity: A Uniform Approach*. Springer-Verlag New  
1034 York Inc., 1999. doi:10.1007/978-3-662-03927-4.