Electronic Colloquium on Computational Complexity, Report No. 71 (2014)

$\widetilde{O}(\sqrt{n})$-Space and Polymomial-time Algorithm for the Planar Directed Graph Reachability Problem

Tetsuo Asano*, David Kirkpatrick†, Kotaro Nakagawa‡and Osamu Watanabe‡

Draft for ECCC, May 4, 2014

1 Introduction

We consider the reachability problem in planar directed graphs. For its formal definition, we begin by defining the reachability in general directed graphs.

Directed Graph Reachability Problem
Input: Directed graph $G = (V, E)$, start vertex v_0, and goal vertex v_*.
Task: Determine whether there exists a path from v_0 to v_* in G.

Throughout this paper we will use n to denote the number of vertices of an input graph, which is the unique input size parameter.

For a directed graph $G = (V, E)$, its *underlying graph* is the undirected graph $`G = (V, `E)$, where the vertex pair $\{u, v\}$ belongs to $`E$ if and only if at least one of (u, v) or (v, u) belongs to E. The *planar directed graph reachability problem* is a special case of the reachability problem where we restrict attention to input graphs whose underlying graph is *planar*. For a simpler setting to introduce some of our algorithmic ideas, we also consider the *grid directed graph reachability problem*, where we restrict attention to input graphs whose underlying graph is an edge-induced subgraph of a square grid. We will frequently refer to these problems with the shorter names "planar reachability" and "grid reachability."

The directed graph reachability problem is a core problem in computational complexity theory. It is a canonical complete problem for the nondeterministic log-space, NL, and the famous open question L = NL is essentially asking whether the problem is solvable deterministically in log-space. The standard breadth first search algorithm and Savitch's algorithm are two of the most fundamental algorithms known for solving the directed graph reachability problem. The former has a (roughly) linear space and time implementation, and the latter uses only $O((\log n)^2)$-space but requires $\Theta(n^{\log n})$ time. Hence a natural and significant question is whether we can design an algorithm for directed graph reachability that is efficient in both space and time. In particular, can we design a polynomial-time algorithm for the directed graph reachability problem that uses only $O(n^\epsilon)$-space for some small constant $\epsilon < 1$? This question was asked by Wigderson in his excellent survey paper [13], and it remains unsettled. The best known result in this direction is the two decades old bound due to Barns, Buss, Ruzzo and Schieber [4], who showed a polynomial-time algorithm for the problem that uses $O(n/2^{\sqrt{\log n}})$ space. Note that this space bound is only slightly sublinear, and improving this bound remains a significant open question. In fact, there are indications that it may be difficult to improve this bound because there are

*Japan Advanced Institute of Technology, Japan
†Dept. of Comp. Sci., Univ. British Columbia, Canada (kirk@cs.ubc.ca)
‡Dept. of Math. and Comput. Sci., Tokyo Inst. of Tech., Japan ({nakagaw1, watanabe}@is.titech.ac.jp)

ISSN 1433-8092

matching *lower bounds* known for solving the directed graph reachability problem on a certain model of computation known as NNJAG; see, e.g., [5]. Though NNJAG is a restrictive model, all the known algorithms for the directed reachability can be implemented in NNJAG without significant blow up in time and space.

Some important progress has been made for restricted graph classes. The most remarkable one is the log-space algorithm of Reingold (which we will refer as `UReach`) for the undirected graph reachability [12]. Recently, Asano and Doerr [2] gave a $\widetilde{O}(n^{1/2+\varepsilon})$-space and polynomial-time algorithm for the grid reachability. (In this paper "$\widetilde{O}(s(n))$-space" means $O(s(n))$-words intuitively and precisely $O(s(n)\log n)$-space.) Inspired by this result Imai et al. proposed [8] an $\widetilde{O}(n^{1/2+\varepsilon})$-space and polynomial-time algorithm for the planar graph reachability. In both algorithms, due to their recursive computation structure, the degree of their polynomial time bounds grow in proportion to $1/\varepsilon$, and it has been left open to design an $\widetilde{O}(n^{1/2})$-space and yet polynomial-time algorithm. More recently, Asano and Kirkpatrick [3] introduced a more efficient way to control the recursion, thereby succeeding to obtain an $\widetilde{O}(\sqrt{n})$-space algorithm with some polynomial time complexity. The main result of this paper is to show that this technique also works to design an $\widetilde{O}(\sqrt{n})$-space and polynomial-time algorithm for the planar reachability.

Although the above two $\widetilde{O}(n^{1/2+\varepsilon})$-space algorithms have similar space and time complexity bounds, their algorithmic approaches differ in some important aspects. The key idea of the $\widetilde{O}(n^{1/2+\varepsilon})$-space algorithm of Imai et al. is to use an algorithmic version of the Planar Separator Theorem shown first by Lipton and Tarjan [10]. Specifically, given any n-vertex undirected planar graph, there is a polynomial-time algorithm for computing an $O(\sqrt{n})$-size "separator", i.e. a set of $O(\sqrt{n})$ vertices whose removal separates the graph into two subgraphs of similar size. For a given input instance (G, v_0, v_*), we first compute a separator S of the underlying planar graph of G that separates G into two smaller subgraphs G^0 and G^1. We then consider a new directed graph H on $S \cup \{v_0, v_*\}$; it has a directed edge (a, b) if and only if there is a path from a to b in either G^0 or G^1. Clearly, reachability (of v_* from v_0) in H is equivalent to the original reachability. On the other hand, since S has $O(\sqrt{n})$ vertices, we can use the standard linear-space and polynomial-time algorithm for solving the reachability in H by using our algorithm recursively on G^0 and G^1 whenever we need to know whether an edge (a, b) exists in H. It should be mentioned that the idea of using separators to improve algorithms for the reachability and related problems is natural, and in fact it has been proposed by several researchers; see, e.g., [7]. The main contribution of [8] is to show that this idea indeed works by giving a space efficient separator algorithm based on the parallel separator algorithms of Miller [9] and Gazit and Miller [6].

The algorithm of Asano and Kirkpatrick uses a similar algorithmic approach. It uses a recursive separation of a grid graph; at each level a separator is formed by the set of vertices on one of the grid center lines. In order to get a polynomial time bound, Asano and Kirkpatrick introduce a "universal sequence" to control the time complexity of each recursive execution on smaller grids. Here we use the same idea for the general planar graph reachability. To achieve this we need a simple separator that allows us to express/identify a hierarchy of subgraphs succinctly/simply as was the case with grid graphs. The main technical contribution of this paper is to show a space efficient way to get such a simple separator and some succinct way to express separated subgraphs. (Note that it has been known that some separator algorithm, e.g., the one by Gazit and Miller [6], yields a simple cycle separator; but we are not sure whether the sublinear-space algorithm of [8] always yields such a simple separator. Here instead of analyzing the separator algorithm of [8], we show an algorithm to obtain cycle-separators from a given separator. Also our notion of a cycle-separator is slightly specific as explained later.)

It should be noted that, though restricted to grid graphs, the problem studied by Asano et al. in [2, 3] is the shortest path problem (a natural generalization of the reachability problem). The focus there is on space efficient and yet practically useful algorithms, including time-space

tradeoffs. In this paper, on the other hand, we are interested in extending a graph class that is solvable in $\widetilde{O}(\sqrt{n})$-space and polynomial-time, and the specific time complexity of algorithms is not so important so long as it is within some polynomial. In fact, since the algorithm of Reingold for the undirected reachability is used heavily, we need very large polynomial to bound our algorithm's running time. In order to keep our discussion as simple as possible, and focus on the key ideas, we restrict our attention here to the reachability problem. However, it is not hard to see that our algorithm for reachability can be modified to the shortest path problem (with a modest increase in the polynomial time bound).

Since we use Reingold's undirected reachability algorithm, our algorithm (and also the one by Imai et al.) have no natural implementation in the NNJAG model. While the worst-case instances for NNJAG given in [5] are non-planar, it is an interesting question whether we have similar worst-case instances based on some planar directed graphs. A more important and challenging question is to define some model in which our algorithm can be naturally implemented and show some limitation of space efficient computation.

Preliminaries

We use standard notions and notation for graphs. We specify a graph as a pair of its vertex set and edge set. For a graph G, by $\mathrm{V}(G)$ and $\mathrm{E}(G)$ we denote respectively the set of vertices of G and the set of edges of G. For any subset U of $\mathrm{V}(G)$, we use $G[U]$ to denote the subgraph of G induced by U; similarly, for any subset F of $\mathrm{E}(G)$, $G[F]$ denotes the subgraph of G induced by F. Like this notation, by notation using a symbol, e.g., G, we mean a graph defined from G. For two graph F and G, by $F \cup G$ we mean a graph $(\mathrm{V}(F) \cup \mathrm{V}(G), \mathrm{E}(F) \cup \mathrm{E}(G))$. For any graph G, consider any subgraph H of G and any vertex v of G that is not in H; then by $H \sqcup_G v$ we denote an induced subgraph of G obtained from H by adding vertex v; that is, $H \sqcup_G v = G[\mathrm{V}(H) \cup \{v\}]$. We also use $H \sqcup_G A$ to denote $G[\mathrm{V}(H) \cup A]$. Notions and notation for planar graphs will be explained in Section 3.

2 Planar Reachability Algorithm

We now describe our algorithm for planar reachability. To illustrate the idea of the algorithm we consider an instance (G, v_0, v_*) of gird reachability. That is, G is a graph obtained by removing some of the edges of a bi-directed grid graph like Figure 1(a). Here we assume that the original bi-directed grid graph is a $(2^h - 1) \times (2^h - 1)$ square grid. Let $`G$ denote the undirected version of this original grid; for the planar reachability this corresponds to the underlying undirected graph of an input directed graph. Note that both $G$ and $`G$ have $n = (2^h - 1) \times (2^h - 1)$ vertices.

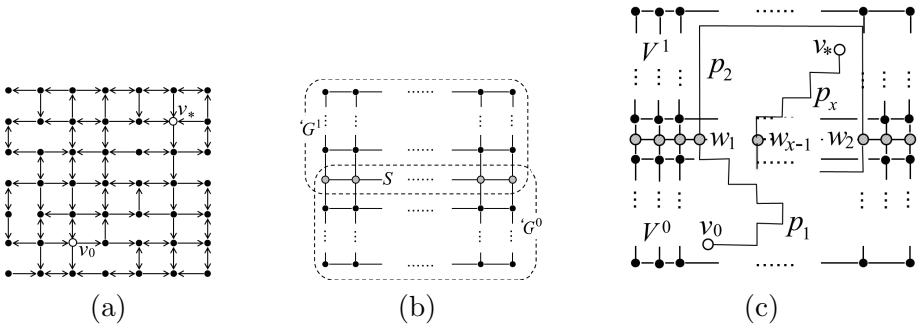


Figure 1: Example grid graph G and a path from v_0 to v_*

Though the reachability is determined on G, the computation is designed based on the undirected graph $`G$. Consider a set $S$ of vertices that are on the horizontal center line. We call $S$ a *grid-separator* because $`G$ is separated to two disconnected subgraphs by removing S. Our

strategy is to determine, for every vertex $v \in G$, the reachability from v_0 in each subgraph independently, thereby saving the space for the computation. More specifically, we consider a subgraph $`G^0$ (resp., $`G^1$) of $`G$ consisting of vertices below (resp., above) S including S (*cf.* Figure 1(b)), and compute the reachability from v_0 on G in the area defined by each subgraph. A crucial point is that we need to keep the reachability information only for vertices in S in order to pass the reachability information to the next computation on the opposite subarea. Also for showing the $\widetilde{O}(\sqrt{n})$-space bound, it is important that S consists of $2^h - 1 = \sqrt{n}$ vertices, and that both subgraphs $`G^0$ and $`G^1$ are almost half of $`G$.

Suppose that v_* is reachable from v_0 in G, and let p be a directed path witnessing this. We explain concretely our strategy to identify p and to confirm that v_* is reachable from v_0 in G. Notice here that path p is divided into some x subpaths $p_1, \ldots, p_x$ such that the following holds for each $j \in [x-1]$ (*cf.* Figure 1(c)): (i) the end vertex w_j of p_j (that is the start vertex of p_{j+1}) is on S, and (ii) all inner vertices of p_j are in the same V^b for some $b \in \{0, 1\}$, where V^0 and V^1 are respectively the set of vertices below and above the separator S. Then it is easy to see that we can find that w_1 is reachable from v_0 by searching vertices in S that are reachable from v_0 in $G[V^0 \cup S]$. Next we can find that w_2 is reachable (from v_0) by searching vertices in S that are reachable in $G[V^1 \cup S]$ from some vertex in S for which we know already its reachability; in fact, by the reachability from w_1 we can confirm that w_2 is reachable from v_0. Similarly, the reachability of $w_3, \ldots, w_{x-1}$ is confirmed, and then by considering the subgraph $G[V^1 \cup S]$ we confirm that v_* is reachable from v_0 because it is reachable from w_{x-1}. This is our basic strategy. Note that the reachability in each subgraph can be checked recursively.

We need one more idea in order to achieve the polynomial-time computability. To control the computation time for checking the reachability in each subgraph, we bound the length of paths by using some appropriately chosen numbers. (In this paper by "length of a path" we mean the number of edges on the path.) Asano and Kirkpatrick [3] proposed to use numbers from a "universal sequences", designed for this purpose. The notion of a universal sequence can be found in, e.g., [11]. Here we consider the following version.

For any $s \geq 0$, the *universal sequence* σ_s *order* s is defined inductively by

$$\sigma_s \;=\; \begin{cases} \langle 1 \rangle & \text{if } s = 0, \text{ and} \\ \sigma_{i-1} \diamond \langle 2^i \rangle \diamond \sigma_{i-1} & \text{otherwise,} \end{cases}$$

where $\diamond$ signifies concatenation of sequences. By the definition, each element of the sequence is a power of 2. The length of the sequence σ_s is $2^{s+1} - 1$. For example, $\sigma_2 = \langle 1, 2, 1, 4, 1, 2, 1 \rangle$. We will use the following properties of universal sequences in the design and analysis of our algorithm; for the sake of completeness, we also give a brief proof outline from [3].

Lemma 1. *(a) The sequence* $\sigma_s = \langle c_1, \ldots, c_{2^{s+1}-1} \rangle$ *is* 2^s-universal *in the sense that for any positive integer sequence* $\langle d_1, \ldots, d_x \rangle$ *such that* $\sum_{i \in [x]} d_i \leq 2^s$*, there exists a subsequence* $\langle c_{i_1}, \ldots, c_{i_x} \rangle$ *of* σ_s *such that* $d_j \leq c_{i_j}$ *holds for all* $j \in [x]$*; (b) the sequence* σ_s *contains exactly* 2^{s-i} *appearances of the integer* 2^i *(and nothing else); and (c) the sequence* σ_s *is computable in* $O(2^s)$*-time and* $\widetilde{O}(1)$*-space.*

Proof. (a) Let m be such an index that $d_1 + \cdots + d_m > d_{m+1} + \cdots + d_k$ and $d_1 + \cdots + d_{m-1} \leq d_{m+1} + \cdots + d_k$. Then, we have $d + \ldots + d_{m-1} \leq 2^{s-1}$ and $d_{m+1} + \cdots + d_k < 2^{s-1}$. Thus, by induction $\langle d_1, \ldots, d_{m-1} \rangle$ is dominated by a subsequence of σ_{s-1} and so is $\langle d_{m+1}, \ldots, d_k \rangle$. The result follows since $d_m \leq 2^s$.
(b) Straightforward, by induction on s.
(c) The i-th element of the universal sequence σ_s is given by 2^k where 2^k is the largest power of 2 which divides i. This computation is done by a naive algorithm using simple only $O(1)$-words. Although it takes $O(\log i)$ time to produce the i-th element in general, a simple analysis shows that the total time required is is $O(2^s)$. □

Now we define our reachability algorithm following the strategy explained above. The technical key point here is to define a sequence of separators dividing subgraphs into two parts in a way that we can specify a current target subgraph succinctly. For this we introduce the notion of "cycle-separator." Intuitively, a cycle-separator S of a graph G is a set of cycles $S = \{C_1, \ldots, C_h\}$ that separates G into two subgraphs, those consisting of vertices located left (resp., right) of the cycles (including cycle vertices). We can show that such a simple cycle-separator can be efficiently computed from any given separator. Based on this we have the following lemma.

Lemma 2. *There exists an $\widetilde{O}(\sqrt{n})$-space and polynomial-time algorithm (which we refer as* `CycleSep`*) that computes a cycle-separator S for a given undirected graph $G = (V, E)$ with its triangulated planar embedding. The size of the separator is at most $c_{\mathrm{sep}}\sqrt{n}$. Furthermore, there is a way to define subsets V^0 and V^1 of V with the following properties: (a) $V^0 \cup V^1 = V$, $V^0 \cap V^1 = S$, and (b) $|V^b| \leq (2/3)|V| + c_{\mathrm{sep}}\sqrt{n}$ for each $b \in \{0, 1\}$.*

Intuitively we can use cycle-separators like grid-separators to define a sequence of progressively smaller subgraphs of a given planar directed graph G. (Note that its underlying graph $\text{`}G$ is used for defining the subgraphs.) From technical reason[1], however, we need to add some edges to the outer faces of $\text{`}G[V^b]$ to get it triangulated under the current embedding. We will show (see Lemma 8) that the number of added edges is bounded by $O(|S|)$ and that there is an algorithm `AddTri` that computes these edges and their planar embedding in $\widetilde{O}(|S|)$-space and polynomial-time. We refer this information as an *additional triangulation edge list* T and consider it with a cycle-separator S and a Boolean label b. By $[\text{`}G]^b_{S,T}$ we mean both a graph obtained from $\text{`}G[V^b]$ by adding those triangulation edges specified by T and its planar embedding obtained by modifying the original planar embedding by T. In general, for any sequence $\mathbf{S} = \langle (b_1, S_1, T_1), \ldots, (b_t, S_t, T_t) \rangle$ of such triples of a label, a cycle-separator and an edge list, we define $[\text{`}G]_{\mathbf{S}}$ by

$$[\text{`}G]_{\mathbf{S}} \;=\; \Bigg[ \cdots \Big[ [\text{`}G]^{b_1}_{S_1,T_1} \Big]^{b_2}_{S_2,T_2} \cdots \Bigg]^{b_t}_{S_t,T_t},$$

which we call a *depth t subarea* of G. We should note here that one can efficiently identify a depth t subarea by using $\mathbf{S}$; that is, there is an algorithm that, for given $\mathbf{S}$ and $v \in V$, determines whether v is in the subarea $[\text{`}G]_{\mathbf{S}}$ by using only $O(\log n)$-space.

Armed with this method of constructing/specifying subareas we now implement our algorithm idea discussed above as a recursive procedure `ExtendReach` (see Algorithm 1). First we explain what it computes. We use global variables to keep the input graph G, the triangulated planar embedding of its underlying graph $\text{`}G$, the start vertex v_0, and the goal vertex v_*. As we will see in the next section, the triangulated planar embedding is $O(\log n)$-space computable. Hence, we can compute it whenever needed; thus, for simplicity we assume here that the embedding is given also as a part of the input. Note that for the space complexity these input data is not counted. On the other hand, we define global variables `A` and `R` that are kept in the work space. The variable `A` is for keeping separator vertices that are currently considered; the vertices v_0 and v_* are also kept in `A`. The array `R` captures the reachability information for vertices in `A`; for any $v \in$ `A`, `R`$[v] =$ **true** iff the reachability of v from v_0 has been cofirmed. Besides these data in the global variables, the procedure takes arguments $\mathbf{S}$ and ℓ, where $\mathbf{S}$ specifies the current subarea of $\text{`}G$ and $\ell$ is a bound on the length of path extensions. Our task is to update the reachability from $v_0$ for all vertices in `A` by using paths of length $\leq \ell$ in the current subarea. More precisely, the procedure `ExtendReach`$(\mathbf{S}, \ell)$ does the following: for each vertex $v \in$ `A`, it sets `R`$[v] =$ `true` if and only if there is a path to $v$ in $G[$`A`$\cup V_{\mathbf{S}}]$ of length $\leq \ell$ from some vertex $u \in$ `A` whose value `R`$[u]$ before the execution equals `true`, where $V_{\mathbf{S}}$ is the set of vertices of the current subarea of $\text{`}G$

[1]The algorithm `CycleSep` is defined based on the separator algorithm of Lemma 4 that assumes a triangulated graph as input. Thus, in order to apply `CycleSep` to divide $\text{`}G[V^b]$ further, we need to get it triangulated.

specified by $\mathbf{S}$. Since any vertex in G that is reachable from v_0 is reachable by a path of length at most $2^{\lceil \log n \rceil}$, the procedure `ExtendReach` can be used to determine the reachability of vertex v_* from v_0 as follows, which is our planar reachability algorithm: (1) Set $\mathtt{A} \leftarrow \{v_0, v_*\}$, $\mathtt{R}[v_0] \leftarrow$ **true**, and $\mathtt{R}[v_*] \leftarrow$ **false**; and (2) Execute `ExtendReach`$(\langle\,\rangle, 2^{\lceil \log n \rceil})$, and then output $\mathtt{R}[v_*]$.

Next we give some explanation on how to compute `ExtendReach` by going through the description of Algorithm 1. Consider any execution of `ExtendReach` for given arguments $\mathbf{S}$ and ℓ (together with data kept in its global variables). Let $V_{\mathbf{S}}$ be the set of vertices of the subarea $[\!`G]_{\mathbf{S}}$ specified by $\mathbf{S}$. There are two cases. If $V_{\mathbf{S}}$ has less than $144c_{\mathrm{sep}}^2$ vertices, then the procedure updates the value of `R` in a straightforward way. As we will see later $G[\mathtt{A} \cup V_{\mathbf{S}}]$ has at most $O(\sqrt{n})$ vertices; hence, we can use any standard linear-space and polynomial-time algorithm (e.g., Bellman-Ford algorithm) to do this task. Otherwise, `ExtendReach` divides the current subarea $[\!`G]_{\mathbf{S}}$ into two smaller subareas with new separator vertex set S'_{t+1} that is added to `A`. It then explore two subareas by using numbers in the universal sequence σ_s to control the length of paths in recursive calls.

Algorithm 1 `ExtendReach`$(\mathbf{S}, \ell)$

Given: (as arguments) A sequence $\mathbf{S} = \langle (b_1, S_1, T_1), \ldots, (b_t, S_t, T_t) \rangle$ of triples of a binary label, a cycle-separator, and an additional triangulation edge list, and a bound $\ell = 2^s$ on the length of path.
// In this description we use $V_{\mathbf{S}}$ to denote the set of vertices of $[\!`G]_{\mathbf{S}}$.
(as global variables) The input graph G, its triangulated planar embedding, the source vertex v_0, the goal vertex v_*, a set `A` of the currently considered vertices, and a Boolean array `R` specifying known reachability from v_0, for all $v \in \mathtt{A}$.

Task: For each vertex $v \in \mathtt{A}$, set $\mathtt{R}[v] =$ `true` if there is a path to v in $G[\mathtt{A} \cup V_{\mathbf{S}}]$ of length at most 2^s from some vertex $u \in \mathtt{A}$ whose value $\mathtt{R}[u]$ before the current procedure execution equals `true`.
// Invariant: $\mathtt{A} = \{v_0, v_*\} \cup \bigcup_{i \in [t]} S_i$. $\mathtt{R}[v] =$ **true** $\Rightarrow$ v is reachable from v_0 in G.

1: **if** the number of vertices of V_t is less than $144c_{\mathrm{sep}}^2$ **then**
2: $R_t \leftarrow \{u \in \mathtt{A} : \mathtt{R}[u] = \textbf{true}\}$;
3: **for** each vertex $v \in \mathtt{A}$ **do**
4: $\mathtt{R}[v] \leftarrow$ **true** iff v is reachable from some $u \in R_t$ in $G[\mathtt{A} \cup V_{\mathbf{S}}]$ by a path of length $\leq \ell$;
// Use any linear space and polynomial-time algorithm here.
5: **end for**
6: **else**
7: Use `CycleSep` and `AddTri` to create a new cycle separator S_{t+1} of $[\!`G]_{\mathbf{S}}$ and its additional triangulation edge lists T^0_{t+1} and T^1_{t+1};
8: $S'_{t+1} \leftarrow S_{t+1} \setminus \mathtt{A}$; $\mathtt{A} \leftarrow \mathtt{A} \cup S'_{t+1}$;
9: $\mathtt{R}[v] \leftarrow$ `false` for each vertex $v \in S'_{t+1}$;
10: **for** each $c_i = 2^{s_i}$ in the universal sequence σ_s (where $i \in [2^{s+1} - 1]$) **do**
11: `ExtendReach`$(\langle (b_1, S_1, T_1), \ldots, (b_2, S_t, T_t), (0, S_{t+1}, T^0_{t+1}) \rangle, c_i)$;
12: `ExtendReach`$(\langle (b_1, S_1, T_1), \ldots, (b_2, S_t, T_t), (1, S_{t+1}, T^1_{t+1}) \rangle, c_i)$;
13: **end for**
14: $\mathtt{A} \leftarrow \mathtt{A} \setminus S'_{t+1}$;
15: **end if**

The correctness of Algorithm 1 is demonstrated in Lemma 3 below. From this, as summarized in Theorem 1, it is clear that our algorithm correctly determines the reachability of vertex v_* from vertex v_0 in the input graph G.

Lemma 3. *For any input instance G, v_0, and v_* of the planar reachability, consider any execution of* `ExtendReach`$(\mathbf{S}, \ell)$ *for some $\mathbf{S} = \langle (b_1, S_1, T_1), \ldots, (b_t, S_t, T_t) \rangle$ and $\ell = 2^s$. Let $V_{\mathbf{S}}$ denote*

the set of vertices of $[\!`G]_{\mathbf{S}}$. *For each vertex* v *that is in* $\mathtt{A}$ *before the execution,* $\mathtt{R}[v]$ *is set to* **true** *during the execution if and only if there is a path to* v *in* $G[\mathtt{A} \cup V_{\mathbf{S}}]$ *of length at most* 2^s, *from some vertex* $u \in \mathtt{A}$ *whose value* $\mathtt{R}[u]$ *before the execution equals* `true`.

Proof. Suppose that there is a path $p = u_0, u_1, \ldots, u_h$ in $G[\mathtt{A} \cup V_{\mathbf{S}}]$, where (i) u_0 and u_h both belong to $\mathtt{A}$, (ii) $h \leq 2^s$, and (iii) $\mathtt{R}[u_0] = \mathtt{true}$ before executing the procedure. For the lemma, it suffices to show that $\mathtt{R}[u_h]$ is set **true** during the execution of the procedure.

We prove our assertion by induction on the size of $V_{\mathbf{S}}$. If $|V_{\mathbf{S}}| \leq 144c_{\mathrm{sep}}^2$, then it is clear from the description of the procedure. Consider the case where $|V_{\mathbf{S}}| > 144c_{\mathrm{sep}}^2$. Then the part from line 7 of the procedure is executed. Let S_{t+1}, T_{t+1}^0, and T_{t+1}^1 be the separator and the edge lists computed there. For any $b \in \{0, 1\}$, let $\mathbf{S}^b$ denote $\langle (b_1, S_1, T_1), \ldots, (b_t, S_t, T_t), (b, S_{t+1}, T_{t+1}^b) \rangle$; also let $`G^b = [\!`G]_{\mathbf{S}^b}$ $(= [[\!`G]_{\mathbf{S}}]_{(b, S_{t+1}, T_{t+1}^b)})$ and $V^b = \mathrm{V}(`G^b) \setminus S_{t+1}$.

We observe that p can be decomposed into some number $x \leq |\mathtt{A} \cup S_{t+1}| - 1$ of subpaths $p_1, p_2, \ldots, p_x$, such that (i) both $\mathrm{start}(p_j)$ and $\mathrm{end}(p_j)$ belong to $\mathtt{A} \cup S_{t+1}$ for each $j \in [x]$, (ii) the internal vertices of p_j belong either to V^0 or V^1 for each $j \in [x]$, and (iii) $\mathrm{end}(p_j) = \mathrm{start}(p_{j+1})$ for each $j \in [x-1]$, where by $\mathrm{start}(p_j)$ and $\mathrm{end}(p_j)$ we mean the start and end vertices of p_j respectively. Let h_j denote the number of edges in path p_j. By construction (i) $h_j \geq 1$ for all $j \in [x]$, and (ii) $\sum_{j \in [x]} h_j = h \leq 2^s$. Then by Lemma 1(a), the sequence $\langle h_1, \ldots, h_x \rangle$ is dominated by the universal sequence $\sigma_s = \langle c_1, \ldots, c_{2^{s+1}} \rangle$. That is, there exists a subsequence $\langle c_{k_1}, c_{k_2}, \ldots c_{k_x} \rangle$ of σ_s such that $h_j \leq c_{k_j}$ for all $j \in [x]$. Thus, for any $j \in [x]$, if $\mathtt{R}[\mathrm{start}(p_j)] = \mathbf{true}$ before the execution of $\mathtt{ExtendReach}(\mathbf{S}^0, c_{k_j})$, then we have $\mathtt{R}[\mathrm{end}(p_j)] = \mathbf{true}$ after the execution because of the induction hypothesis. Hence, by executing the fragment:

$\mathtt{ExtendReach}(\mathbf{S}^0, c_{k_1})$; $\mathtt{ExtendReach}(\mathbf{S}^1, c_{k_1})$; $\cdots$
$\cdots$ $\mathtt{ExtendReach}(\mathbf{S}^1, c_{k_{x-1}})$; $\mathtt{ExtendReach}(\mathbf{S}^0, c_{k_x})$; $\mathtt{ExtendReach}(\mathbf{S}^1, c_{k_x})$;

we have $\mathtt{R}[\mathrm{end}(p_x)] = \mathtt{true}$ since $\mathtt{R}[\mathrm{start}(p_1)] = \mathtt{true}$ by our assumption. Therefore, $\mathtt{R}[u_h]$ (where $u_h = \mathrm{end}(p_x)$) is set **true** as desired since the above fragment must be executed as a part of the execution of line 10–13 of the procedure. □

By analyzing the time and space complexity of our algorithm, we conclude as follows.

Theorem 1. *For any input instance* G, v_0, *and* v_* *of the planar directed graph reachability problem (where* n *is the number of vertices of* G*), our planar reachability algorithm determines whether there is a path from* v_0 *to* v_* *in* G *in* $\widetilde{O}(\sqrt{n})$ *space and polynomial-time.*

Proof. The correctness of the algorithm follows immediately from Lemma 3. For the complexity analysis, we consider the essential part, that is, the execution of $\mathtt{ExtendReach}(\langle\,\rangle, 2^{\lceil \log n \rceil})$.

First we bound the depth of recursion during the execution. Let $t_{\max}$ denote the maximum depth of recursive calls in the execution, for which we would like to show that $t_{\max} \leq 2.5 \log n$ holds. Consider any depth t recursive call of `ExtendReach`; in other words, the execution of $\mathtt{ExtendReach}(\mathbf{S}, \ell)$ with a sequence $\mathbf{S}$ of length t. (Thus, the inital call of `ExtendReach` is regarded as depth 0 recursive call.) Here some depth t subarea $[\!`G]_{\mathbf{S}}$ is examined; let $n_t$ be the number of vertices of this subarea. Assume that $n_t \geq 144c_{\mathrm{sep}}^2$. Then two smaller subareas of $[\!`G]_{\mathbf{S}}$ are created and `ExtendReach` is recursively executed on them. Let n_{t+1} denote the number of vertices of a lager one of these two smaller subareas. Then by Lemma 2 we have

$$n_{t+1} \;\leq\; \frac{2n_t}{3} + c_{\mathrm{sep}}\sqrt{n_t} \;\leq\; \frac{3n_t}{4}$$

since $n_t \geq 144c_{\mathrm{sep}}^2$. Hence, $t_{\max}$ is bounded by $2.5 \log n$ as desired because $n(3/4)^{2.5 \log n} < 144c_{\mathrm{sep}}^2$.

For bounding the memory space used in the execution $\mathtt{ExtendReach}(\langle\,\rangle, 2^{\lceil \log n \rceil})$, it is enough to bound the number of vertices in $\mathtt{A}$ because the number of words needed to keep in the work

memory space during the execution is proportional to $|\mathtt{A}|$. Note further that $\mathtt{A} = \{v_0, v_*\} \cup \bigcup_{i \in [t]} S_i$ at any depth t recursive call of `ExtendReach`. On the other hand, by using the above notation, it follows from the above and Lemma 2 we have

$$\begin{aligned} |\mathtt{A} \setminus \{v_0, v_*\}| &\leq \sum_{i \in [t_{\max}]} |S_i| \leq \sum_{i \in [t_{\max}]} c_{\mathrm{sep}} \sqrt{n_i} \\ &\leq \sum_{i \in [t_{\max}]} c_{\mathrm{sep}} \sqrt{\left(\frac{3}{4}\right)^{i-1} n} \leq \left(c_{\mathrm{sep}} \sqrt{n}\right) \left(\sum_{i \geq 0} \left(\frac{3}{4}\right)^{i/2} \right) = O(\sqrt{n}), \end{aligned}$$

which gives us the desired space bound.

For bounding the time complexity by some polynomial, it suffices to show that the total number of calls of `ExtendReach` is polynomially bounded. To see this, we estimate $N(t, 2^s)$, the max. number of calls of `ExtendReach` during any depth t recursive call of `ExtendReach`$(\mathbf{S}, 2^s)$ that occurs in the execution of `ExtendReach`$(\langle\,\rangle, 2^{\lceil \log n \rceil})$. (Precisely speaking, $N(t, 2^s) = 0$ if no call of type `ExtendReach`$(\mathbf{S}, 2^s)$ occurs.) Clearly, $N(t_{\max}, 2^s) = 0$ for any s. Also it is easy to see that $N(t, 2^0) = 2 + 2N(t+1, 2^0)$ for any $t < t_{\max}$; hence, we have $N(t, 2^0) \leq 2 \cdot (2^{t_{\max} - t} - 1) \leq 2^{t_{\max} - t + 1}$. Consider any $t < t_{\max}$ and $s \geq 1$. From the description of `ExtendReach` and the property of the universal sequence σ_s (Lemma 1(b)), we have

$$N(t, 2^s) = 2 \sum_{i \in [2^{s+1}]} (1 + N(t+1, c_i)) = 2^{s+2} + \sum_{0 \leq j \leq s} 2^{s-j} N(t+1, 2^j),$$

from which we can derive $N(t, 2^s) = 2N(t+1, 2^s) + 2N(t, 2^{s-1})$. Then by induction we can show

$$N(t, 2^s) \leq 2^{t_{\max} - t + s + 1} \binom{t_{\max} - t + s}{s}.$$

Thus, $N(0, 2^{\lceil \log n \rceil})$, the total number of calls of `ExtendReach` is polynomially bounded. □

3 Cycle-separators

We define the notion of a cycle-separator. Throughout this section, we consider only undirected graphs. In particular, we fix any sufficiently large planar undirected graph $G = (V, E)$ and discuss a method for defining/constructing a cycle-separator for G; all symbols containing G (resp., V and E) are related to G (resp., V and E).

We begin with some basic notions and facts on planar graphs. A graph is *planar* if it can be drawn on a plane so that the edges intersect only at end vertices. Such a drawing is called a *planar embedding*. Here we use the standard way to specify a *planar embedding* of G; that is, a sequence of vertices adjacent to v in a clockwise order around v under the planar embedding, for all $v \in V$. We use $\mathrm{N}(v)$ to denote this sequence for v, which is often regarded as a set. We say that a planar graph is *triangulated* if addition of any edge results in a nonplanar graph. For a planar graph and its planar embedding, its *triangulation* (w.r.t. this planar embedding) means to add edges to the planar graph until all its faces under the planar embedding (including the outer one) are bounded by three edges. Allender and Mahajan [1] reduced the problem of computing a planar embedding to the undirected reachability. Hence, by using the algorithm `UReach` of Reingold, we can compute a planar embedding of G by using $O(\log n)$-space. Once some planar embedding is computed, it is also easy to obtain some triangulation w.r.t. this embedding. For example, we can compute as follows: (i) identify all cycles (with some direction) with no edge in its left or right side, and then (ii) triangulate each cycle by adding edges connecting all vertices of the cycle to the vertex with the smallest index in the cycle. Altogether we have

an $O(\log n)$-space algorithm that computes a triangulated planar embedding for a give planar graph. In our reachability algorithm, the above $O(\log n)$-space algorithm is used (implicitly) before starting the actual computation to obtain some triangulated planar embedding for the underlying graph of a given planar directed graph; after this, the algorithm keeps necessary information to maintain some triangulated planar embedding for the current graph. Thus, we may assume in the following that our target graph G is given with some triangulated planar embedding.

3.1 Definition and construction of a cycle-separator

The Planar Separator Theorem guarantees that every planar graph has a separator of size $O(\sqrt{n})$ that disconnects a graph into two subgraphs each of which has at most $2n/3$ vertices, which we call a 2/3-*separator*. Several efficient separator algorithms have been proposed, and based on separator algorithms of Miller [9] and Gazit and Miller [6], Imai et al. [8] has shown $O(\sqrt{n})$-space and polynomial-time separator algorithm that yields a 2/3-*separator*. We use such a separator algorithm as a black box in this paper. On the other hand, we introduce here a way to specify *two* subgraphs disconnected by a separator in order to use them in the context of sublinear-space computation.

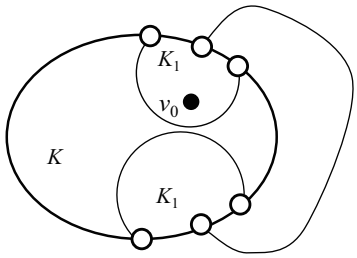
A *labeled-separator* of G is a pair of a separator S and a set $\tau = \{v_1, \ldots, v_k\}$ of vertices of G (which we simply denote by $S\tau$) such that no two vertices of τ belong to the same connected component of $G[V \setminus S]$. Graphs $G^0_{S\tau}$ and $G^1_{S\tau}$ are two disconnected subgraphs of $G[V \setminus S]$ defined by $S\tau$; $G^0_{S\tau} = \bigcup_{i=1}^{k} K_i$ where each K_i is the connected component of $G[V \setminus S]$ containing v_i, and $G^1_{S\tau}$ is a subgraph of G consisting of all the other connected components of $G[V \setminus S]$.

By the planarity, we can show that $G[V \setminus S]$ has at most $2|S| - 4$ connected components. (Recall that we assumed G is triangulated and hence connected.) Thus, each labeled-separator can be stored in $\widetilde{O}(|S|)$-space. Furthermore, by using `UReach`, we can identify, for each $v_i \in \tau$, the connected component K_i containing v_i in $O(\log n)$-space. Since counting is also possible in $O(\log n)$-space, for a given 2/3-separator, we can in fact collect connected components $K_1, \ldots, K_k$ of $G[V \setminus S]$ (and their representative vertices $v_1, \ldots, v_k$) so that $|\mathrm{V}(G^0_{S\tau})| \leq 2|V|/3$ and $|\mathrm{V}(G^1_{S\tau})| \leq 2|V|/3$ hold with $\tau = \{v_1, \ldots, v_k\}$. The following lemma summarize these observations.

Lemma 4. *There exists an $\widetilde{O}(\sqrt{n})$-space and polynomial-time algorithm that yields a 2/3-labeled-separator of size $\leq c_{\mathrm{sep}}\sqrt{n}$ for a given triangulated planar graph, where c_{sep} is some constant, which has been used in the previous section.*

For using separators in our reachability algorithm, we need simple separators that can be used like grid-separators for grid graphs. For this we introduce the notion of a cycle-separator and show an efficient way to transform a given separator to a cycle-separator.

We first explain why we need this new separator notion. For using the above lemma, we need to have a graph triangulated. Although we may assume that each connected component is triangulated, some edges need to be added to triangulate its outer face; see, e.g., Figure 2. Then some of the connected components created afterwards may use those added triangulation edges, and in order to identify such connected components, we need to keep separator vertices and the connectivity information around them. While we can identify $G^0_{S\tau}$ and $G^1_{S\tau}$ with S and τ, what we actually need is a way to identify $G^0_{S\tau} \sqcup_G S$ and $G^1_{S\tau} \sqcup_G S$ with additional triangulation edges, and for this we need the information of edges incident to vertices in S that should be cut to define, e.g., $G^0_{S\tau} \sqcup_G S$. This is easy for the grid case; for example, 'G^0 of Figure 1(b) is obtained by simply cutting edges "above" the separator vertices of S. Unfortunately, on the other hand, the number of such cut edges may become large to keep in the general planar graph case, and there is no trivial way to compute them in polynomial-time on-the-fly in our recursive



K is one of the connected components separated from the shaded part by separator vertices indicated by small circles. After adding some edges in the outer face of K, K (with the triangulation edges) is divided further (by the separator algorithm), and this yields a connected component K_1 and its label vertex v_0. Note that the triangulation edge added to the separator vertices is used in the connected component K_1; hence, the connectivity around separator vertices is crucial for defining connected components.

Figure 2: An example that a labeled separator does not work

procedure. Thus, we introduce the notion of a cycle-separator, which may be regarded as a generalization of the grid-separator.

Recall that we assume some planar embedding of G; the following notions are defined with respect to this embedding. For any cycle C of G, we use a sequence $\langle u_1, \ldots, u_r \rangle$ of vertices of G in the order of appearing in C under one direction. We call such a sequence as a *cycle representation*. In the following, we identify a cycle and its cycle representation; furthermore, a cycle may be treated as a vertex set or an edge set. For any cycle representation C, let $G_{\mathrm{in}}[C]$ (resp., $G_{\mathrm{out}}[C]$) denote the subgraph of G consisting of vertices of G located in the left (resp., right) of the cycle following the cycle representation. We note here that $G_{\mathrm{in}}[C]$ and $G_{\mathrm{out}}[C]$ are disconnected in $G[V \setminus C]$. Thus, each cycle can be used as a separator. Intuitively, a cycle-separator is a separator that is specified by a set of cycle representations; our notion of cycle-separator is more restrictive, which will be defined later. The following lemma shows that one can find some separator S' consisting of cycles as a subset of a given separator S. Its proof will be given in Subsection 3.3.

Lemma 5. *Consider any labeled-separator $S\tau$ of G with $\tau = \{v_1, \ldots, v_k\}$, and let $K_1, \ldots, K_k$ be its associated connected components. Then we can define a labeled-separator $S'\tau'$ of G with $\tau' = \{v'_1, \ldots, v'_{k'}\}$ and a set $\{C_1, \ldots, C_h\}$ of cycle representations satisfying the following properties for the set $\{K'_1, \ldots, K'_{k'}\}$ of connected components associated with $S'\tau'$:*

(1) *$S' \subseteq S$, $\mathrm{V}(G^b_{S'\tau'}) \subseteq \mathrm{V}(G^b_{S\tau}) \cup S$ for any $b \in \{0,1\}$, and $G^0_{S'\tau'} = \bigcup_{i\in[k']} K'_i$;*

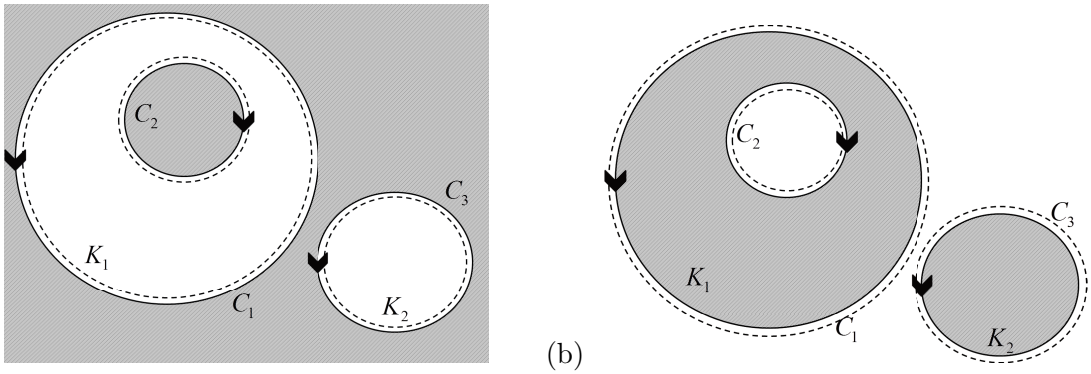
(2) *$\bigcup_{i\in[h]} G[C_i] = G[S']$ (that is, $\bigcup_{i\in[h]} C_i = S'$);*

(3) *every two cycles C_i and C_j, $i \neq j$, are edge disjoint; and*

(4) *for each $i \in [k']$, there exist some t and some $C_{j_1}, \ldots, C_{j_t}$ such that $K'_i = \bigcap_{r\in[t]} G_{\mathrm{in}}[C_{j_r}]$ holds.*

Using the notation above. From the property (4) of the lemma, each K'_i is expressed as $\bigcap_{r=1}^{t} G_{\mathrm{in}}[C_{j_r}]$ by using some cycles $C_{j_1}, \ldots, C_{j_t}$ of S'. In fact, we show here a way to express $G^0_{S'\tau'}$ and $G^1_{S'\tau'}$ by using the cycles $\{C_1, \ldots, C_h\}$ of S'. Consider any cycle C of S'; recall that it is a cycle representation with a certain orientation. Let $E^1(C)$ (resp., $E^0(C)$) denote *intuitively* the set of all edges of G that are incident from some vertex of C from its right (resp., left); let $E^1(S') = \bigcup_{C\in S'} E^1(C)$ and $E^0(S') = \bigcup_{C\in S'} E^1(C)$. Then for each $b \in \{0,1\}$, we define $V^b_{S'}$ by

$$V^b_{S'} \;=\; \{\, v \in V \;:\; v \text{ is connected to } S' \text{ by a path with no edge from } E^{1-b}(S') \,\}.$$



S' consists of three cycles C_1, C_2, C_3, each of which has an orientation specified as in the figure. Dashed lines indicate the cuts corresponding to edges in (a) $E^1(S')$ and (b) $E^0(S')$ respectively. Then shadow parts are respectively (a) $G[V^0_{S'}]$ and (b) $G[V^1_{S'}]$.

Figure 3: (a) $G[V^0_{S'}]$ and (b) $G[V^1_{S'}]$

Then $G[V^b_{S'}]$ is a subgraph of G consisting of vertices that are connected to some vertex of S' in G after cutting all edges in $E^{1-b}(S')$; see Figure 3. We show below (i.e., Lemma 2) that $G[V^b_{S'}]$ indeed represents $G^b_{S'\tau'} \sqcup_G S'$.

Now we define $E^1(S')$ and $E^0(S')$ formally. Consider any vertex $u \in S'$. It should be on some cycle of S', and let (v', u) and (u, v) denote two edges adjacent to u in one of such cycles. Let w_1 be the right triangle adjacent vertex of (u, v), and let $w_1, \ldots, w_i, w_{i+1}$ be a clockwise ordered sequence of $\mathrm{N}(v)$ starting from w_1 such that w_{i+1} is the first vertex in S'. Usually w_{i+1} is v', but it could be a vertex on some other cycle of S' (when these cycles share vertex u). We then define $E^1_{(u,v)}$ by

$$E^1_{(u,v)} \;=\; \{\,\{u, w_1\}, \ldots, \{u, w_i\}\,\},$$

and define $E^1(S') = \cup_{(u,v)\in S'} E^1_{(u,v)}$. We define $E^0(S')$ similarly by using the reverse order (i.e., the counterclockwise order) of the planar embedding. Note that $V^b_{S'}(G)$ is defined as above with these formally defined $E^b(S')$'s.

The following lemma shows that this definition of $E^1(S')$ and $E^0(S')$ is consistent with our intuition.

Lemma 6. *For each $b \in \{0, 1\}$, $E^b(S')$ is the set of all edges of G that has one end point in S' and the other end point in $G^b_{S'\tau'}$.*

Proof. We show the lemma only for $E^1(S')$; the lemma for $E^0(S')$ is proved similarly. Let $\widehat{E}$ denote the set of edges of G that has one end point in S' and the other end point in $G^1_{S'\tau'}$. For each cycle edge (u, v) of S', it is easy to see that $E^1_{(u,v)} = \{\{u, w_1\}, \ldots, \{u, w_i\}\}$ is a subset of $\widehat{E}$ because w_1, which is the right adjacent vertex of (u, v), is in $G^1_{S'\tau'}$ and other $w_2, \ldots, w_i$, which are all connected to w_1 by some path not crossing any cycle of S', are in $G^1_{S'\tau'}$. Hence, $E^1(S') \subseteq \widehat{E}$.

Next we show that $\widehat{E} \subseteq E^1(S')$. For this, consider any edge $\{x, y\} \in \widehat{E}$. We may assume that $x \in S'$ and $y \in G^1_{S'\tau'}$. Note that $y \in \mathrm{N}(x)$. We consider the reverse order (i.e., the counterclockwise order) of the planar embedding of $\mathrm{N}(x)$ from y; let z be the first vertex that is in S', and w is the one that is just before z under this ordering. Note that (z, x) must be an edge of some cycle of S' so that w, which is also in $G^1_{S'\tau'}$, becomes the right adjacent vertex

of (z,x). Then it is easy to see that $\{x,y\}$ appears in $E^1_{(z,x)} \subseteq E^1(S')$. Thus, $\widehat{E} \subseteq E^1(S')$ as desired. □ □

Lemma 7. *For each $b \in \{0,1\}$, we have $\mathrm{V}(G^b_{S'\tau'}) \cup S' = V^b_{S'}$.*

Proof. First note that $V^0_{S'} \cup V^1_{S'} = V$ and that $V^0_{S'} \cap V^1_{S'} = S'$; the latter holds because removing $E^0(S') \cup E^1(S')$ disconnects S' from the other vertices of G. Thus, for the proof, it suffices to show that $\mathrm{V}(G^0_{S'\tau'}) \subseteq V^0_{S'}$ and $\mathrm{V}(G^1_{S'\tau'}) \subseteq V^1_{S'}$.

Consider any vertex v of $G^0_{S'\tau'}$ and any vertex u in S'. Since G is connected, there is some path p from v to u. We trace this path from v to u, and let x be the first vertex on this path that is in S' and let y be the one before x on this path. Clearly, $y \in G^0_{S'\tau'}$; hence, from the above lemma, $\{x,y\}$ is an edge in $E^0(S')$, and it is not in $E^1(S')$. Also the above lemma guarantees that no other edges $\{x,y\}$ on the subpath of p from v to x belongs to $E^1(S')$. Thus, v is connected to $x \in S'$ even if we remove all edges in $E^1(S')$; that is, $v \in V^0_{S'}$. This proves that $\mathrm{V}(G^0_{S'\tau'}) \subseteq V^0_{S'}$. Similarly we can show that $\mathrm{V}(G^1_{S'\tau'}) \subseteq V^1_{S'}$. □ □

Let us summarize what we have discussed. For a given labeled-separator $S\tau$, we showed a way to define a subset S' of S that can be used as a labeled-separator with some $\tau' \subseteq \tau$ and that is represented by a set $\{C_1,\ldots,C_h\}$ of cycle representations. Furthermore, we can use these cycle representations to determine $G^0_{S'\tau'}$ and $G^1_{S'\tau'}$ without using the label τ'; that is, $G[V^b_{S'}] = G^b_{S'\tau'} \sqcup_G S'$ holds for each $b \in \{0,1\}$. Also from our discussion above, it is easy to see that there is an $O(\log n)$-space algorithm that determines whether $v \in V^b_{S'}$ for given $b \in \{0,1\}$, S, and $v \in V$. In this paper, by a *cycle-separator* we mean a set S' of cycle representations that is obtained from some labeled-separator in the way we explained. Now Lemma 2 is immediate. By using the algorithm given in Lemma 4, we can compute a 2/3-labeled-separator S of size $\leq c_{\mathrm{sep}}\sqrt{n}$ from which we can compute a cycle separator S' such that $|V^b_{S'}| \leq |\mathrm{V}(G^b_{S\tau})| + |S| \leq (2/3)|V| + c_{\mathrm{sep}}\sqrt{n}$ holds (Lemma 5(1)). The computation can be done using $O(\sqrt{n})$-space and in polynomial-time.

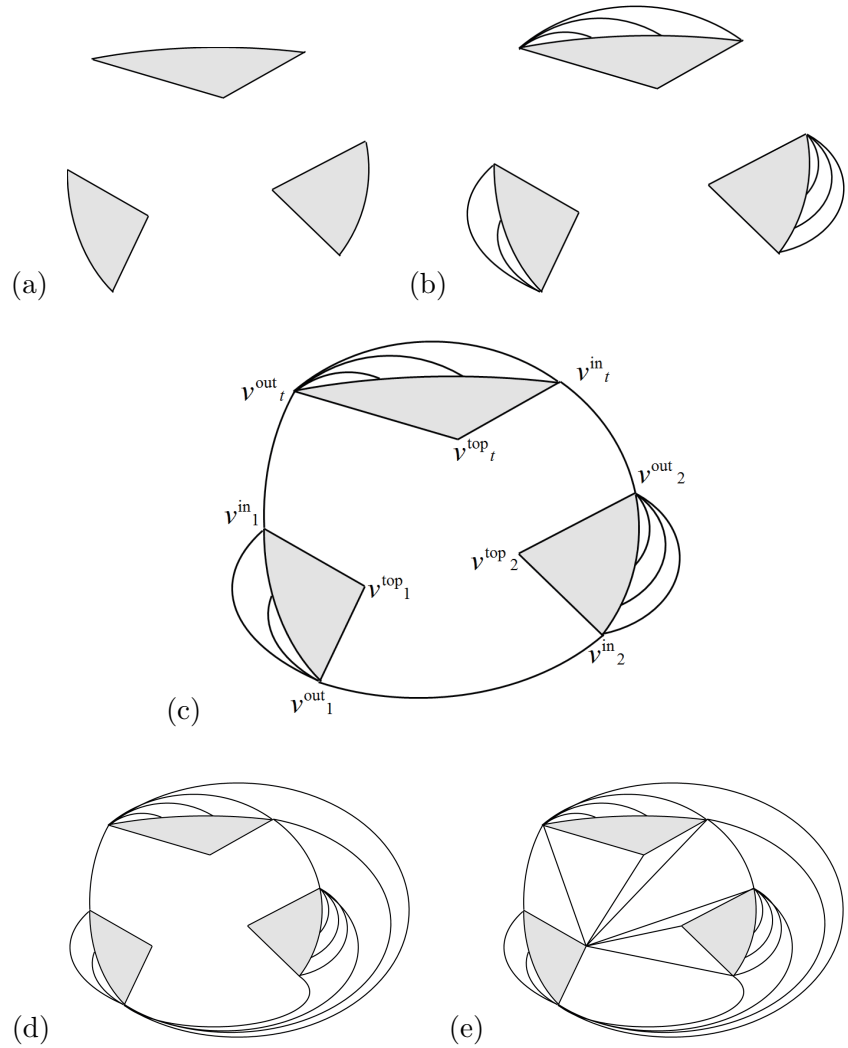
3.2 Depth t subarea

In this subsection we use S to denote a cycle-separator. Intuitively, we can use $G[V^0_S]$ and $G[V^1_S]$ to determine subareas. But in order to compute the reachability in each subarea recursively, these subareas also need to be triangulated. We may assume that G is triangulated, and hence the inside of each $G[V^b_S]$ is triangulated; but we need to triangulate its outer face. The following lemma gives a way to do this triangulation.

Lemma 8. *There exists a polynomial-time algorithm (which we refer as* `AddTri`*) that, for given undirected graph G with its planar embedding, its cycle-separator S, and $b \in \{0,1\}$, computes edges added to triangulate $G[V^b_S]$ together with their planar embedding by using $\widetilde{O}(|S|)$-space.*
Remark. *Recall that a planar embedding of G is specified as a sequence $\mathrm{N}(v)$ of v's neighbors for all vertices v. What the algorithm actually produces is a set T of revised $\mathrm{N}(v)$ for all vertices $v \in S$, which we call* additional triangulation edge list. *Note that this edge list is expressed in $\widetilde{O}(|S|)$-space. Let $[G]^b_{S,T}$ denote a graph obtained from $G[V^b_S]$ by adding triangulation edges specified by T.*

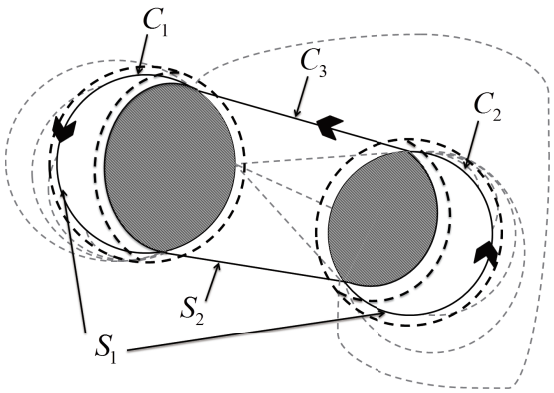
Proof. We explain a way that the algorithm `AddTri` adds new edges to vertices in S. The way to modify $\mathrm{N}(v)$ follows easily, and we omit explaining it. In the algorithm, we need to identify which vertex (resp., edge) of G belongs to $G[V^b_S]$; we provide $O(\log n)$-space algorithms for these tasks in Lemma 9.

The algorithm adds edges for every "face" of $G[V^b_S]$, namely, a connected component of G that is separated from $G[V^b_S]$ by cutting edges in $E^{1-b}(S)$. Consider one of such faces, and let



This figure shows how edges are added at each step of the triangulation algorithm explained in Lemma 8. In this example, the situation where $G[V_S^b]$ has a face with three cycles as (a). The other side of each cycle (i.e., the shadow parts) is a part of $G[V_S^b]$.

Figure 4: Each step of the triangulation algorithm



This figure illustrates the idea of checking whether a given vertex v exists in $[G]^{01}_{(S_1,T_1),(S_2,T_2)}$. K_1 and K_2 are connected components of $G^0_{(S_1,T_1)}$ that are surrounded by cycles C_1 and C_2 respectively. On the other hand, C_3 is a cycle-separator of S_2 that defines $([G]^0_{(S_1,T_1)})^1_{(S_2,T_2)}$ (the shaded parts). We cut edges along dashed lines and check whether v is reachable to S_2 $(= C_3)$.

Figure 5: The idea of testing whether v exists in $[G]^{01}_{(S_1,T_1),(S_2,T_2)}$

F denote it. The algorithm identifies all cycles, say, $C_1, \ldots, C_r$, of S facing to F (Figure 4(a)); that is, these are cycles incident to F with edges in $E^{1-b}(S)$. Note that the order of these cycles is not essential because there is no edge among them in $G[V_S^b]$. Once these cycles are identified, the algorithm ignore vertices and edges not in $G[V_S^b]$.

The algorithm transforms each of these cycles to a triangle as Figure 4(b); that is, for each C_i, the algorithm identifies its vertex v with the smallest index, and connect the other vertices of C_i to v except for two vertices adjacent to v in C_i. Then we obtain triangulated cycles, which we still refer as $C_1, \ldots, C_r$.

Consider now triangle C_1. We assume (by reversing the order if necessary) that the face is on the left side of C_1 under the cycle representation of its three vertices, and let $\langle u, v, w\rangle$ be this cycle representation. We name u, v, and w as v_1^{out}, v_1^{in}, and v_1^{top} respectively. Similarly, we give names to three vertices of the other cycles. Then the algorithm adds an edge connecting v_1^{out} and v_2^{in}, v_2^{out} and v_3^{in}, …, v_t^{out} and v_1^{in} (Figure 4(c)).

We now have two faces, one surrounded by cycle $C_{\mathrm{out}} := (v_1^{\mathrm{out}}, v_2^{\mathrm{in}}, v_2^{\mathrm{out}}, \ldots, v_t^{\mathrm{in}}, v_t^{\mathrm{out}}, v_1^{\mathrm{in}})$ and one surrounded by cycle $C_{\mathrm{in}} := (v_1^{\mathrm{out}}, v_2^{\mathrm{in}}, v_2^{\mathrm{top}}, v_2^{\mathrm{out}}, \ldots, v_t^{\mathrm{top}}, v_t^{\mathrm{out}}, v_1^{\mathrm{in}}, v_1^{\mathrm{top}})$. Remaining work is similar to the previous step. For the former face, edges are added to connect v_1^{out} and all other vertices of C_{out}; Figure 4(d)). For the latter face, edges are added to connect v_1^{out} and all other vertices of C_{in}; Figure 4(e)).

The algorithm considers only edges and vertices of S. Therefore, $\widetilde{O}(|S|)$-space is sufficient for running this algorithm. □ □

Finally we explain a way to identify a depth t subarea of G. Recall that it is specified by a sequence $\mathbf{S} = \langle (b_1, S_1, T_1), \ldots, (b_t, S_t, T_t)\rangle$ as follows:

$$[G]_{\mathbf{S}} \;=\; \Big[\cdots\Big[[G]^{b_1}_{S_1,T_1}\Big]^{b_2}_{S_2,T_2}\cdots\Big]^{b_t}_{S_t,T_t},$$

The task of identifying this subarea is to determine a given vertex (resp., an edge) is in the graph $[G]_{\mathbf{S}}$. We show a $O(\log n)$-space algorithm for these tasks.

Lemma 9. *There exist $O(\log n)$-space algorithms that recognize respectively vertices and edges of $[G]_{\mathbf{S}}$ that is specified by $\mathbf{S} = \langle (b_1, S_1, T_1), \ldots, (b_t, S_t, T_t)\rangle$.*

Proof. For any $k \in [t]$, let $V_k = \mathrm{V}\big([G]_{\mathbf{S}_k}\big)$ and $E_k = \mathrm{E}\big([G]_{\mathbf{S}_k}\big)$, where $\mathbf{S}_k = \langle (b_1, S_1, T_1), \ldots, (b_k, S_k, T_k)\rangle$. Also define $\widetilde{E}_k$ and $\widetilde{T}_k$ by

$$\widetilde{E}_k \;=\; \bigcup_{i\in[k]} E^{1-b_i}(S_i), \quad \text{and} \quad \widetilde{T}_k \;=\; \bigcup_{i\in[k]} T_i.$$

Then for any $\{u, v\}$, it is an edge in E_t if and only if both u and v are vertices of V_t *and* it is an edge in $E \cup \widetilde{T}_t \setminus \widetilde{E}_t$. Hence, the task of checking whether a given edge is in $[G]_{\mathbf{S}}$ is reduced to that of checking its two end points are in $[G]_{\mathbf{S}}$. Thus, we only need to design an algorithm that checks whether a given $v \in V$ exists in $[G]_{\mathbf{S}}$.

The idea of the algorithm is illustrated in Figure 5. It checks whether v is reachable to some vertex in S_t by using only edges from $E \cup \widetilde{T}_t \setminus \widetilde{E}_t$ We can implement this check as some $O(\log n)$-space algorithm by using `UReach`. Note that whenever `UReach` asks whether an edge exists in $E \cup \widetilde{T}_t \setminus \widetilde{E}_t$, it can be answered in $O(\log n)$-space by using $\mathbf{S}$ given as input.

We show that this idea indeed works. More precisely, we prove

$$v \in V_t \;\Leftrightarrow\; v \text{ is connected to some vertex in } S_t \text{ in } G[E \cup \widetilde{T}_t \setminus \widetilde{E}_t] \tag{1}$$

holds inductively.

The case where $t = 1$ is explained right after the proof of Lemma 7. (Note that the triangulation does not change the connectivity of vertices not in S_1.)

For the induction step, for any $t \geq 2$, we suppose (1) holds for $t-1$ and show that it holds for t. First we note that no $w \notin V_{t-1}$ is reachable to S_t in $G[E \cup \widetilde{T}_t \setminus \widetilde{E}_t]$. To see this, first note that $S_t \subseteq V_{t-1}$; hence, every vertex in S_t is reachable to S_{t-1} in $G[E \cup \widetilde{T}_{t-1} \setminus \widetilde{E}_{t-1}]$. Thus, if some $w \notin V_{t-1}$ were reachable to S_t by using edges in $E \cup \widetilde{T}_t \setminus \widetilde{E}_t$, then it must be reachable to S_{t-1} by using edges in $E \cup \widetilde{T}_{t-1} \setminus \widetilde{E}_{t-1}$, contradicting the induction hypothesis. (Note that T_t, a set of edges among S_t, does not change the reachability.) Therefore, for any $v \notin V_{t-1}$, we have that (i) $v \notin V_t$ and (ii) v is not reachable to S_t in $G[E \cup \widetilde{T}_t \setminus \widetilde{E}_t]$, implying (1).

Now consider any $v \in V_{t-1}$, i.e., a vertex of $[G]_{\mathbf{S}_{t-1}}$. Then by inductive definition of V_t $(= \mathrm{V}([G]_{\mathbf{S}_t}))$, we see that

$$v \in V_t \iff v \in \mathrm{V}\left(\left[[G]_{\mathbf{S}_{t-1}}\right]_{S_t}^{b_t}\right).$$

That is,

$$\begin{aligned} v \in V_t &\iff v \text{ is reachable to } S_t \text{ by using edges in } E_{t-1} \setminus E^{1-b_t} \\ &\iff v \text{ is reachable to } S_t \text{ by using edges in } \left(E \cup \widetilde{T}_{t-1} \setminus \widetilde{E}_{t-1}\right) \setminus E^{1-b_t}, \end{aligned}$$

where the last characterization follows from the following facts: (i) E_{t-1} is the set of edges in $E \cup \widetilde{T}_{t-1} \setminus \widetilde{E}_{t-1}$ whose two end points are in V_{t-1}, and (ii) no vertex $w \notin V_{t-1}$ is reachable to S_t by using edges in $E \cup \widetilde{T}_{t-1} \setminus \widetilde{E}_{t-1}$. From the last characterization, we have (1) since $\left(E \cup \widetilde{T}_{t-1} \setminus \widetilde{E}_{t-1}\right) \setminus E^{1-b_t} = E \cup \widetilde{T}_{t-1} \setminus \widetilde{E}_t$ and adding T_t does not change the reachability situation. □ □

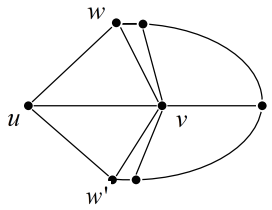
3.3 Proof of Lemma 5

We give a proof of our main technical lemma.

First, define S' and τ'. We define vertices for S' by using edges of $G[S]$ that are "borders" of G_S^0 and G_S^1. More precisely, for any edge $\{u, v\}$ of $G[S]$, we consider two triangles $\{w, u, v\}$ and $\{w', u, v\}$ adjacent to $\{u, v\}$. (Below we refer these w and w' as *adjacent triangle vertices* of $\{u, v\}$.) Edge $\{u, v\}$ is regarded as a *border edge* if and only if

$$\neg\left[\; \exists b \in \{0,1\}\left[\, w \in G_S^b \wedge w' \in G_S^b \,\right] \;\right]$$

holds, that is, they do not belong to the same subgraphs defined by S. Let E'_{all} be the set of all such border edges. We define S' to be the set of end points of E'_{all}. Clearly S' is a subset of S, and we have $V[G_{S'}^b] \subseteq V[G_S^b] \cup S$ for any $b \in \{0, 1\}$. Then define τ' by removing all v_j



Suppose that some vertex $v \in G[E']$ is connected only to u in $G[E']$. Then only u is included in S' from $\mathrm{N}(v)$, and two adjacent triangle vertices w and w' of $\{u, v\}$ are connected in $G[V \setminus S']$.

Figure 6: Two triangle vertices of edge $\{u, v\}$ (for Claim 1)

from $\tau = \{v_1, \ldots, v_k\}$ that are connected to some v_i for some $i < j$ in $G[V \setminus S']$. Also for each $v_i \in \tau'$, we define K'_i to be a connected component of $G[V \setminus S']$ containing v_i. Then it is easy to see that S' is a labeled-separator of G, and that $K'_1, \ldots, K'_{k'}$ are the connected components associated with S' such that $G^0_{S'} = \bigcup_{i \in [k']} K'_i$ holds. Thus, the property (1) of the lemma holds.

Next we define a set of cycle representations consisting of vertices of S'. Consider connected components $K'_1, \ldots, K'_{k'}$ associated with S'. For each K'_i, we define cycle representations corresponding to it. Fix one K'_i and let K' denote it. Let E' denote the set of edges of E'_{all} such that K' contains one of its adjacent triangle vertices. We define cycle representations corresponding to K' one by one (in any order) as follows:

(i) select one edge in E' that has not been selected so far;

(ii) give a direction (u, v) to this edge so that its left adjacent triangle vertex belongs to K'; record it as the first edge of a new constructing cycle representation; record u as u_0;

(iii) repeat the following until v becomes u_0: let $\mathrm{N}'(v)$ be the set of vertices of $G[E']$ that are adjacent to v; select w in $\mathrm{N}'(v)$ that is the immediately before vertex of u in the embedding order around v; give a direction (v, w) to the edge; record (v, w) as the next cycle edge; $(u, v) := (v, w)$;

(iv) go back to (i) to create a new cycle representation.

We show below that the process yields desired cycle representations for K'. Before showing this, we prove some basic facts by the following three claims.

Claim 1. *Degree of any vertex of $G[E']$ is at least 2.*

Proof. Since $G[E']$ has no isolated vertex, degree of any vertex in $G[E']$ is at least 1. Hence, we only need to prove that no vertex of v in $G[E']$ has degree 1. To prove this by contradiction, suppose that there exists a vertex v in $G[E']$ with degree 1 in $G[E']$. Let $\{u, v\}$ be the only edge incident to v in $G[E']$, and let w and w' be its adjacent triangle vertices. Since $\{u, v\} \in E'$, exactly one of its adjacent triangle vertices is in K', and we may assume that w belongs to K' and that w' does not. Note on the other hand, since $\{u, v\}$ is the unique edge incident to v in $G[E']$, considering edges incident to v (in G), it is easy to see that w and w' are connected in $K' \subseteq G[V \setminus S']$ (see Figure 6); hence, w' is also in K'. A contradiction. □ (Claim 1)

Claim 2. *Let (v, w) be a directed edge recorded as a cycle edge in the above process. Then its left adjacent triangle vertex belongs to K'.*

Proof. We prove this by induction. The base case is due to step (ii) where we give a direction to the first recorded edge so that its left adjacent triangle vertex belongs to K'.

For induction step, we consider some point in step (iii) when some vertex w is selected from $\mathrm{N}'(v)$. Let (u, v) be the directed edge just have been recorded before this point. By induction

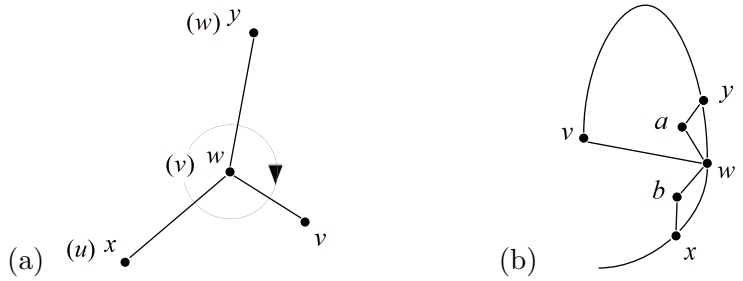


Figure 7: Situation when w is selected twice (for Claim 3)

hypothesis, we may assume that the left adjacent triangle vertex of (u, v) belongs to K'. Since $(u, v) \in E'$, its right adjacent triangle vertex does not belong to K'. By construction, w is the vertex of $\mathrm{N}'(v)$ that is located immediately before u in the embedding order around v. Let $(w, w_1, \ldots, w_r, u)$ be the embedding order around v in G. Note that $w_1, \ldots, w_r \notin G[E']$. For showing the left adjacent triangle vertex of (v, w) is in K', it suffices to show that w_1, the right adjacent triangle vertex of (v, w), does not belong to K'. To show that w_1 is not in K', observe first that w_r is the right adjacent triangle vertex of (u, v); hence, w_r is not in K' as confirmed above. Consider w_{r-1} next. Since $\{v, w_r\}$ is not in E', both of adjacent triangle vertices of $\{v, w_r\}$ belong to K', or neither of them belongs to K'. Since u, the one of adjacent triangle vertex of $\{v, w_r\}$, does not belong to K', w_{r-1}, the other one of adjacent triangle vertex of $\{v, w_r\}$, does not belongs to K' either. Similarly, we can show that none of $w_{r-2}, \ldots, w_1$ is in K'; in particular, w_1 is not in K' as desired. □ (Claim 2)

Claim 3. *Consider the iterations at step (iii), and let w be the vertex selected at one of the iterations. Then w is not selected during the current cycle construction unless $w = u_0$.*

Proof. To prove the claim by contradiction, suppose that $w \neq v_0$ is selected again at step (iii) and that (v, w) is recorded as the next cycle edge. Let (x, w) and (w, y) be two edges recorded as parts of the current cycle before.

First we show that v is not on the right of (x, w). For this, suppose that v were on the right of (x, w) and consider the iteration at (iii) where u, v, w correspond to x, w, y (see Figure 7(a)); that is, (x, w) has been recorded and y is selected as a vertex in $\mathrm{N}'(w)$ that is immediately before x in the embedding around w. Recall that our embedding is given in the clockwise order; hence, v must have been selected instead of y. A contradiction.

Next suppose that v were on the left of (x, w). Then we have a cycle C' starting from (w, y) ending by (v, w), and x is not $C_{\mathrm{in}}[C']$ (see Figure 7(b)). Let a (resp., b) be the left adjacent triangle vertex of (x, w) (resp., (w, y)). Then it is easy to see that a is in $G_{\mathrm{in}}[C']$ and b is in $G_{\mathrm{out}}[C']$; thus, a and b are not connected in $G[E \setminus C']$ (and hence not connected in $G[E \setminus E']$ either). On the other hand, by Claim 2, both a and b must be in K', a connected component of $G[E \setminus E']$. A contradiction. □ (Claim 3)

Now from the above claims, it is clear that one cycle representation is constructed by step (i)-(iii). Also it is easy to see that our process defines a set of cycles using all edges in E'. This set of cycles is defined for each K'_i, and the final set of cycle representations is the union of all these sets of cycles. Note that every edge of E'_{all} is chosen in E' for some K'_i. Hence, every edge in E'_{all} appears in some cycle; on the other hand, only edges in E'_{all} are used for cycle edges. From these observations, the property (2) of the lemma follows.

For the property (3), suppose that we created two cycles C_i and C_j that share some edge $\{u, v\}$ (in either direction) and derive a contradiction. Note that $\{u, v\}$ belongs to E'_{all}, that is, $\{u, v\}$ is a border edge.

First consider the case where these cycles are defined for different K'_i (for C_i) and K'_j (for C_j). Then $\{u, v\}$ is selected in E' when $K' = K'_i$; hence, one of its triangle vertex must be in K'_i. Then similarly, the other triangle vertex must be in K'_j. Thus, these two triangle vertices are both in G^0_S, which contradicts to the assumption that $\{u, v\}$ is a border edge.

Next consider the case where C_i and C_j are created for the same K'. Suppose first that C_i has (u, v) while C_j has (v, u). Since C_i has (u, v), by Claim 2 K' has the left adjacent triangle vertex of (u, v). Similarly, the assumption that C_j has (v, u) implies that K' has the left adjacent vertex of (v, u), which is the right adjacent vertex of (u, v). This contradicts to the assumption that $\{u, v\}$ is a border edge. Suppose next that (u, v) is a directed edge of both C_i and C_j; we may assume that (v, w) (resp., (v, w')) is selected as the next edge of C_i (resp., C_j) for some $w \neq w'$. Consider the process that defines C_i. This process selects w after recording (u, v). Therefore w is the vertex of $\mathrm{N}'(v)$ that is located immediately before u in the embedding order around v. The situation is the same for w', and then w' must be the vertex of $\mathrm{N}'(v)$ that is located immediately before u in the embedding order around v. A contradiction.

For the property (4), recall that we have shown that $G^0_{S'} = \bigcup_{i=1}^{k'} K'_i$. Thus, it suffices to show that each K'_i is expressed as $\bigcap_r G_{\mathrm{in}}[C_{j_r}]$ by using some $C_{j_1}, \ldots, C_{j_t}$ defined by our process.

Let us focus any K'_i; as above, let K' denote it, and let E' denote the set of edges of $G[S']$ such that one of its adjacent triangle vertex belongs to K'. Also let $C_1, \ldots, C_r$ denote cycles that our process defines for K'. Below we show that $K' = \bigcap_{j\in[r]} G_{\mathrm{in}}[C_j]$.

We first show that $K' \subseteq \bigcap_{j\in[r]} G_{\mathrm{in}}[C_j]$. Consider any C' of $C_1, \ldots, C_r$. Since K' is a connected component of $G[V \setminus S']$ and $C' \subseteq G[S']$, K' is a connected component of $G[V \setminus C']$. Thus, due to the property of cycles, we have either $K' \subseteq G_{\mathrm{in}}[C']$ or $K' \subseteq G_{\mathrm{out}}[C']$ and (and not both). Here, consider step (ii) of the process for defining C'; as explained there the edge direction is determined so that $G_{\mathrm{in}}[C']$ has some vertex in K'. Thus we have $K' \subseteq G_{\mathrm{in}}[C']$.

Next we show that $K' \supseteq \bigcap_{j\in[r]} G_{\mathrm{in}}[C_j]$. For this, we show that for any $v \in V$, we have $v \in \bigcap_{j\in[r]} G_{\mathrm{in}}[C_j] \Rightarrow v \in K'$. To prove its contraposition, consider any vertex $v_0 \notin K'$, and show $v_0 \notin \bigcap_{j\in[r]} G_{\mathrm{in}}[C_j]$. This conclusion is clear when v_0 is in some cycle of $C_1, \ldots, C_r$. Thus, we assume that $v_0 \notin K' \cup \bigcup_{j\in[r]} C_j$. Consider any path $(v_0, \ldots, v_p)$ in G such that $v_p \in \bigcup_{j\in[r]} C_j$ while none of $v_1, \ldots, v_{p-1}$ belongs to $\bigcup_{j\in[r]} C_j$. Note that v_0 and v_{p-1} are connected by a path not intersecting with any cycle of $C_1, \ldots, C_r$. Let $(v_{p-1}, w_1, \ldots, w_{q-1}, w_q)$ be a subsequence of planar embedding $\mathrm{N}(v_p)$ around v_p such that w_q is the first vertex in $\bigcup_{j\in[r]} C_j$. Let C' be one of $C_1, \ldots, C_r$ that contains $\{v_p, w_q\}$. We show below that $w_q \notin G_{\mathrm{in}}(C')$, which implies $v_0 \notin G_{\mathrm{in}}(C')$ as desired because w_q is connected to v_{p-1} (and hence, connected to v_0) by a path not intersecting with any cycle of $C_1, \ldots, C_r$.

Suppose contrary that $w_q \in G_{\mathrm{in}}(C')$. Then by definition of $G_{\mathrm{in}}(C')$, w_q must be located in the left of C', and it must be the left triangle adjacent vertex of either (v_p, w_q) or (w_q, v_p). Thus, by Claim 2, w_q belongs to K', and this implies that $v_0 \in K'$ since w_q is connected to v_0 by a path not intersecting with any cycle of $C_1, \ldots, C_r$. A contradiction. This completes the proof of Lemma 5.

References

[1] E. Allender and M. Mahajan, The complexity of planarity testing, *Information and Computation*, 189(1):117–134, 2004.

[2] T. Asano and B. Doerr, Memory-constrained algorithms for shortest path problem,p in *Proc. of the 23th Canadian Conf. on Comp. Geometry* (CCCG'93), 2011.

[3] T. Asano and D. Kirkpatrick, A $O(\sqrt{n})$-space algorithm for reporting a shortest path on a grid graph, in preparation.

[4] G. Barnes, J.F. Buss, W.L. Ruzzo, and B. Schieber, A sublinear space, polynomial time algorithm for directed s-t connectivity, in *Proc. Structure in Complexity Theory Conference*,

[5] J. Edmonds, C.K. Poon, and D. Achlioptas, Tight lower bounds for st-connectivity on the NNJAG model, *SIAM J. Comput.*, 28(6):2257–2284, 1999.

[6] H. Gazit and G.L. Miller, A parallel algorithm for finding a separator in planer graphs, in *Proc. of the 28th Ann. Sympos. on Foundations of Comp. Sci.* (FOCS'87), 238–248, 1987.

[7] M.R. Henzinger, P. Klein, S. Rao, and S. Subramanian, Faster shortest-path algorithms for planar graphs, *Journal of Comput. Syst. Sci.* 55:3–23, 1997.

[8] T. Imai, K. Nakagawa, A. Pavan, N.V. Vinodchandran, and O. Watanabe, An $O(n^{\frac{1}{2}+\epsilon})$-space and polynomial-time algorithm for directed planar reachability, in *Proc. of the 28th Conf. on Comput. Complexity* (CCC'13), 277–286, 2013.

[9] G.L. Miller, Finding small simple cycle separators for 2-connected planar graphs, *J. Comput. Syst. Sci.*, 32(3):265–279, 1986.

[10] R.J. Lipton and R.E. Tarjan, A separator theorem for planar graphs, *SIAM Journal on Applied Mathematics* 36 (2):177–189, 1979.

[11] N. Pippenger and M.J. Fischer, Relations among complexity measures, *J. ACM*, 26 (2):361–381, 1979.

[12] O. Reingold, Undirected connectivity in log-space, *J. ACM*, 55(4), 2008.

[13] A. Wigderson, The complexity of graph connectivity, in *Proc. 17th Math. Foundations of Comp. Sci.* (MFCS'92), LNCS 629, 112–132, 1992.

ECCC ISSN 1433-8092

http://eccc.hpi-web.de