# 論文 / 著書情報
# Article / Book Information

| | |
|---|---|
| 題目(和文) | 有向平面グラフ上到達可能性問題に対する Õ(√n)領域多項式時間アルゴリズム |
| Title(English) | Õ(√n)-Space Polynomial Time Algorithm for Directed Planar Reachability |
| 著者(和文) | 中川航太郎 |
| Author(English) | Kotaro Nakagawa |
| 出典(和文) | 学位:博士（理学）,<br>学位授与機関:東京工業大学,<br>報告番号:甲第9962号,<br>授与年月日:2015年9月25日,<br>学位の種別:課程博士,<br>審査員:渡辺　治,伊東　利哉,小島　定吉,鹿島　亮,田中　圭介 |
| Citation(English) | Degree:,<br>Conferring organization: Tokyo Institute of Technology,<br>Report number:甲第9962号,<br>Conferred date:2015/9/25,<br>Degree Type:Course doctor,<br>Examiner:,,,, |
| 学位種別(和文) | 博士論文 |
| Type(English) | Doctoral Thesis |

# $\widetilde{O}(\sqrt{n})$-Space Polynomial Time Algorithm for Directed Planar Reachability

Kotaro Nakagawa

August 3, 2015

# Contents

# Chapter 1

# Introduction

## 1.1 Sub-linear space algorithm for reachability

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 [14], and it remains unsettled. The best known result in this direction is the two decades old bound due to Barns, Buss, Ruzzo and Schieber [5], 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 matching *lower bounds* known for solving the directed graph reachability problem on a certain model of computation known as NNJAG; see, e.g., [6]. 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 [13]. 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 [9] 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 [4] 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.

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 [11]. 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., [8]. The main contribution of [9] is to show that this idea indeed works by giving a space efficient separator algorithm based on the parallel separator algorithms of Miller [10] and Gazit and Miller [7].

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 [7], yields a simple cycle separator; but we are not sure whether the sublinear-space algorithm of [9] always yields such a simple separator. Here instead of analyzing the separator algorithm of [9], we show an algorithm to obtain cycle-separators from a given separator.)

It should be noted that, though restricted to grid graphs, the problem studied by Asano et al. in [2, 4] 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 [6] 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.

### 1.1.1 Outline of this dissertation

In next chapter, we explain standard notions and notation. We also introduce how to represent several properties of graphs. The third chapter describes a $\widetilde{O}(n^{1/2+\epsilon})$ space and polynomial time algorithm for solving the reachability problem. In the first section of this chapter, we show an $\widetilde{O}(\sqrt{n})$-space and polynomial time algorithm which computes a planar separator. The reachability algorithm is shown in the second section, this algorithm uses the separator algorithm.

In the forth chapter, we improve upon the reachability algorithm from the third chapter. in the first section, we introduce the notions of a technical sequence, a universal sequence and describe the improved algorithm. In the second section, we introduce a notion of cycle separators those are used for finding a target subarea

without any additional informations.

Finally the conclusion is in the last chapter.

# Chapter 2

# Preliminaries

## 2.1 Standard notions and notation

We will use the standard notions and notation for algorithms, complexity measures, and graphs without defining them. Throughout this paper, for any set $X$, $|X|$ denotes the number of elements in $X$. By log we mean the base 2 logarithm.

Although we are given a directed graph for the reachability problem, we often consider its underlying undirected graph as an input to some procedures. Thus, while $G = (V, E)$ is usually used to denote an input graph, it can be either an directed or an undirected graph; their distinction should be clear from the context. When necessary, for a directed graph $G$, by $\underline{G}$ we mean its underlying undirected graph. We use $n$ to denote the number of vertices of an input graph $G$; on the other hand, we sometimes use $\widehat{n}$ to denote the number of vertices of a graph $\widehat{G}$ considered in each context. For any $U \subseteq V$, let $G(U)$ (resp., $\underline{G}(U)$) denote the subgraph of $G$ (resp., $\underline{G}$) induced by $U$.

For discussing the complexity of algorithms, we follow the standard computation model. In particular, for discussing sublinear space complexity, we consider a computation model in which an input is present on some read-only outside memory, an output is produced on some write-only outside memory, and only internal memory space is sublinearly bounded.

Our algorithms heavily depend on the $O(\log n)$ space (and polynomial time) algorithm of Reingold [13] for the undirected graph reachability problem. We denote this algorithm by `UReach`. Also as shown by Allender and Mahajan [**?**], we have some $O(\log n)$-space algorithm that tests whether a given undirected graph is planar and produces (if it is planar) its combinatorial planar embedding (i.e., the order of edges adjacent to each vertex in some planar embedding).

The notion of separator is central throughout this paper. Here we define this notion formally.

**Definition 2.1.1.** *For any undirected graph $G$ and for any constant $\rho$, $0 < \rho < 1$, a subset of vertices $S$ is called a* $\rho$-separator *if (i) removal of $S$ disconnects $G$ into two subgraphs $A$ and $B$, and (ii) the number of vertices of any component is at most $\rho n$.*

*The* size *of a separator is the number of vertices in the separator.*

It is well known that every planar graph with $n$ vertices has a (2/3)-separator of size $O(\sqrt{n})$ [11].

### 2.1.1 Embedding of planar graphs

We first define some basic notions on planar graphs. While our argument should be mathematically precise, we would like to avoid rigorous but tedious topological treatments of plane graphs. For this, we will use the basic topological concepts on $\mathbb{R}^2$ (which we call *the plane*) such as line, cycle, connectivity, subplane[1], and boundary. Intuitively, a *subplane* is an open subset $O$ of $\mathbb{R}^2$ such that any pair of its points in $O$ is connected by some *line* in $O$, and its *boundary* is a set of *cycles* that separates $O$ from $\mathbb{R}^2 \setminus O$. With this exception, we will give the rest of our explanation precisely.

**Definition 2.1.2.** *A graph $G$ is* planar *if it has a* planar embedding*, a way to arrange the vertices of $\mathrm{N}_G(v)$ on the plane (i.e., $\mathbb{R}^2$) for all $v \in V$ so that no pair of edges intersect each other except for their end points. A planar embedding is specified by a* combinatorial embedding representation*, that is, lists of enumerations of vertices of $\mathrm{N}_G(v)$ for all $v \in V$ in a clockwise order around $v$ under the embedding.*
**Remark.** *In this paper, we will use a* plane graph *to mean a planar graph that is embedded to the plane following one of such combinatorial embedding representations. We also identify vertices and edges of a plane graph with the corresponding points and lines in the plane $\mathbb{R}^2$.*

**Definition 2.1.3.** *A plane graph is* triangulated *(w.r.t. its planar embedding) if addition of any edge results in a nonplanar graph. For a plane graph, its* triangulation *means to add edges to the plane graph until it gets triangulated w.r.t. this planar embedding.*

We define the notion of "face" and prepare necessary notions for discussing faces of a plane graph. Consider any plane graph $G$. A *cycle* (of $G$) is a connected subgraph where each of its vertex has two incident edges in the subgraph. If $G$ has a cycle, then $\mathbb{R}^2 \setminus G$ is separated. Each separated subplane is called a *face*. We note the following fact about faces; see, e.g., [15]. Recall here that $G$ is called *2-connected* if every pair $\{u, v\}$ of vertices of $G$ are connected by two paths not sharing vertices except for $\{u, v\}$.

**Fact 1.** *Consider any plane graph $G$. Any cycle $c$ of $G$ separates the plane into two subplanes. Furthermore, if one of them has no vertex of $G$, then this subplane is in fact a face of $G$ that has $c$ as a boundary. On the other hand, if $G$ is 2-connected, then every face of $G$ has a boundary consisting of one cycle.*

Since we will mainly consider 2-connected plane graphs in this paper, we may assume (almost) one-to-one correspondence between faces and their boundary cycles. Consider any 2-connected plane graph $G$. For any cycle $c$ of $G$, if one of two subplanes

[1]The notion of "subplane" is sometimes called "region", but we will use the term "region" in a different way in this paper.

separated by $c$ has no vertex of $G$, then this subplane is called *the face defined by* $c$. We also refer the side of $c$ with this subplane as the *inside* of $c$. From the above lemma, we see that every face of $G$ is defined by some cycle. (There may exist a special case where both separated subplanes have no vertex of $G$, in which case we consider that the cycle defines two faces.)

**Definition 2.1.4** (face boundary representation)**.** *For any 2-connected plane graph $G$, consider any cycle $c$ that defines a face. A* face boundary representation *is a sequence $\sigma$ of vertices of $c$ such that (i) $\sigma$ starts and ends with the same vertex, (ii) every pair of vertices appearing next each other in $\sigma$ is adjacent in $G$, and (iii) the left side of $c$ w.r.t. the vertex ordering of $\sigma$ is the inside (i.e., the face side) of $c$.*

**Definition 2.1.5** (complete face information)**.** *Two faces are* edge-connected *(resp.,* incident*) if they share some boundary edge, (resp., some vertex on their boundaries). A* complete face information *is a list of face boundary representations and a list of edge-connected and incident pairs of faces.*

**Fact 2.** *For any plane graph with $n$ vertices, its complete face information can be expressed by $O(n)$ bits.*

# Chapter 3

# $\widetilde{O}(n^{0.5+\epsilon})$-space algorithm

## 3.1 Separator algorithm

### 3.1.1 Outline

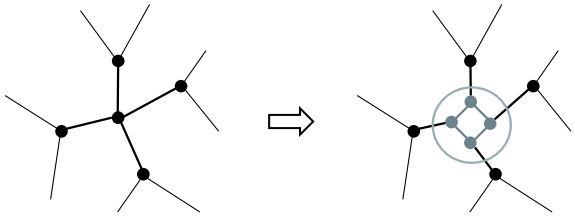
Here explain an outline of our separator algorithm. For a given graph $G$, we compute a planar embedding of $G$ and transform it to a "triangulated plane graph", which is transformed further to a "vertex expanded graph" so that its "dual graph" $\widetilde{G}$ becomes a triangulated plane graph. We then select some of its dual vertices to define an induced subgraph $\widetilde{H}$ whose faces have a weight proportional to the number of deleted dual vertices inside of the faces. We define $\widetilde{H}$ so that we can apply the separator algorithm of Gazit and Miller [7] to obtain a separator of $\widetilde{H}$ in $\widetilde{O}(\sqrt{n})$-space and polynomial-time. Then it is not so hard to obtain the desired separator for $G$ from the separator of $\widetilde{H}$. We show that these pre- and post-processes can be done in $O(\sqrt{n})$-space and polynomial-time.

For discussing their planar separator algorithm, Gazit and Miller [7] considered "face incidence graph", by regarding faces of $G^{\mathrm{tr}}$ as vertices and defining an edge between a pair of incident faces. Here we use the standard dual graph notion by which we can argue more simply.

**Definition 3.1.1.** *For any graph $G'$, its* dual graph *is* $\mathrm{dual}(G') = (V', E')$ *where (i) $V'$ is the set of all faces of $G'$, and (ii) $E'$ is the set of edge-connected pairs of faces of $G'$.*

Clearly the dual graph of $G^{\mathrm{tr}}$ is planar graph and we can easily compute its planar embedding from the one for $G^{\mathrm{tr}}$. Below we assume this embedding for the dual graph. On the other hand, the dual graph is not triangulated. Since it would be simpler if we may assume that all faces of the dual graph are triangles, we make further simple modification to our base graph $G^{\mathrm{tr}}$. Consider any vertex $v$ of $G^{\mathrm{tr}}$, and let $E_v$ be the set of edges incident to $v$. We replace $v$ by a face consisting of $|E_v|$ boundary edges, where each edge of $E_v$ is now connected to one vertex of the face following the order of embedding (Figure 3.1). We call this replacement as *vertex expansion*; a graph obtained by applying the vertex expansion to all vertices is called a *vertex expanded*

*graph*. (Note that a vertex of degree three is also expanded, and there is no vertex of degree two in $G^{\mathrm{tr}}$ except for the special case where $G$ is a simple triangle; hence, all vertices get expanded by our vertex expansion.)



An example of the vertex expansion. A degree four vertex in the center of the left graph is expanded to get the right graph.

Figure 3.1: Vertex expansion

The following fact is easy to show. (Here we assume that $G^{\mathrm{tr}}$ is not a single triangle.)

**Fact 3.** *Let $G^{\mathrm{ve}}$ be a vertex expanded graph of $G^{\mathrm{tr}}$. Then (i) it has $\Theta(n)$ vertices and $\Theta(n)$ faces, (ii) it is three regular (i.e., each vertex has three incident edges), (iii) it has a planar embedding naturally defined from $G^{\mathrm{tr}}$'s embedding, and (iv) its dual graph* $\mathrm{dual}(G^{\mathrm{ve}})$ *is a triangulated plane graph w.r.t. the embedding naturally defined from $G^{\mathrm{ve}}$'s embedding.*

It is also easy to design an $O(\log n)$-space algorithms for obtaining $G^{\mathrm{tr}}$ from $G$, $G^{\mathrm{ve}}$ from $G^{\mathrm{tr}}$, and $\mathrm{dual}(G^{\mathrm{ve}})$ from $G^{\mathrm{ve}}$ respectively. Thus, we assume that $G$, $G^{\mathrm{tr}}$, $G^{\mathrm{ve}}$, and $\mathrm{dual}(G^{\mathrm{ve}})$ are all given as input. We assume that the correspondences between $G^{\mathrm{tr}}$ and $G^{\mathrm{ve}}$, and between $G^{\mathrm{ve}}$ and $\mathrm{dual}(G^{\mathrm{ve}})$ are also given. We design an algorithm that computes a separator of the dual graph $\mathrm{dual}(G^{\mathrm{ve}})$. From the following lemma, it suffices to obtain 1/10-separator of size $O(\sqrt{n})$ for proving our target theorem. (Recall that $n$ is the number of vertices of the initial $G$, and we may assume that $n$ is sufficiently large.)

**Lemma 3.1.1.** *Let $\widetilde{S}$ be any 1/10-separator of* $\mathrm{dual}(G^{\mathrm{ve}})$ *of size $O(\sqrt{n})$. Then $G$ has a* 1/31*-separator $S$ of size $O(\sqrt{n})$.*

*Proof.* First we introduce a way to define a set of vertices of $G^{\mathrm{ve}}$ (resp., $G$) that corresponds to a given dual vertex of $\widetilde{G} := \mathrm{dual}(G^{\mathrm{ve}})$. Consider any vertex $\widetilde{v}$ of $\widetilde{G}$. Note that $\widetilde{v}$ is a face in $G^{\mathrm{ve}}$; we define $\mathrm{inv}'(\widetilde{v})$ by a set of vertices of $G^{\mathrm{ve}}$ that are incident to this face. Then define $\mathrm{inv}(\widetilde{v})$ by a set of vertices whose vertex expansion intersects with $\mathrm{inv}'(\widetilde{v})$. The following more direct interpretation of $\mathrm{inv}(\widetilde{v})$ would be helpful for our discussion. The face of $G^{\mathrm{ve}}$ that $\widetilde{v}$ corresponds is either a face obtained by expanding a vertex $v$ of $G$ or a hexagon face that corresponds to a triangle face $\{v_1, v_2, v_3\}$ of $G$. (Recall that $G$ is triangulated.) Hence, we have $\mathrm{inv}(\widetilde{v}) = \{v\}$ for the former case and $\mathrm{inv}(\widetilde{v}) = \{v_1, v_2, v_3\}$ for the latter case. For any set $\widetilde{W}$ of dual

vertices of $\widetilde{G}$, we define $\mathrm{inv}'(\widetilde{W})$ (resp., $\mathrm{inv}(\widetilde{W})$) by $\mathrm{inv}'(\widetilde{W}) = \bigcup_{\widetilde{v}\in\widetilde{W}} \mathrm{inv}'(\widetilde{v})$, and $\mathrm{inv}(\widetilde{W}) = \bigcup_{\widetilde{v}\in\widetilde{W}} \mathrm{inv}(\widetilde{v})$.

Now for a given separator $\widetilde{S}$, we show that $S := \mathrm{inv}(\widetilde{S})$ is a separator of $G$ with a desired property. Note first that we have $|S| \leq 3|\widetilde{S}|$ from the above interpretation of inv. Thus, from the assumption, we have $|S| = O(\sqrt{n})$. Below we use $\widetilde{H}_1$ and $\widetilde{H}_2$ to denote sets of vertices of $\widetilde{G}$ separated by $\widetilde{S}$. Also for each $i \in \{1, 2\}$, we let $H_i = \mathrm{inv}(\widetilde{H}_i) \setminus S$. These symbols are used also to denote their corresponding induced graphs.

We show that $S$ is indeed a separator of $G$ separating $H_1$ and $H_2$. For this we consider $S' := \mathrm{inv}'(\widetilde{S})$ and $H'_i := \mathrm{inv}'(\widetilde{H}_i) \setminus S'$ for each $i \in \{1, 2\}$, and show that $H'_1$ and $H'_2$ are separated by $S'$. Suppose otherwise, and assume that some $v'_1 \in H'_1$ and $v'_2 \in H'_2$ are connected after removing $S'$. Then there should be a path $p'$ connecting $v'_1$ and $v'_2$ that has no vertex in $S'$. Consider a set $P'$ of faces of $G^{\mathrm{ve}}$ that share an edge with $p'$. Then clearly, a set of vertices of $\widetilde{G}$ corresponding to faces in $P'$ forms a connected component of $\widetilde{G}$ that has no intersection with $\widetilde{S}$. Furthermore, due to the three regurality of $G^{\mathrm{ve}}$, any vertex of $\widetilde{G}$ (whose face is) incident to $v'_1$ (resp., $v'_2$) is connected to this connected component. This means that there are vertices $\widetilde{v}_1 \in \widetilde{H}_1$ and $\widetilde{v}_2 \in \widetilde{H}_2$ that are connected by some path in $\widetilde{G}$ with no common vertex with $\widetilde{S}$, contradicting that $\widetilde{S}$ separates $\widetilde{H}_1$ and $\widetilde{H}_2$.

Next we show that $S$ is a (1/31)-separator. Without losing generality we may assume that $|H_1| \leq |H_2|$, and here we show that $|H_1| \geq n/31$ for sufficiently large $n$.

Consider first vertices of $\widetilde{H}_1$ and $\widetilde{G}$. As explaiend above, each vertex of $\widetilde{G}$ corresponds to either a vertex of $G$ or a triangle face of $G$; we call a vertex of $\widetilde{G}$ a *v-vertex* for the former case and a *f-vertex* for the latter case. The converse relation also holds; that is, each vertex of $G$ and each triangle face of $G$ corresponds to some vertex of $\widetilde{G}$. Thus, we have $\widetilde{n} = n + F$, where $\widetilde{n}$ is the number vertices of $\widetilde{G}$ and $F$ is the number of faces of $G$. Let $V_1$ and $F_1$ denote respectively the number of v-vertices and f-vertices of $\widetilde{H}_1$. Then we have from our assumption that

$$V_1 + F_1 \ \geq\ \widetilde{n}/10 \ \geq\ n/10. \tag{3.1}$$

Now consider a graph $H_1^+$ induced by $\mathrm{inv}(\widetilde{H}_1)$. (Recal that $H_1 = \mathrm{inv}(\widetilde{H}_1) \setminus S$.) Let $n_1^+$, $e_1^+$, and $f_1^+$ denote the number of vertices, edges, and faces of $H_1^+$ respectively. Note that each face of $H_1^+$ is either an *inner face*, a triangle face corresponding to a f-vertex of $\widetilde{H}_1$, or an *outer face*, a face consisting of edges of separator $S$. Reall that $F_1$ denotes the number of inner faces. On the other hand, we use $x$ and $y$ to denote the number of outer faces and the total number of edges of all outer faces. Then we have $n_1^+ = V_1$ and $f_1^+ = F_1 + x$. On the other hand, since all inner faces are triangles, we have $2e_1^+ = 3F_1 + y$. Then apply the Euler's formula for plane graphs to $H_1^+$, we have $n_1^+ = e_1^+ - f_1^+ + 2$, which derives $V_1 - 0.5F_1 = y/2 - x + 2$. Hence, by using (3.1) we have

$$3V_1 \ \geq\ n/10 + y - 2x + 4 \ \geq\ n/10 - 2x.$$

Reall that $H_1 = H_1^+ \setminus S$, and note that $2x \leq 3|S|$ because each vertex of $S$ can create at most three edges. Thus, we have

$$|H_1| \ \geq\ n_1^+ - |S| \ =\ V_1 - |S| \ \geq\ n/30 - 2x/3 - |S| \ \geq\ n/30 - 2|S|.$$

Then the desired bound holds for sufficiently large $n$ since $|S| = O(\sqrt{n})$. □

**Notation 1.** *For simplifying our discussion, we will regard from now on* $G^{\mathrm{ve}}$ *as the initial input graph* $G$*. Thus, we will use* $G = (V, E)$ *to denote* $G^{\mathrm{ve}}$*. Also we will use* $\widetilde{G} = (\widetilde{V}, \widetilde{E})$ *to denote* $\mathrm{dual}(G^{\mathrm{ve}})$*. Let* $n$ *and* $\widetilde{n}$ *denote the number of vertices of these new* $G$ *and* $\widetilde{G}$ *respectively. In general, we will use a prefix "d-" to denote a vertex, an edge, a path, and a cycle in* $\widetilde{G}$ *to distinguish them from those in* $G$*, and we will attach "˜ " to variables denoting these objects; for example, a vertex of* $\widetilde{G}$ *is called a d-vertex and it is denoted by, e.g.,* $\widetilde{v}$*. (Since subplanes and faces are notions involving some topological interpretation on* $\mathbb{R}^2$*, we will not use "d-" to denote them even if they are defined w.r.t. the dual graph* $\widetilde{G}$*.)*

We will make use of a separator algorithm given by Miller [10], applied on a "weighted" subgraph of $\widetilde{G}$. In order to explain the algorithm of Miller we need some notions and basic facts. (Here for simplicity, we explain these notions and facts for a general graph $G'$ and its subgraph $H'$ though we will apply them to our graph $\widetilde{G}$ and its subgraph $\widetilde{H}$.)

A *weighted graph* is a plane graph whose faces are given weights in such a way that their total is at most 1. In particular, we will use the following weight for a subgraph of a given plane graph; for a given plane graph $G'$ with $n'$ vertices and its subgraph $H'$, the weight of each face $f'$ of $H'$ is defined by $n'_{f'}/n'$, where $n'_{f'}$ is the number of vertices of $G'$ that are located inside of $f'$. We generalize the notion of $\rho$-separator for such weighted graphs. Note that a weight of a subgraph of a weighted graph is simply the total weight of the faces in the subgraph.

**Definition 3.1.2** ($\rho$-separator for a weighted graph)**.** *For any weighted graph, and for any* $\rho \in (0, 1)$*, a* $\rho$-separator *of the graph is a set of vertices* $S$ *such that the removal of* $S$ *creates two disconnected subgraphs each of which has weight* $\geq \rho$*.*

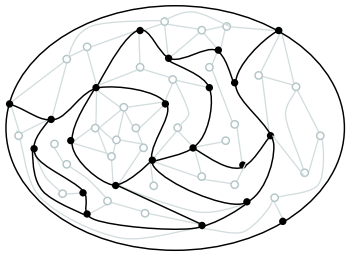
Any cycle of a plane graph separates the graph into two parts. Such a separator is called a *cycle separator*. We can easily see the following fact on cycle separators. (*Cf.* A separator of $H'$ is not necessarily a separator of $G'$.)

**Fact 4.** *Let* $H'$ *be any weighted subgraph of a plane graph of* $G'$ *with* $n'$ *vertices. A cycle separator of* $H'$ *is also a separator of* $G'$*, and if it is* $\rho$*-separator w.r.t. the weight of* $H'$*, the removal of* $S'$ *from* $G'$ *creates two disconnected subgraphs each of which has at least* $\rho n'$ *vertices.*

In [10] Miller gave an algorithm satisfying the following theorem.

**Theorem 3.1.2.** *For any plane and two-connected weighted graph with* $n'$ *vertices and* $n''$ *faces such that each of its face has a weight* $\leq 2/3$ *and size* $\leq d$*. Then there exists a cycle* $1/3$*-separator of size* $2\sqrt{2\lfloor d/2 \rfloor n'}$*. Furthermore, there exists a parallel algorithm using polynomial number of processors that finds one of such separators in* $O(n')$*-time (under the unit cost).*

Looking into the proof of this theorem, we see that the algorithm of Miller for proving the theorem operates only on faces of a given weighted graph if its complete



An example explaining Fact 4. Black vertices and edges are the vertices and edges of $G'$ selected for a subgraph $H'$ whereas grey vertices and edges are those in $G'$ not selected for $H'$. The weight of each face of $H'$ is defined proportional to the number of vertices of $G'$ inside of the face. Note that there is no edge connecting two vertices of $G'$ in two faces of $H'$. Thus, any cycle separator of $H'$ is also a cycle separator of $G'$.

Figure 3.2: Graph $G'$ and its subgraph $H'$

face information is given. The algorithm yields a separator of size $\leq 2d\sqrt{n''}$ (see the remark of Section 1.2 of [7]) in linear time w.r.t. $n''$. Thus, we in fact have the following algorithm, which will be used in our discussion.

**Collorary 3.1.3.** *(Corollary to the proof of Theorem 3.1.2 stated in Section 1.2 of [7])*
*There exists a polynomial-time algorithm such that for any plane and two-connected weighted plane graph with $n''$ faces of size $\leq d$, it takes the complete face information of the graph and yields its cycle $1/3$-separator of size $2d\sqrt{n''}$ in polynomial-time and $\widetilde{O}(n'')$-space.*

Gazit and Miller [7] proposed a way to obtain some subgraph from $G$ for which the above theorem (in our case its collorary) can be applied to get a separator of size $\widetilde{O}(\sqrt{n})$. Unfortunately, though, their algorithm uses $O(\log^2 n)$-parallel-time and it is unclear whether this can be executed in $\widetilde{O}(\sqrt{n})$-space. Thus, though following the outline of Gazit and Miller, we will give a new approach to define our desired separator algorithm for Theorem **??**.

Now we explain the outline of our approach. As mentioned above, our technical goal is to obtain a separator of $\widetilde{G} = (\widetilde{V}, \widetilde{E})$. For this, we first obtain from $\widetilde{G}$ its weighted subgraph $\widetilde{H} = (\widetilde{U}, \widetilde{D})$. More specifically, we show an algorithmic way to obtain[1] $\widetilde{H}$ satisfying the following conditions. (We let $k = \sqrt{n}$ and we will fix this usage hereafter.)

(F1) $\widetilde{H}$ is a subgraph of $\widetilde{G}$ induced by some subset of d-edges of $\widetilde{G}$. $\widetilde{H}$ is a plane graph w.r.t. the embedding naturally inherited from $\widetilde{G}$. Also it is 2-connected.

(F2) $\widetilde{H}$ contains $O\big(\max\{k, n/k\}\big)$ faces.

(F3) $\widetilde{H}$ has no face of weight $> 2/3$, or if it has a face $f$ of weight $> 2/3$, then the following holds for a subgraph $\widetilde{G}_f$ of $\widetilde{G}$ induced by d-vertices of $\widetilde{G}$ in $f$: There exists a d-vertex $\widetilde{b}$ in $\widetilde{G}_f$ such that any vertex in $\widetilde{G}_f$ is reachable from $\widetilde{b}$ by a

[1] We will also show that the algorithm also computes its complete face information.

d-path of length $O(k)$ in $\widetilde{G}_f$; that is, $\widetilde{G}_f$ has a BFS (breadth-first-search) d-tree spanning $\widetilde{G}_f$ rooted at $\widetilde{b}$ of depth $O(k)$.

(F4) The size of each face of $\widetilde{H}$ is $O(\sqrt{k})$.

This $\widetilde{H}$ is called a *frame graph*. We use $n'$ and $n''$ to denote respectively the number of d-vertices and faces of $\widetilde{H}$, and let $d$ be its max. face size. Then from the above conditions, we have $d = O(\sqrt{k}) = O(n^{1/4})$ and $n'' = O(k) = O(n^{1/2})$; hence, $n' \leq d \times n'' = O(n^{3/4})$.

For obtaining our target separator, we apply either the separator algorithm of Lipton and Tarjan (Theorem **??**) or that of Miller (Corollary 3.1.3) depending on the max. weight of the faces of $\widetilde{H}$.

Consider first the case where there is no face with weight $> 2/3$. In this case we use Miller's algorithm to compute a cycle 1/3-separator of $\widetilde{H}$. From conditions (F4) and Corollary 3.1.3, it follows that the size of this separator is $O(d\sqrt{n''}) = O(\sqrt{n})$, and that Miller's algorithm can be executed in $\widetilde{O}(n'')$-space ($= \widetilde{O}(\sqrt{n})$-space). From Fact 4, it is easy to see that the obtained separator is the desired 2/9-separator of $\widetilde{G}$ for sufficiently large $n$.

Consider next the case where there is some face $f$ with weight $> 2/3$. In this case we consider the induced subgraph $\widetilde{G}_f$ of $\widetilde{G}$ consisting of d-vertices of $\widetilde{G}$ located either on the boundary or inside of $f$. We apply the algorithm of Lipton and Tarjan to $\widetilde{G}_f$. Clearly, the obtained 1/3-separator is a separator of $\widetilde{G}$. Since $f$ has weight $> 2/3$, there must be at least $2\widetilde{n}/3$ d-vertices in $\widetilde{G}_f$; hence, each separated part of $\widetilde{G}$ in $\widetilde{G}_f$ has at least $2\widetilde{n}/9$ d-vertices. Thus, the obtained separator is a 2/9-separator of $\widetilde{G}$, and its size is at most $d = O(n^{1/4})$. Recall that the work space needed for the algorithm of Lipton and Tarjan is essentially for conducting a breadth-first-search and that the amount of required work space is linear to the depth of its BFS d-tree; hence, from condition (F3) it follows that the algorithm of Lipton and Tarjan can be executed in $\widetilde{O}(k)$-space ($= \widetilde{O}(\sqrt{n})$-space).

In the following sections, we will design an algorithm — `Algo_frame` — that constructs a frame graph satisfying the above conditions. Its explanation is given as follows. First in Section 2 we prepare some notions for our discussion, and then explain the first step converting $\widetilde{G}$ toward a frame graph. Then in Section 3 we propose a way to define a frame graph. In Section 4, we explain algorithm `Algo_frame` and confirm that it runs in $\widetilde{O}(\sqrt{n})$-space and polynomial-time to compute the frame graph and its complete face information.

### 3.1.2 Voronoi Region, Floor, and Ceiling

In this section, we prepare some key notions for our following discussion. We then introduce modifications on $G$ and $\widetilde{G}$ as the first step of defining a frame graph.

Though our target frame graph is a subgraph of the dual graph $\widetilde{G}$, we also need to consider the graph $G$ for our analysis. Unless specified otherwise, we use symbols $v$ and $e$ (resp., $\widetilde{v}$ and $\widetilde{e}$) to denote vertices and edges of $G$ (resp., d-vertices and d-edges of $\widetilde{G}$). We also use $\widetilde{u}$, $\widetilde{w}$, and sometimes $\widetilde{b}$ for d-vertices.

Here we note/recall the correspondence between $G$ and $\widetilde{G}$. First note that each face of $G$ is a d-vertex of $\widetilde{G}$. We say that a d-edge $\widetilde{e} = (\widetilde{u}, \widetilde{v})$ *crosses* an edge $e$ (and in parallel, $e$ *crosses* $\widetilde{e}$) if two faces corresponding to $\widetilde{u}$ and $\widetilde{v}$ share the edge $e$. Note that there is one-to-one correspondence between d-edge and the edge that it crosses.

**Definition 3.1.3.** *A* region *is a set $R$ of d-vertices of $\widetilde{G}$ such that any two d-vertices of $R$ is connected by some d-path consisting of only d-vertices of $R$. We also consider region $R$ as a set of faces of $G$, and a set of edges defined by*

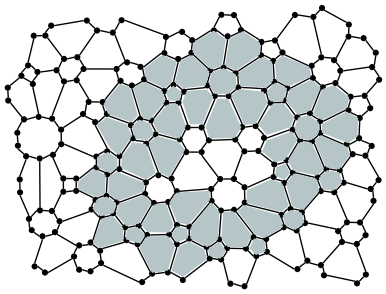
$$\{\, e \in E \mid \text{there exists exactly one face in } R \text{ that contains } e \,\}$$

*is the* boundary *of $R$.*

**Remark.** *Though a region is a set of d-vertices of $\widetilde{G}$, we also regard it as a set of faces of $G$. Thus, we will use symbols without $\widetilde{\ }$ for regions.*

We recall an important fact from [**?**] that is provable for three regular graphs.

**Fact 5.** *For any region (of our three regular graph $G$), its boundary consists of simple cycles.*



An example of a region and its boundary. Here vertices and edges are those in $G$; each face is regarded as a d-vertex of $\widetilde{G}$. The shaded part is a region. The boundary of this region consists of two cycles. In general Miller showed that the boundary of any region is a set of simple cycles [**?**].

Figure 3.3: An example of a region and its boundary

We introduce some specific family of regions. For any d-vertices $\widetilde{u}$ and $\widetilde{v}$, by $\mathrm{dist}(\widetilde{u}, \widetilde{v})$ we mean the distance between $\widetilde{u}$ and $\widetilde{v}$, that is, the length of a shortest d-path between $\widetilde{u}$ and $\widetilde{v}$. For any d-vertex $\widetilde{v}$, we define a total order (denoted by $<_{\widetilde{v}}$) in $\widetilde{V}$ based on the distance from $\widetilde{v}$. For any d-vertices $\widetilde{u}$ and $\widetilde{w}$, we say that $\widetilde{u}$ is nearer to $\widetilde{v}$ than $\widetilde{w}$ (written as $\widetilde{u} <_{\widetilde{v}} \widetilde{w}$) if we have either

- $\mathrm{dist}(\widetilde{u}, \widetilde{v}) < \mathrm{dist}(\widetilde{w}, \widetilde{v})$, or
- $\mathrm{dist}(\widetilde{u}, \widetilde{v}) = \mathrm{dist}(\widetilde{w}, \widetilde{v})$, and $\widetilde{u}$ has a smaller index[2] than $\widetilde{w}$.

For any d-vertex $\widetilde{v}$ and any integer $d \geq 0$, let $\mathrm{L}(\widetilde{v}, d)$ denote the set of d-vertices whose distance from $\widetilde{v}$ is $d$.

[2]We assume that d-vertices are indexed in a certain way.

**Definition 3.1.4** ($k$-neighborhood)**.** *For any d-vertex $\widetilde{v}$, let $\mathrm{d_{nb}}(\widetilde{v})$ be the largest integer $d$ such that $|\cup_{0\le i\le d}\mathrm{L}(\widetilde{v},i)| < k$ $(=\sqrt{n})$ holds. Then we define $\mathrm{N}_k(\widetilde{v})$ and $\mathrm{N}_k^+(\widetilde{v})$ by*

$$\mathrm{N}_k(\widetilde{v}) \;=\; \bigcup_{0\le i\le d_0}\mathrm{L}(\widetilde{v},i), \quad \text{and} \quad \mathrm{N}_k^+(\widetilde{v}) \;=\; \bigcup_{0\le i\le d_0+1}\mathrm{L}(\widetilde{v},i),$$

*where $d_0=\mathrm{d_{nb}}(\widetilde{v})$. We call $\mathrm{N}_k(\widetilde{v})$ and $\mathrm{N}_k^+(\widetilde{v})$ a* $k$-neighborhood *and a* $k$-neighborhood$^+$ *respectively.*

**Remark.** *By $\mathrm{N}_k(\widetilde{v})$ and $\mathrm{N}_k^+(\widetilde{v})$ we also mean subgraphs of $\widetilde{G}$ induced by sets $\mathrm{N}_k(\widetilde{v})$ and $\mathrm{N}_k^+(\widetilde{v})$.*

The following fact is immediate from the above definition.

**Fact 6.** *For any d-vertex $\widetilde{v}$, we have the following: (i) both $\mathrm{N}_k(\widetilde{v})$ and $\mathrm{N}_k^+(\widetilde{v})$ are regions, (ii) $|\mathrm{N}_k(\widetilde{v})|<k$ and $|\mathrm{N}_k^+(\widetilde{v})|\ge k$, and (iii) the diameter of $\mathrm{N}_k(\widetilde{v})$ and $\mathrm{N}_k^+(\widetilde{v})$ is at most $k$, that is, the distance between $\widetilde{v}$ and any vertex in $\mathrm{N}_k^+(\widetilde{v})$ is at most $k$.*

We select representative $k$-neighborhoods and define related Voronoi regions; these are basic regions for defining our frame graph $\widetilde{H}$ of $\widetilde{G}$.

**Definition 3.1.5.** *A subset $\widetilde{I}$ of $\widetilde{V}$ is a* $k$-maximal independent set *if it satisfies following conditions:*

- *For any two d-vertices $\widetilde{b}_1,\widetilde{b}_2\in\widetilde{I}$, $\mathrm{N}_k^+(\widetilde{b}_1)\cap\mathrm{N}_k^+(\widetilde{b}_2)=\emptyset$.*
- *For any d-vertex $\widetilde{v}\notin\widetilde{I}$, there exists a d-vertex $\widetilde{b}\in\widetilde{I}$ such that $\mathrm{N}_k^+(\widetilde{v})\cap\mathrm{N}_k^+(\widetilde{b})\neq\emptyset$.*

The following properties are immediate from the definition.

**Fact 7.** *Consider any $k$-maximal independent set $\widetilde{I}$ of $\widetilde{G}$. Then we have (i) $|\widetilde{I}|\le\widetilde{n}/k$, and (ii) for any d-vertex $\widetilde{v}\in\widetilde{V}$, we have some d-vertex $\widetilde{b}\in\widetilde{I}$ and some d-vertex $\widetilde{u}$ such that $\widetilde{u}\in\mathrm{N}_k^+(\widetilde{v})\cap\mathrm{N}_k^+(\widetilde{b})$ and $\mathrm{dist}(\widetilde{u},\widetilde{b})\le 2k$.*

Here again we rely on Section 4 and assume some $\widetilde{O}(\sqrt{n})$-space and polynomial-time algorithm that computes a $k$-maximal independent set for $k=\sqrt{n}$. Thus, in the following discussion, we fix $\widetilde{I}$ to denote the $k$-maximal independent set obtained by this algorithm and assume that it is given also as a part of an input. For this $\widetilde{I}$, we define Voronoi regions. Intuitively, for each $\widetilde{b}\in\widetilde{I}$, we would like to define the "Voronoi region" $\mathrm{Vr}(\widetilde{b})$ of $\widetilde{b}$ by a set of d-vertices for which $\widetilde{b}$ is closest among all d-vertices in $\widetilde{I}$. But such a d-vertex in $\widetilde{I}$ that is closes to $\widetilde{v}$ may not be computable within our desired space bound. Thus, we use "closest $k$-neighborhood" to determine our Voronoi region, by which we can, as we will see later, identify Voronoi regions in $O(k+n/k)$-space.

**Definition 3.1.6** (Voronoi region)**.** *For any set $W$ of d-vertices, $\mathrm{nrst}_{\widetilde{v}}(W)$ denotes a d-vertex $\widetilde{u}$ in $W$ such that $\widetilde{u}<_{\widetilde{v}}\widetilde{w}$ of all $\widetilde{w}\in W\setminus\{\widetilde{u}\}$. For any sets $W$ and $W'$ of d-vertices, we write $W<_{\widetilde{v}}W'$ if $\mathrm{nrst}_{\widetilde{v}}(W)<_{\widetilde{v}}\mathrm{nrst}_{\widetilde{v}}(W')$. For any d-vertex $\widetilde{v}$, the* boss *of $\widetilde{v}$ (denoted by $\mathrm{boss}(\widetilde{v})$) is a d-vertex $\widetilde{b}\in\widetilde{I}$ such that $\mathrm{N}_k^+(\widetilde{b})<_{\widetilde{v}}\mathrm{N}_k^+(\widetilde{b}')$ for any*

$\widetilde{b}' \in \widetilde{I} \setminus \{\widetilde{b}\}$. *The* Voronoi region *of* $\widetilde{b}$ *(denoted by* $\mathrm{Vr}(\widetilde{b})$*) is a set of d-vertices* $\widetilde{v}$ *such that* $\mathrm{boss}(\widetilde{v}) = \widetilde{b}$.

**Remark.** *In nutshell,* $\widetilde{b}$ *is the boss of* $\widetilde{v}$ *if and only if its* $k$*-neighborhood is nearest to* $\widetilde{v}$ *among all d-vertices in* $\widetilde{I}$*. It may be the case that there is some other d-vertex in* $\widetilde{I}$ *that is nearer to* $\widetilde{v}$*.*

By definition for every d-vertex $\widetilde{v} \in \widetilde{V}$, there exists a unique Voronoi region $\mathrm{Vr}(\widetilde{b})$ that contains $\widetilde{v}$. Furthermore, for any Voronoi region $\mathrm{Vr}(\widetilde{b})$, we show below that it is indeed a region and that every d-vertex in $\mathrm{Vr}(\widetilde{b})$ is reachable by a d-path in $\mathrm{Vr}(\widetilde{b})$ of length $\leq 2k$.

**Lemma 3.1.4.** *For any d-vertex* $\widetilde{b} \in \widetilde{I}$ *and any d-vertex* $\widetilde{v} \in \mathrm{Vr}(\widetilde{b})$*, there exists a d-path from* $\widetilde{b}$ *to* $\widetilde{v}$ *of length* $\leq 2k$ *consisting of only d-vertices in* $\mathrm{Vr}(\widetilde{b})$*.*

*Proof.* Consider any d-vertex $\widetilde{v} \in \mathrm{Vr}(\widetilde{b})$. By Fact 7 (ii), there exists some $\widetilde{b}' \in \widetilde{I}$ such that $\mathrm{N}_k^+(\widetilde{v}) \cap \mathrm{N}_k^+(\widetilde{b}') \neq \emptyset$ holds. Clearly, $\widetilde{b}$ also satisfies $\mathrm{N}_k^+(\widetilde{v}) \cap \mathrm{N}_k^+(\widetilde{b}) \neq \emptyset$ because otherwise $\widetilde{b}$ cannot be the boss of $\widetilde{v}$. Thus, there is a path from $\widetilde{v}$ to $\widetilde{b}$ of length $\leq 2k$.

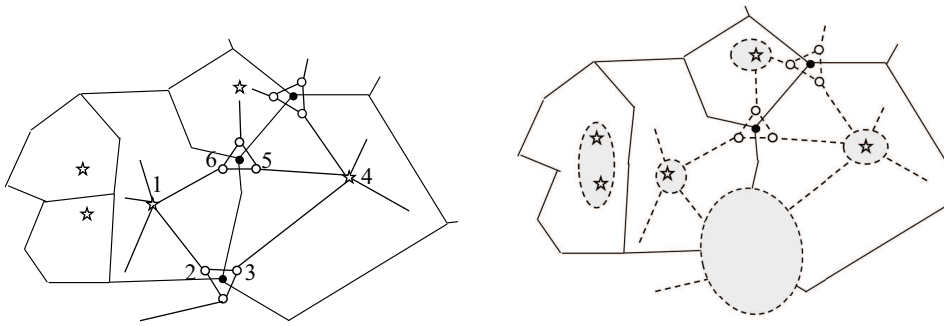
For the lemma, it suffices to show that there exists a path of length $\leq 2k$ from $\widetilde{v}$ to $\widetilde{b}$ consisting of only vertices in $\mathrm{Vr}(\widetilde{b})$. For this, consider $\widetilde{u} := \mathrm{nrst}_{\widetilde{v}}(\mathrm{N}_k^+(\widetilde{b}))$, the d-vertex in $\mathrm{N}_k^+(\widetilde{b})$ nearest to $\widetilde{v}$, which in fact is the d-vertex in the whole $\cup_{\widetilde{b}' \in \widetilde{I}} \mathrm{N}_k^+(\widetilde{b}')$ nearest to $\widetilde{v}$ witnessing that $\widetilde{v} \in \mathrm{Vr}(\widetilde{b})$. Clearly, any shortest d-path from $\widetilde{u}$ to $\widetilde{b}$ belongs to $\mathrm{N}_k^+(\widetilde{b}) \subseteq \mathrm{Vr}(\widetilde{b})$ and its length is at most $k$. (Note that it is easy to check that $\mathrm{N}_k^+(\widetilde{b}) \subseteq \mathrm{Vr}(\widetilde{b})$.) Consider any shortest d-path $\widetilde{v}$ to $\widetilde{u}$. Again it is clear that its length is at most $k$ (because otherwise $\mathrm{N}_k^+(\widetilde{v})$ has no intersection with $\cup_{\widetilde{b}' \in \widetilde{I}} \mathrm{N}_k^+(\widetilde{b}')$). We show that each d-vertex on the d-path belongs to $\mathrm{Vr}(\widetilde{b})$. Suppose otherwise; that is, there exists some d-vertex $\widetilde{v}'$ on the d-path that belongs to the other $\mathrm{Vr}(\widetilde{b}')$. Then for this $\widetilde{v}'$, there exists some $\widetilde{u}' \in \mathrm{Vr}(\widetilde{b}')$ that is nearer to $\widetilde{v}'$ than $\widetilde{u}$. But since $\widetilde{v}'$ is on one of the shortest d-paths from $\widetilde{v}$ to $\widetilde{u}$, this means that $\widetilde{u}'$ is also nearer to $\widetilde{v}$ than $\widetilde{u}$, a contradiction. Therefore the lemma follows. □

The lemma shows that each Voronoi region $\mathrm{Vr}(\widetilde{b})$ has a spanning BFS d-tree on (the subgraph induced by) the Voronoi region with $\widetilde{b}$ as a root and that the depth of such trees is $O(k)$. We will make a frame graph based on these Voronoi regions. More specifically, we will define faces of the frame graph by d-cycles like the one shown in Figure 3.4 (a); this d-cycle defines a subplane consisting of subregions of two Voronoi regions, and we remove all d-vertices in this subplane to define a face with this d-cycle as a face boundary. Let us call such faces "standard" below. From the above lemma, faces defined by such d-cycles satisfy (F3). On the other hand, we need to reduce the length of those d-cycles in order to meet (F4). For this, we follow[3] [7] and introduce the notions of "floor" and "ceiling" d-cycles to define some more faces as in Figure 3.4 (b).

We explain some intuition on the notions of "floor" and "ceiling." Consider any d-vertex $\widetilde{b} \in \widetilde{I}$. For each $\ell \geq 1$, let $\mathrm{L}_{\mathrm{nb}}(\widetilde{b}, \ell)$ denote $\mathrm{L}(\widetilde{b}, \mathrm{d}_{\mathrm{nb}}(\widetilde{b}) + \ell)$; that is, it is the set

[3]Though we follow the outline of the argument in [7], the notions defined here are different from those in [7] even if some notions are given the same name.

(a) (b)

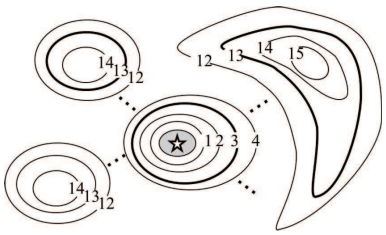



This figure shows a way to define faces in our frame graph. Only vertices, d-vertices, paths, and d-paths necessary for explanation are shown. (a) Solid lines are the boundaries of Voronoi regions, and black vertices are vertices of $G$ that are incident to three Voronoi regions. White vertices are d-vertices incident to these black vertices. The boss d-vertex of each Voronoi region is indicated by a star. Six d-vertices labeled with $1 \sim 6$ connected with dashed lines indicating d-paths forms a d-cycle that defines a "standard" face. (b) In order to reduce the length of d-cycles of such standard faces to meet (F4), some other faces (shadow parts) are introduced, which are defined by "floor" and "ceiling" d-cycles.

Figure 3.4: Voronoi regions and floor and ceiling d-cycles

of d-vertices distance $\ell$ from $\mathrm{N}_k(\widetilde{b})$. Note that $\mathrm{N}_k^+(\widetilde{b}) = \mathrm{N}_k(\widetilde{b}) \cup \mathrm{L}_{\mathrm{nb}}(\widetilde{b}, 1)$. Let us call those d-vertices in $\mathrm{L}_{\mathrm{nb}}(\widetilde{b}, \ell)$ "level $\ell$ d-vertices" (only tentatively for this explanation). A connected component of $\mathrm{L}_{\mathrm{nb}}(\widetilde{b}, \ell)$ is either a d-path or a d-cycle; let us assume that we can always find a d-cycle in $\mathrm{L}_{\mathrm{nb}}(\widetilde{b}, \ell)$ for each $\ell \geq 1$. Intuitively, the size (i.e., the number of d-vertices) of such d-cycles would increase *for a while* when $\ell$ increases from $\ell = 1$ but it should not be monotonically increasing; in fact, it should be small for sufficiently large $\ell$. We consider d-cycles (of some $\mathrm{L}_{\mathrm{nb}}(\widetilde{b}, \ell)$) of size $\leq \sqrt{k}$, which we call "necks" (Figure 3.5). We choose one neck consisting of d-vertices of some low level $\ell_0$ and some necks consisting of d-vertices of some high level $\ell_1$, and we consider only d-vertices of level between $\ell_0$ and $\ell_1$. A neck of level $\ell_0$ is a "floor" and necks of level $\ell_1$ are "ceilings." A key point here is the distance between floor and ceiling in the same Voronoi region is bounded by $O(\sqrt{k})$. We will choose floors and ceilings so that the subplanes under the floors and over the ceiling are small enough; then we may regard those parts as faces in our frame graph to reduce the size of every standard face to satisfy (F4).

Now we argue precisely. We begin with defining the notion of "core", which is used in case no neck appropriate for a floor exits. Recall that $\mathrm{d}_{\mathrm{nb}}(\widetilde{v})$ is the largest $d$ such that $| \cup_{0 \leq i \leq d} \mathrm{L}(\widetilde{v}, i) | < k$ $(= \sqrt{n})$ holds.




A figure showing an intuitive idea of a level structure from one d-vertex $\widetilde{b}$ that is shown as a star. A shadow part indicates the $k$-neighborhood of $\widetilde{b}$. Each cycle indicates a d-cycle (that we assume to find) in $\mathrm{L_{nb}}(\widetilde{b}, \ell)$ for each level $\ell \geq 1$, where the number indicates its level. A bold line cycle is either a floor or a ceiling.

Figure 3.5: An intuitive idea of a level structure

**Definition 3.1.7** (core)**.** *For any $\widetilde{b} \in \widetilde{I}$, let $\mathrm{d_{core}}(\widetilde{b})$ denote the largest $d \leq \mathrm{d_{nb}}(\widetilde{b})$ such that $|\mathrm{L}(\widetilde{v}, d)| \leq \sqrt{k}$. The* core *of $\widetilde{b}$ (denoted by $\mathrm{Core}(\widetilde{b})$) is defined by*

$$\mathrm{Core}(\widetilde{b}) \;=\; \bigcup_{0 \leq i \leq \mathrm{d_{core}}(\widetilde{b})} \mathrm{L}(\widetilde{b}, i).$$

We first note the following relationship.

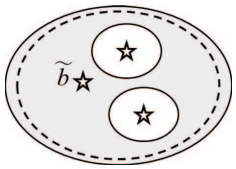
**Lemma 3.1.5.** *For any $\widetilde{b} \in \widetilde{I}$, we have $\mathrm{d_{nb}}(\widetilde{b}) - \mathrm{d_{core}}(\widetilde{b}) \leq \sqrt{k}$.*

*Proof.* Suppose otherwise, that is, $\mathrm{d_{core}}(\widetilde{b}) + \sqrt{k} < \mathrm{d_{nb}}(\widetilde{b})$ holds. Then we have

$$\bigcup_{i=1}^{\sqrt{k}} L(\widetilde{b}, \mathrm{d_{core}}(\widetilde{b}) + i) \;\subseteq\; \mathrm{N}_k(\widetilde{b}).$$

On the other hand, by definition, for any $d$ such that $\mathrm{d_{core}}(\widetilde{b}) < d \leq \mathrm{d_{nb}}(\widetilde{b})$, we have $|L(\widetilde{v}, d)| > \sqrt{k}$, from which we have $|\mathrm{N}_k(\widetilde{b})| > k$, contradicting the definition of $k$-neighborhood. □

Consider any $\widetilde{b} \in \widetilde{I}$ and its core $\mathrm{Core}(\widetilde{b})$. By definition $\mathrm{Core}(\widetilde{b})$ is a subset of $\mathrm{N}_k(\widetilde{b})$. Hence, $k' := |\mathrm{Core}(\widetilde{b})| < k$; that is, the core contains less than $k$ d-vertices. It is also a region. Thus, its boundary is a set of disjoint cycles of $G$. Let $c$ be one of such cycles. Its *exterior* is a subplane not containing the core. $c$ is called a *core cycle* if its exterior contains at least $2\widetilde{n}/3$ d-vertices. It may be the case that no core cycle exists. In this case, by removing the core, we would have separated subgraphs $\widetilde{G}_1, \ldots, \widetilde{G}_t$ into $\cup_{1 \leq i \leq t_0} \widetilde{G}_i$ and $\cup_{t_0 < i \leq t} \widetilde{G}_i$ for some $t_0$ such that both parts have at least $(\widetilde{n}/3) - k'$ d-vertices. Hence, the core itself is, say, a 1/4-separator for sufficiently large $k = \sqrt{n}$ and $\widetilde{n}$, and its size is $k' \leq \sqrt{n}$; thus, the core itself can be used as a separator satisfying our goal theorem. Therefore, in the following we may assume that each core has a core cycle. Note also that two core cycles do not exist because the exterior of such cycles are disjoint and no disjoint two sets can contain $2\widetilde{n}/3$ d-vertices in each.




An example of a core with a boundary consisting of multiple cycles. A shadow part is a core $\mathrm{Core}(\widetilde{b})$ and three solid line cycles define its boundary. We assume here that the outmost cycle is a core cycle; then the cycle with a dashed line is its associated core d-cycle. One of the two subplanes divided by the core d-cycle that contains boss $\widetilde{b}$ is regarded as the inside of the core d-cycle.

Figure 3.6: An example of a core, a core cycle, and a core d-cycle

**Definition 3.1.8.** *For each core* $\mathrm{Core}(\widetilde{b})$*, its* core cycle *is a cycle such that (i) it is a part of the boundary of the region* $\mathrm{Core}(\widetilde{b})$*, and (ii) its exterior has at least* $2\widetilde{n}/3$ *d-vertices. The* core d-cycle *of* $\mathrm{Core}(\widetilde{b})$ *is a subgraph of* $\widetilde{G}$ *induced by the set of d-vertices in* $\mathrm{Core}(\widetilde{b})$ *(i.e., faces of G) sharing an edge with the core cycle. The* inside *of the core d-cycle is the side with boss d-vertex* $\widetilde{b}$*.*

**Remark.** *As explained above, we may assume that each core has a unique core cycle. It is also easy*[4] *to show that a core d-cycle is indeed a d-cycle.*

The following size bounds are immediate from the definition.

**Fact 8.** *Each core d-cycle has at most* $\sqrt{k}$ *d-vertices and at most* $\widetilde{n}/3$ *d-vertices in its inside.*

We define the notions of neck, floor, and ceiling precisely. The main difference from the above intuitive explanation is that they are defined based on all $k$-neighborhoods instead of a single one. First, our level structure is defined as follows. For any $\ell \geq 1$, let $\mathrm{L}_{\mathrm{nb}}(\ell)$ denote a set of d-vertices $\widetilde{v}$ whose distance from its nearest $k$-neighborhood in $\{\mathrm{N}_k(\widetilde{b})\}_{\widetilde{b}\in\widetilde{I}}$ is $\ell$. More formally, it is defined by

$$\mathrm{L}_{\mathrm{nb}}(\ell) \;=\; \left\{\, \widetilde{v} \,\middle|\, \mathrm{dist}(\widetilde{v}, \widetilde{v}_{\mathrm{nrst}}) = \ell, \text{ where } \widetilde{v}_{\mathrm{nrst}} = \mathrm{nrst}_{\widetilde{v}}(\mathrm{N}_k(\mathrm{boss}(\widetilde{v}))) \,\right\}$$

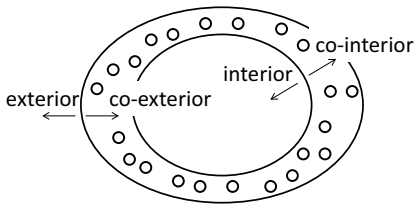
Clearly, a family $\{\mathrm{L}_{\mathrm{nb}}(\ell)\}_{\ell\geq 1}$ is a partition of $\widetilde{V}' := \widetilde{V} \setminus \cup_{\widetilde{b}\in\widetilde{I}} \mathrm{N}_k(\widetilde{b})$. For any d-vertex $\widetilde{v} \in \widetilde{V}'$ (resp., any set $U \subseteq \widetilde{V}'$ of d-vertices), the *level* of $\widetilde{v}$ (resp., $U$) is $\ell$ such that $\widetilde{v} \in \mathrm{L}_{\mathrm{nb}}(\ell)$ holds (resp., $U \subseteq \mathrm{L}_{\mathrm{nb}}(\ell)$ holds). We use $\mathrm{L}_{\mathrm{nb}}(\ell)$ also to denote a subgraph of $\widetilde{G}$ induced by $\mathrm{L}_{\mathrm{nb}}(\ell)$, which we call a *sliced graph* of level $\ell$.

We first note the following properties of sliced graphs.

**Lemma 3.1.6.** *For any* $\ell \geq 1$*, consider any connected component* $\widetilde{C}$ *of sliced graph* $\mathrm{L}_{\mathrm{nb}}(\ell)$ *that contains some d-vertex adjacent to some d-vertex of level* $\ell + 1$*. Then* $\widetilde{C}$*, as a set of d-vertices, is a separator of* $\widetilde{G}$*.*

[4]The argument is easy and in fact a similar and a bit more complicated case will be shown later for floor and ceiling d-cycles. Thus, we omit the proof here.

*Proof.* Let $\widetilde{C}$ be any connected component considered in the lemma. By definition $\widetilde{C}$ is a region. Assume now that there exists some d-vertex $\widetilde{v} \in \widetilde{C}$ that is adjacent to some d-vertex $\widetilde{u}$ of level $\ell + 1$ Since $C$ is a region, its boundary is a disjoint union of cycles of $G$. Since $\widetilde{C}$ consists of only level $\ell$ d-vertices, each cycle edge is an edge between a pair of d-vertices of level either $\ell - 1$ and $\ell$ or $\ell$ and $\ell + 1$. Let us call the former one an *interior edge* and the latter one an *exterior edge*. We show that no boundary cycle has both interior and exterior boundary edges. Suppose otherwise, there exists a vertex $v$ on the cycle that is incident with both interior and exterior boundary edges. Since $G$ is three regular, this $v$ is incident to three faces, one corresponding to a level $\ell$ d-vertex (of $\widetilde{C}$), one corresponding to a level $\ell - 1$ d-vertex, and one corresponding to a level $\ell + 1$ d-vertex. But this contradicts the definition of level. Thus, $\widetilde{C}$, as a region, has at least two boundary cycles; *interior cycle* consisting of interior edges and *exterior cycle* consisting of exterior edges. Note that they both define two disjoint subplanes with some d-vertices that are clearly separated when $\widetilde{C}$ is removed from $\widetilde{G}$. (As a corollary to the above argument, we can also show that a set of d-vertices sharing edges with the interior cycle (resp., the exterior cycle) forms a d-cycle, which fact will be used in our later discussion.) □




An example of a connected component $\widetilde{C}$ of sliced graph $\mathrm{L}_{\mathrm{nb}}(\ell)$. White vertices are d-vertices of $\widetilde{C}$, and two solid line cycles are interior and exterior cycles of $G$ respectively.

Figure 3.7: A connected component $\widetilde{C}$ of Lemma 3.1.6

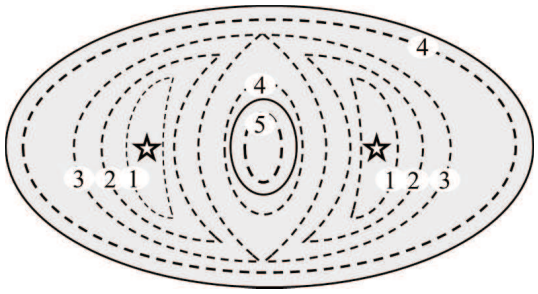
As mentioned in the above proof, any connected component of $\mathrm{L}_{\mathrm{nb}}(\ell)$ is a region. In particular, such a connected component of $\mathrm{L}_{\mathrm{nb}}(\ell)$ is called a *neck* if it has at most $\sqrt{k}$ vertices. We can assume a cycle similar to a core cycle for each neck. Consider any neck $N$ of level $\ell$; note that it is a connected component[5] of $\mathrm{L}_{\mathrm{nb}}(\ell)$. Hence, as argued in the above proof, $N$ has cycles separating $G$ and these cycles are classified into two types: (i) cycles consisting of interior edges, and (ii) cycles consisting of exterior edges. Let $c_1$ be any type (i) cycle of $N$. This $c_1$ separates the plane into two subplanes, one containing $N$ and the other containing d-vertices of level $\ell - 1$. The subplane not containing $N$ is called an *interior*. On the other hand, the other side of $N$ is called an *co-interior*. If the interior part has more than $2\widetilde{n}/3$ d-vertices, then $c_1$ is called a *heavy interior cycle*. Similarly, for any cycle $c_2$ of type (ii), a subplane not containing $N$ is called an *exterior* and the other side of $N$ is called an *co-experior*. And $c_2$ is called a *heavy exterior cycle* if the exterior part has more than $2\widetilde{n}/3$ d-vertices. It may be the case that $N$ has neither heavy interior cycle nor heavy

[5]Though a neck is a connected component of $\widetilde{G}$, we use $N$ instead of $\widetilde{N}$ to denote it because it is regarded as a region in the following discussion.

exterior cycle. But in this case, by the same argument as the core, we can use neck $N$ itself as a separator satisfying our goal theorem. Therefore, we may assume again that each neck has either a heavy interior cycle or a heavy exterior cycle (only one, again from the same reason as core cycle).

**Definition 3.1.9** (floor and ceiling)**.** *Consider any neck $N$. (As discussed above, we may assume that it has either a heavy interior cycle or a heavy exterior cycle.) A* floor cycle *(resp.,* ceiling cycle*) is a heavy exterior cycle (resp., heavy interior cycle) that is not contained in the co-exterior of any other heavy exterior cycle nor in the co-interior of any other heavy interior cycle. A* floor d-cycle *(resp.,* ceiling d-cycle*) is a subgraph of $\widetilde{G}$ induced by a set of d-vertices in $N$ sharing edges with a floor cycle (resp., ceiling cycle). For each floor d-cycle, its* inside *and* outside *are its co-exterior and exterior side respectively. Similarly, for each ceiling d-cycle, its* inside *and* outside *are its co-interior and interior side respectively.*

**Remark.** *It is possible that some heavy interior cycle (resp., exterior cycle) is contained in the co-exterior (resp., co-interior cycle) of some other heavy exterior cycle; see Figure 3.8. From the proof of Lemma 3.1.6 we can guarantee that the set of d-vertices sharing edges with a floor cycle (resp., ceiling cycle) induces a d-cycle.*




An example of the situation explained in the remark of Lemma 3.1.9. Cycles given by dashed lines are necks consisting of d-vertices of levels indicated in the figure. The outmost cycle with a solid line is a heavy exterior cycle of the closest neck (i.e., the level 4 d-cycle just in its inside); that is, we have enough number of d-vertices exist outside of this cycle. The other solid cycle is an interior cycle of a neck defined as the level 5 innermost component. Since there are enough number of d-vertices outside of the outmost cycle, this interior cycle is also a heavy interior cycle. Then a shadow part that is the inside of the outmost heavy exterior cycle contains this heavy interior cycle. Thus, this heavy interior cycle is not a ceiling cycle, while the outmost heavy exterior cycle can be a floor cycle.

Figure 3.8: An example of floor and ceiling cycles

The following size bounds are immediate from the definition.

**Fact 9.** *Each floor and ceiling d-cycle has at most $\sqrt{k}$ d-vertices and at most $\widetilde{n}/3$ d-vertices in its inside.*

### 3.1.3 Modification of $G$

We now explain a modification of $\widetilde{G}$ that is considered as the first step for defining a frame graph. Roughly, we follow the idea explaned above and convert floor and ceiling d-cycles into faces by removing all d-vertices located inside of these d-cycles. Recall that these faces are given weights (in a frame graph) as we discussed; that is, the weight of each face is defined as the proportion of the number of removed d-vertices to $\widetilde{n}$, the total number of vertices of $\widetilde{G}$.

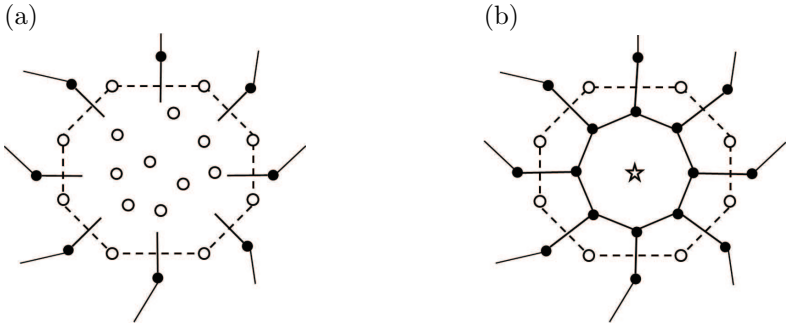
While this modification is enough for $\widetilde{G}$, the corresponding modification on the original graph $G$ is also needed for our discussion. Thus, we define our modification on $G$ (and then define a new $\widetilde{G}$ as its dual graph). Let $\widetilde{f}$ be any one of frame or ceiling d-cycles, e.g., the one given in Figure 3.9 (a). Recall that each d-edge $\widetilde{G}$ corresponds to a unique edge of $G$ that it crosses. For our modification, we first identify all edges corresponding to d-edges of $\widetilde{f}$; let $E_0$ be the set of these edges. Next we remove all vertices of $G$ that are in the inside of $\widetilde{f}$; their incident edges are also removed. On the other hand, we add each edge $e$ of $E_0$ back to $G$ by creating a new vertex as an end vertex of $e$ located inside of $\widetilde{f}$. Then we connect these newly added vertices by edges to form a cycle inside of $\widetilde{f}$; see, e.g., Figure 3.9 (b). This is the modification made on $G$ for each floor and ceiling d-cycle. Let us tentatively use $G'$ to denote the obtained modified graph, and use $\widetilde{G'}$ to denote its dual graph. Note that this modification creates one new face inside of $\widetilde{f}$, which is a d-vertex in $\widetilde{G'}$. Thus, precisely speaking, $\widetilde{f}$ is not a face of $\widetilde{G'}$. In the following explanation, however, we will sometimes regard $\widetilde{f}$ as a face that has one exceptional *center* d-vertex. When defining a frame graph and the weight of its faces, we count all d-vertices removed from $\widetilde{f}$ assuming that they are located in one face incident to the center d-vertex, where the choice of this face is determined by a certain systematic way using, e.g., the indices of the adjacent d-vertices of the center d-vertex.

Due to this modification, we need to revise the notion of Voronoi region accordingly. First note that if a Voronoi region $\mathrm{Vr}(\widetilde{b})$ contains a floor d-cycle $\widetilde{f}$, its boss d-vertex $\widetilde{b}$ that should be included in the inside of $\widetilde{f}$ is removed by our modification. Thus, as its replacement, we will use the center d-vertex of $\widetilde{f}$ as a new boss d-vertex. It is also clear that each Voronoi region $\mathrm{Vr}(\widetilde{b})$ (as a set of d-vertices) should be changed due to our modification. Let us tentatively use $\mathrm{Vr}'(\widetilde{b}')$ to denote this new set of d-vertices (with a new boss d-vertex $\widetilde{b}'$). Below we will further revise Voronoi boundaries. (Note that the graph $G'$ itself will not be changed in the following modification.)

First consider floor d-cycles. We note that each Voronoi region intersects at most one floor d-cycle.

**Lemma 3.1.7.** *Every Voronoi region* $\mathrm{Vr}(\widetilde{b})$*, as a set of d-vertices, intersects at most one floor d-cycle. On the other hand, it is possible that more than two Voronoi regions intersects the same floor d-cycle; any two Voronoi regions intersecting the same floor d-cycle share at least one boundary edge.*

*Proof.* By definition each Voronoi region contains exactly one $k$-neighborhood, and each floor d-cycle has at least one $k$-neighborhood in its inside. Then the first part of the lemma follows. Recall that the degree of every vertex of $G$ is three. Thus, if two




An example of floor or ceiling d-cycles and its modification. (a) A cycle given by a dashed line is a floor or ceiling d-cycle $\widetilde{f}$ having several d-vertices in its inside. Solid edges denote edges kept in $E_0$ that cross d-edges of $\widetilde{f}$. (b) All vertices (and hence, d-vertices) in the inside of $\widetilde{f}$ are removed, and new vertices corresponding to edges of $E_0$ and a cycle connecting them are introduced. The center d-vertex that corresponds to this cycle face is indicated by a star.

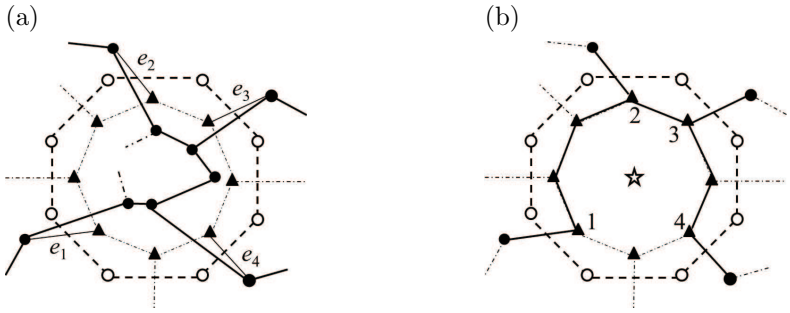
Figure 3.9: Modification of a floor/ceiling d-cycle

Voronoi region shares some vertex (on their boundaries) it should also share at least one edge (of their boundaries). From this the second part of the lemma follows. □

We want each Voronoi region to have one floor d-cycle. First consider the case where several Voronoi regions (of the old type) intersects with some floor d-cycle $\widetilde{f}$. In the modified graph $G'$ we merge all these Voronoi regions into one. It is easy to see that the merged Voronoi regions is a region since as stated in the above lemma, two Voronoi regions intersecting to the same floor d-cycle share at least one boundary edge. Also from the above lemma, we see that $\widetilde{f}$ is the unique floor d-cycle that is contained in this new Voronoi region. Thus, by applying this merger in $G'$, we may assume that every floor d-cycle is contained in one Voronoi region. Note that there is still the case where no floor d-cycle can be defined around some boss d-vertex. In this case we use a core d-cycle as a floor d-cycle, and the same modification for floor d-cycles are applied to the core d-cycle and its inside. In the following, such a core d-cycle is also called, for simplicity, a *floor d-cycle*. (Note that for a Voronoi region that contains a floor d-cycle its core d-cycle is contained inside of the floor d-cycle.) Now we can assume one-to-one mapping between Voronoi regions and floor d-cycles, and we can use the center d-vertex of each floor d-cycle as a new boss d-vertex for each Voronoi region; that is, we can use the set $\widetilde{I}'$ of center d-vertices of floor d-cycles as the set of boss d-vertices of new Voronoi regions.

Next consider ceiling d-cycles. It may occur that some ceiling d-cycle (i.e., its face) overlaps with Voronoi boundaries. Let $\widetilde{f}$ be a ceiling d-cycle that contains some parts of Voronoi boundaries inside of $\widetilde{f}$; see, e.g., Figure 3.10 (a). As we discussed, we modify $G$ so that the inside of $\widetilde{f}$ is replaced by one cycle whose vertices correspond

to d-edges of $\widetilde{f}$; let us call this cycle "inner cycle" and these vertices "inner vertices." Thus, for each Voronoi boundary that goes inside of $\widetilde{f}$, there exists an exactly one corresponding edge of $G'$; let us call such edges "portal edges." Note that portal edges are connected to some inner vertices in $G'$, which we call "portal vertices." Fix any simple and systematic way to give an orientation to the inner cycle, and we give an order to portal vertices. Then we can replace the part of Voronoi boundaries inside of $\widetilde{f}$ with (i) edges of the inner cycle from the first portal vertex to the last portal vertex, and (ii) portal edges connecting these portal vertices. It is easy to see that this change does not affect the set of d-vertices in each Voronoi region $\mathrm{Vr}'(\widetilde{b}')$ except for adding one center d-vertex to some regions. Now by $\{\mathrm{Vr}'(\widetilde{b}')\}_{\widetilde{b}'\in\widetilde{I}'}$ we denote the family of Voronoi regions revised in this way.




A figure showing a way of modifying Voronoi boundaries overlapped with some ceiling d-cycle $\widetilde{f}$. (a) A cycle given by a dashed line is a ceiling d-cycle $\widetilde{f}$, and a cycle with black triangles is its inner cycle. Bold solid edges denote edges on Voronoi boundaries of $G$. Edges labeled by $e_1, \ldots, e_4$ are portal edges. (b) Bold edges consisting of a part of the inner cycle and portal edges are a new part of Voronoi boundaries inside of $\widetilde{f}$.

Figure 3.10: A ceiling d-cycle overlapping with some Voronoi boundaries

Finally we state the following key property.

**Lemma 3.1.8.** *Consider any Voronoi region* $\mathrm{Vr}'(\widetilde{b}')$ *of* $G'$ *with a boss d-vertex* $\widetilde{b}'$. *For any d-vertex* $\widetilde{v}$ *in* $\mathrm{Vr}'(\widetilde{b}')$, *there exists a d-path in* $\mathrm{Vr}'(\widetilde{b}')$ *of length* $\leq 4\sqrt{k}+2$ *from* $\widetilde{b}'$ *to* $\widetilde{v}$.

*Proof.* Let $\widetilde{f}$ be the floor d-cycle of $\mathrm{Vr}'(\widetilde{b}')$ that has $\widetilde{b}'$ as its center d-vertex. For simplicity let us assume that $\widetilde{v}$ is not a newly added center d-vertex, and we consider the original Voronoi region $\mathrm{Vr}(\widetilde{b})$ that contains $\widetilde{v}$. For the lemma it suffices to show that the distance from $\widetilde{f}$ to $\widetilde{v}$ in $\mathrm{Vr}(\widetilde{b})$ is at most $4\sqrt{k}$.

Assume to the contrary that the shortest path length between $\widetilde{b}$ and $\widetilde{v}$ is $\geq 4\sqrt{k}+1$, and we derive a contradiction. Consider any one of the shortest paths from $\widetilde{b}$ to $\widetilde{v}$, and let $(\widetilde{v}_1, \ldots, \widetilde{v}_\ell)$ be its tail subpath where $\widetilde{v}_\ell = \widetilde{v}$ and $\widetilde{v}_1$ is the first d-vertex on the path from $\widetilde{b}$ that is not in $\mathrm{N}_k(\widetilde{b})$ nor inside of $\widetilde{f}$. If $\widetilde{f}$ is not a core d-cycle, then

$\widetilde{v}_1$ is simply a vertex of $\widetilde{f}$ and we have $\ell \geq 4\sqrt{k}$. On the other hand, if $\widetilde{f}$ is originally a core d-cycle (for the case that $\mathrm{Vr}(\widetilde{b})$ has no original floor d-cycyle), then $\widetilde{v}_1$ is a d-vertex with $\mathrm{dist}(\widetilde{b}, \widetilde{v}_1) = \mathrm{d}_{\mathrm{nb}}(\widetilde{b}) + 1$; that is, it is a level 1 d-vertex. Also note that since $\mathrm{d}_{\mathrm{nb}}(\widetilde{b}) - \mathrm{d}_{\mathrm{core}}(\widetilde{b}) \leq \sqrt{k}$ (Lemma 3.1.5), we have $\ell \geq 4\sqrt{k} + 1 - \sqrt{k} - 1 = 3\sqrt{k}$.

Since $(\widetilde{v}_1, \ldots, \widetilde{v}_\ell)$ is a part of a shortest path from $\widetilde{b}$ to $\widetilde{v}$, the level of $\widetilde{v}_i$'s increases one by one when $i$ increases. In fact, since all $\widetilde{v}_i$'s are in $\mathrm{Vr}(\widetilde{b})$, we have $\widetilde{v}_i \in \mathrm{L}_{\mathrm{nb}}(\ell_1 + (i-1))$, where $\ell_1$ is the level of $\widetilde{v}_1$. For each $i$, $1 \leq i < \ell$, let $\widetilde{C}_i$ be a connected component of a sliced graph containing $\widetilde{v}_i$. Note first that $\widetilde{C}_i$ is not a neck. If it were a neck, then it would be a floor d-cycle or ceiling d-cycle. But $\widetilde{C}_i$ cannot be a floor d-cycle because if so, it would contain $\widetilde{f}$ in its inside, contradicting the fact that $\widetilde{f}$ is a floor d-cycle. On the other hand, $\widetilde{C}_i$ cannot be a ceiling d-cycle because if so, $\widetilde{v}$ that has a higher level should be inside of this ceiling d-cycle, which contradicts to our choice of $\widetilde{v}$. Thus, $\widetilde{C}_i$ is not a neck, and hence we have $|\widetilde{C}_i| > \sqrt{k}$.

For each $i$, $1.5\sqrt{k} \leq i \leq 2.5\sqrt{k}$, we let $\widetilde{C}'_i$ be a set of $\sqrt{k}$ d-vertices of $\widetilde{C}_i$ that are nearest to $\widetilde{v}_i$, and let $\widetilde{C}' = \cup_{1.5\sqrt{k} \leq i \leq 2.5\sqrt{k}} \widetilde{C}'_i$. Then we have $|\widetilde{C}'| > k$ since $|\widetilde{C}'_i| = \sqrt{k} + 1$. Note also that the distance between any d-vertex of $\widetilde{C}'_i$ and $\widetilde{v}_i$ is at most $\sqrt{k}$. Now let $\widetilde{B}$ be a set of d-vertices whose distance from $\widetilde{v}_{2\sqrt{k}}$ is at most $1.5\sqrt{k}$. Then we have $\widetilde{B} \supset \widetilde{C}'$, and hence we have $|\widetilde{B}| > k$. On the other hand, since $\mathrm{N}^+_k(\widetilde{b})$ is the nearest $k$-neighborhood$^+$ of $\widetilde{v}_{2\sqrt{k}}$ and the distance between $\widetilde{v}_{2\sqrt{k}}$ and $\mathrm{N}^+_k(\widetilde{b})$ is at least $2\sqrt{k}$, no d-vertex of $\widetilde{B}$ belongs to $\mathrm{N}^+_k(\widetilde{b}'')$ for any $\widetilde{b}'' \in \widetilde{I}$. That is, $\mathrm{N}^+_k(\widetilde{v}_{2\sqrt{k}})$ has no intersection with $\cup_{\widetilde{b}'' \in \widetilde{I}} \mathrm{N}^+_k(\widetilde{b}'')$, contradicting the choice of $\widetilde{I}$. □

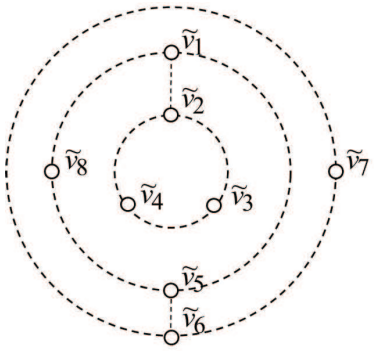
As we will confirm in Section 4, the modified graph $G'$ and its dual graph $\widetilde{G'}$ are computable in $O(\log n)$-space. Thus, in the next section, we will assume that the process of modifying a given input graph to $G'$ and obtain $\widetilde{G'}$ and that these graphs are given as an input. For simplifying notation, let us use from now on $G$ and $\widetilde{G}$ for these modified graphs. Also we use $\widetilde{I}$ and $\{\mathrm{Vr}(\widetilde{b})\}_{\widetilde{b} \in \widetilde{I}}$ for the revised set of boss d-vertices and family of Voronoi regions.

### 3.1.4 Frame Graph

We define a frame graph $\widetilde{H}$ satisfying conditions (F1) $\sim$ (F4) from our dual graph $\widetilde{G}$. Our task is to identify d-cycles of $\widetilde{G}$ that define faces of $\widetilde{H}$ by removing d-vertices inside of these d-cycles.

In order to discuss our face construction, it would be easier if we consider each d-cycle with an orientation. Furthermore, we would sometimes need a bit more generalized notion. For this we introduce here the notion of "multiple d-cycle with an orientation" (in short, m.d-cycle). An m.d-cycle $\widetilde{c}$ is a sequence of d-vertices connected with d-edges that defines nonoverlapping subplane(s) under the assumed planar embedding. Intuitively, an m.d-cycle is simply a collection of incident d-cycles; see Figure 3.11 for examples. Formally we have the following definition.

**Definition 3.1.10** (m.d-cycle)**.** *An* m.d-cycle $\widetilde{c}$ *(under clock/anticlock-wise order) is a sequence of d-vertices satisfying the following conditions: (i) it starts and ends*




An example of an m.d-cycle. White vertices are d-vertices that are connected by d-edges indicated by dashed lines. We may specify an m.d-cycle by $(\widetilde{v}_1, \widetilde{v}_2, \widetilde{v}_3, \widetilde{v}_4, \widetilde{v}_2, \widetilde{v}_1, \widetilde{v}_5, \widetilde{v}_6, \widetilde{v}_7, \widetilde{v}_6, \widetilde{v}_5, \widetilde{v}_8, \widetilde{v}_1)$, traversing d-vertices under the clockwise order. Then the set of d-edges appearing once forms d-cycles while the set of duplicate d-edges forms d-paths. Note, on the other hand, a sequence by $(\widetilde{v}_1, \widetilde{v}_2, \widetilde{v}_3, \widetilde{v}_4, \widetilde{v}_2, \widetilde{v}_1, \widetilde{v}_8, \widetilde{v}_5, \widetilde{v}_6, \widetilde{v}_7, \widetilde{v}_6, \widetilde{v}_5, \widetilde{v}_1)$ consisiting of the same set of d-vertices is not an m.d-cycle.

Figure 3.11: An example of an m.d-cycle

*with the same d-vertex, (ii) every pair of d-vertices appearing next each other in $\widetilde{c}$ is adjacent in $\widetilde{G}$, and (iii) for any d-vertex $\widetilde{v}$ that appears more than once in $\widetilde{c}$, it must appear in a subsequence $(\widetilde{u}, \widetilde{v}, \widetilde{w})$ in $\widetilde{c}$ so that $\widetilde{w}$ is clockwise (resp., anticlockwise) the next d-vertex of $\widetilde{u}$ among adjacent d-vertices of $\widetilde{v}$ appearing in $\widetilde{c}$ under the embedding of $\widetilde{G}$.*

**Remark.** *We may naturally regard an m.d-cycle as a sequence of directed d-edges following its order.*

Consider any m.d-sequence $\widetilde{c}$. As mentioned above, we regard it as a sequence of directed d-edges. It may be possible that some d-edge of $\widetilde{G}$ appears more than once in $\widetilde{c}$, which we call a *duplicate d-edge*; but it is easy to see from the definition that any duplicate d-edge $\widetilde{e} = \{\widetilde{u}, \widetilde{v}\}$ must appear exactly twice as two directed edges $(\widetilde{u}, \widetilde{v})$ and $(\widetilde{v}, \widetilde{u})$ (because otherwise, the clockwise/anticlockwise order would create an infinite loop). Also it is easy to see (Figure 3.11) that duplicate d-edges of $\widetilde{c}$ form d-paths (which we will refer as the *line part*) and that the other d-edges of $\widetilde{c}$ form d-cycles. Let us ignore the situation where $\widetilde{c}$ consists of only duplicate d-edges. Then $\widetilde{c}$ has some cycle(s) and the plane (where $\widetilde{G}$ is embedded) is separated by removing $\widetilde{c}$. We would like to classify these subplances as "inside" or "outside" of $\widetilde{c}$. Intuitively speaking, we define the inside/outside of $\widetilde{c}$ by the set of d-triangles located left/right of directed d-edges of $\widetilde{c}$. (Recall that $\widetilde{G}$ is triangulated.)

Define this notion formally. Consider the planar embedding of $\widetilde{G}$. For any d-edge of $\widetilde{e}$ of $\widetilde{c}$ that appears once in $\widetilde{c}$, we have two d-triangles sharing $\widetilde{e}$, which we call *adjacent* to $\widetilde{e}$ (and *adjacent* to $\widetilde{c}$). Among these two adjacent d-triangles, the one located left (resp., right) of $\widetilde{e}$ w.r.t. the direction of $\widetilde{e}$ is called *left adjacent* (resp., *right adjacent*). In general, for any d-triangle (that is not adjacent to $\widetilde{c}$) we say it is in the *left* (resp., *right*) of $\widetilde{c}$ if its d-vertex is connected to some left adjacent (resp., right adjacent) d-triangle. Note that a d-triangle with a duplicate d-edge of $\widetilde{c}$ is not considered adjacent to $\widetilde{c}$; thus, its side is determined by the connectivity to some adjacent d-triangle. From the way to determine the order of d-vertices of m.d-cycles we have the following properties.

**Lemma 3.1.9.** *Consider any m.d-cycle $\widetilde{c}$ of $\widetilde{G}$. W.r.t. $\widetilde{c}$, the side of every d-triangle of $\widetilde{G}$ is uniquely determined as left or right. For any duplicate d-edge of $\widetilde{c}$, the two d-triangles sharing this d-edge is in the same side of $\widetilde{c}$; that is, they are both in the left (or in the right) of $\widetilde{c}$.*

*Proof.* In this proof, d-edges are all directed, and we omit specifying "directed" below.

We show the first assertion; that is, there is no d-vertex of $\widetilde{G} \setminus \widetilde{c}$ that can be regarded both left and right of $\widetilde{c}$. We show this by induction on "multiplicity" of m.d-cycles. The *multiplicity* of $\widetilde{c}$ (denoted by $\mathrm{mlt}(\widetilde{c})$) is defined by

$$\mathrm{mlt}(\widetilde{c}) \;=\; |\widetilde{E}_{\widetilde{c}}| - |\widetilde{V}_{\widetilde{c}}|,$$

where $\widetilde{V}$ and $\widetilde{E}$ denote respectively the d-vertices and d-edges of graph $\widetilde{c}$. Note that each d-vertex occurs at least once in sequence $\widetilde{c}$ as the starting end point of some d-edge of $\widetilde{c}$. Number $\mathrm{mlt}(\widetilde{c})$ is simply the total number of times that each d-vertex occurs after its first occurrence as the starting end point of some d-edge of $\widetilde{c}$. It is easy to see that $\mathrm{mlt}(\widetilde{c}) = 0$ if and only if $\widetilde{c}$ is a simple directed d-cycle. Thus, the first assertion is clear if $\mathrm{mlt}(\widetilde{c}) = 0$ because no d-vertex located left (resp., right) of *d-cycle* $\widetilde{c}$ is connected to some d-vertex that is right (resp., left) adjacent to $\widetilde{c}$ (Fact 1).

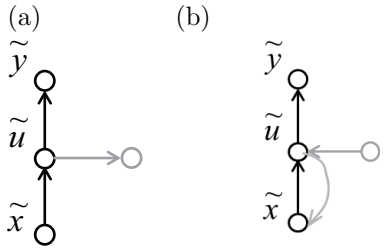
Consider the case where $\mathrm{mlt}(\widetilde{c}) \geq 1$, and let $\widetilde{v}$ be any d-vertex $\widetilde{v}$ that is connected in $\widetilde{G} \setminus \widetilde{c}$ to the top d-vertex (which we denote by $\widetilde{v}'_1$) of some d-triangle right adjacent to $\widetilde{c}$ that has a d-edge $\widetilde{e}_1 = (\widetilde{v}_1, \widetilde{v}_2)$ in $\widetilde{c}$ as nonduplicate d-edge (that is, $(\widetilde{v}_2, \widetilde{v}_1)$ does not appear in $\widetilde{c}$). Our task is to show that it cannot be connected in $\widetilde{G} \setminus \widetilde{c}$ to any d-vertices that is left adjacent to $\widetilde{c}$. For this, we define a sequence $\widetilde{c}_{\widetilde{v}}$ of d-edges that are collected by the following process starting from $\widetilde{e}_1$: for d-edge $\widetilde{e}_i = (\widetilde{v}_i, \widetilde{v}_{i+1})$ last collected, choose $\widetilde{v}_{i+2}$ that is anticlockwise[6] the next among all ending end points of d-edges from $\widetilde{v}_{i+1}$ in $\widetilde{c}$, and collect $\widetilde{e}_{i+1} := (v_{i+1}, v_{i+2})$ into $\widetilde{c}_{\widetilde{v}}$ if it has not been collected in the process (otherwise, terminate the process); for our explanation, let us use $\widetilde{v}^{\mathrm{top}}_{i+1}$ to denote the top d-vertex of the right adjacent d-triangle with d-edge $\widetilde{e}_{i+1}$. We note that this process always terminates by choosing $\widetilde{e}_1$ again. Because if otherwise, there exist three d-edges $(\widetilde{x}, \widetilde{u})$, $(\widetilde{y}, \widetilde{u})$, and $(\widetilde{u}, \widetilde{z})$ in $\widetilde{c}$ such that $(\widetilde{u}, \widetilde{z})$ is anticlockwise the next d-edge from $\widetilde{u}$ from both $(\widetilde{x}, \widetilde{u})$ and $(\widetilde{y}, \widetilde{u})$, which situation we refer by (*). But then in order to meet the condition (iii) of Definition 3.1.10, $\widetilde{c}$ should have either $(\widetilde{u}, \widetilde{x})$ or $(\widetilde{u}, \widetilde{y})$, and the situation (*) would not occur. Thus, $\widetilde{c}_{\widetilde{v}}$ regarded as a sequence of d-vertices is indeed an m.d-cycle. It is also easy to see that top d-vertices $\widetilde{v}'_1, \ldots$ are connected in $\widetilde{G} \setminus \widetilde{c}$; hence, they are all connected to $\widetilde{v}$ in $\widetilde{G} \setminus \widetilde{c}$.

If $\widetilde{c}$ is defined by anticlockwise order, then $\widetilde{c}_{\widetilde{v}}$ must be the same as $\widetilde{c}$. Otherwise, it is easy to see that $\widetilde{c}_{\widetilde{v}}$ is a strict subgraph of $\widetilde{c}$.

Consider the case where $\widetilde{c}$ is the same directed graph as $\widetilde{c}_{\widetilde{v}}$ (hence $\widetilde{c}$ is defined by anticlockwise order). Consider any d-vertex $\widetilde{u}$ that appears in $\widetilde{c}_{\widetilde{v}}$ for more than once; let $(\widetilde{v}_{i-1}, \widetilde{v}_i)$ be the first directed d-edge of $\widetilde{c}_{\widetilde{v}}$ such that $\widetilde{v}_i = \widetilde{u}$, and let $(\widetilde{v}_j, \widetilde{v}_{j+1})$ be anticlockwise the next d-edge of $(\widetilde{v}_i, \widetilde{v}_{i+1})$ in $\widetilde{c}_v$ such that $\widetilde{v}_j = \widetilde{u}$. Then clearly, both subsequences $\widetilde{c}_1 = (\widetilde{v}_1, \ldots, \widetilde{v}_i, \widetilde{v}_{j+1}, \ldots)$ and $\widetilde{c}_2 = (\widetilde{v}_i, \widetilde{v}_{i+1}, \ldots, \widetilde{v}_j)$ are m.d-cycles splitting $\widetilde{c}_{\widetilde{v}}$. Furthermore, we have $\mathrm{mlt}(\widetilde{c}_1), \mathrm{mlt}(\widetilde{c}_2) < \mathrm{mlt}(\widetilde{c}_{\widetilde{v}})$ (Claim **??** below).

[6]This choice becomes "clockwise" when we consider $v$ that is connected to some d-vertex left adjacent to $\widetilde{c}$.

Now consider any directed d-edge $\widetilde{e}$ of $\widetilde{c}$, and let $\widetilde{x}$ and $\widetilde{y}$ be respectively its left and right adjacent d-vertices. Since $\widetilde{e}$ is in $\widetilde{c}_{\widetilde{v}}$, its right adjacent d-vertex $\widetilde{x}$ is connected to $\widetilde{v}$ in $\widetilde{G} \setminus \widetilde{c}$, as we argued above. Also $\widetilde{e}$ must appear in either $\widetilde{c}_1$ or $\widetilde{c}_2$. Let us assume that it appears in $\widetilde{c}_1$. Then by induction it holds that $\widetilde{y}$ is not connected to $\widetilde{v}$ in $\widetilde{G} \setminus \widetilde{c}_1$ because $\widetilde{x}$ is connected to $\widetilde{v}$ in $\widetilde{G} \setminus \widetilde{c}_1$. Therefore $\widetilde{y}$ is not connected to $\widetilde{v}$ in $\widetilde{G} \setminus \widetilde{c}$ either.

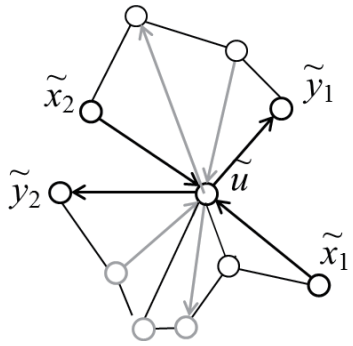



This figure shows two cases that would not occur in $\widetilde{c}_{\widetilde{v}}$. In both graphs black arrows indicate d-edges collected for $\widetilde{c}_{\widetilde{v}}$ and gray arrows indicate d-edges of $\widetilde{c}$ not collected for $\widetilde{c}_{\widetilde{v}}$. (a) If $\widetilde{c}$ had a d-edge indicated by the gray arrow, then this d-edge would have been collected instead of $(\widetilde{u}, \widetilde{y})$. (b) If $\widetilde{c}$ had a d-edge going in to $\widetilde{u}$ like the gray arrow, $\widetilde{c}$ would have $(\widetilde{u}, \widetilde{x})$ indicated by the gray arrow, and if $\widetilde{c}$ had both $(\widetilde{x}, \widetilde{u})$ and $(\widetilde{u}, \widetilde{x})$, then $(\widetilde{u}, \widetilde{x})$ would have been collected for $\widetilde{c}_{\widetilde{v}}$.

Figure 3.12: Examples of some situations that would not occur in $\widetilde{c}_{\widetilde{v}}$

Next consider the case where $\widetilde{c}_{\widetilde{v}}$ is a strict subgraph of $\widetilde{c}$. We first show that any d-edge of $\widetilde{c} \setminus \widetilde{c}_{\widetilde{v}}$ is connected to some d-vertex that is left adjacent to $\widetilde{c}_{\widetilde{v}}$. To see this consider any d-vertex $\widetilde{u}$ of $\widetilde{c}_{\widetilde{v}}$ that is the end point of some d-edges of $\widetilde{c} \setminus \widetilde{c}_{\widetilde{v}}$. Consider all d-edges of $\widetilde{c}$ that has $\widetilde{u}$ as one of their end points; let $\widetilde{E}_{\widetilde{u}}$ (resp., $\widetilde{E}^{-}_{\widetilde{u}}$) be the set of such d-edges that belong to (resp., do not belong to) $\widetilde{c}_{\widetilde{v}}$. Consider any pair of d-edges $\widetilde{e}_1 = (\widetilde{x}, \widetilde{u})$ and $\widetilde{e}_2 = (\widetilde{u}, \widetilde{y})$ of $\widetilde{E}_{\widetilde{u}}$ that appear consecutively in $\widetilde{c}_{\widetilde{v}}$. From the construction of $\widetilde{c}_{\widetilde{v}}$, $\widetilde{c}$ has no d-edge going out from $\widetilde{u}$ in the subplane from $\widetilde{e}_1$ to $\widetilde{e}_2$ anticlockwise (see Figure 3.12 (a)). Then in the same subplane there should not exist any d-edge of $\widetilde{c}$ going in to $\widetilde{u}$ either; because if otherwise (e.g., Figure 3.12 (b)), then $\widetilde{c}$ must have d-edge $\widetilde{e}'_1 := (\widetilde{u}, \widetilde{x})$ but this $\widetilde{e}'_1$ should have been collected to $\widetilde{c}_{\widetilde{v}}$ right after $\widetilde{e}_1$. Thus, every d-edge of $\widetilde{E}^{-}_{\widetilde{u}}$ must be located in a subplane from some d-edge of $\widetilde{E}_{\widetilde{u}}$ going in to $\widetilde{u}$ to the clockwise adjacent d-edge of $\widetilde{E}_{\widetilde{u}}$ that should be in fact a d-edge going out from $\widetilde{u}$ (see Figure 3.13). Consider d-edges of $\widetilde{E}^{-}_{\widetilde{u}}$ that belong to any one of such subplanes between, say, $\widetilde{e}_{\text{in}}$ and $\widetilde{e}_{\text{out}}$ of $\widetilde{c}_{\widetilde{v}}$. Then it is easy to see that all their end points opposite to $\widetilde{u}$ are connected in $\widetilde{G} \setminus \widetilde{c}$, and they are all connected to the d-vertex left adjacent to $\widetilde{e}_{\text{in}}$. Thus, any d-edge of $\widetilde{c} \setminus \widetilde{c}_{\widetilde{v}}$ is connected to some d-vertex that is left adjacent to $\widetilde{c}_{\widetilde{v}}$.

Once we have the above property for all d-edges of $\widetilde{c} \setminus \widetilde{c}_{\widetilde{v}}$, then the rest of the argument is similar to the previous case. Note that $\text{mlt}(\widetilde{c}_{\widetilde{v}}) < \text{mlt}(\widetilde{c})$ (Claim **??**




In this figure black arrows (resp., gray arrows) indicate d-edges of $\widetilde{E}_{\widetilde{u}}$ (resp., $\widetilde{E}^-_{\widetilde{u}}$), and lines indicate the other d-edges of $\widetilde{G}$. We consider the situation where pairs of d-edges $(\widetilde{x}_i, \widetilde{u})$ and $(\widetilde{u}, \widetilde{y}_i)$ (where $i \in \{1, 2\}$) are collected consecutively for $\widetilde{c}_{\widetilde{v}}$. Then no d-edges of $\widetilde{E}^-_{\widetilde{u}}$ exist in the subspaces between $(\widetilde{x}_i, \widetilde{u})$ and $(\widetilde{u}, \widetilde{y}_i)$ anticlockwise, and they should be arranged either in the subspace between $(\widetilde{x}_1, \widetilde{u})$ and $(\widetilde{u}, \widetilde{y}_2)$ clockwise or in the subspace between $(\widetilde{x}_2, \widetilde{u})$ and $(\widetilde{u}, \widetilde{y}_1)$ clockwise; in other words, they are all located "left" of these pairs of d-edges.

Figure 3.13: Arrangements of d-edges of $\widetilde{c} \setminus \widetilde{c}_{\widetilde{v}}$

below). Thus, by induction, our assertion holds for $\widetilde{c}_{\widetilde{v}}$. Consider any directed d-edge $\widetilde{e}$ of $\widetilde{c}$. If $\widetilde{e}$ is in $\widetilde{c}_{\widetilde{v}}$, then by induction hypothesis its left adjacent d-vertex is not connected to $\widetilde{v}$ in $\widetilde{G} \setminus \widetilde{c}_{\widetilde{v}}$. Hence, it is not connected to $\widetilde{v}$ in $\widetilde{G} \setminus \widetilde{c}$. On the other hand, if $\widetilde{e}$ is not in $\widetilde{c}_{\widetilde{v}}$. Then as argued above $\widetilde{e}$ (and its right adjacent d-vertex) is connected to some left adjacent d-vertex of $\widetilde{c}_{\widetilde{v}}$. Thus, again by induction hypothesis it cannot be connected to $\widetilde{v}$ in $\widetilde{G} \setminus \widetilde{c}_{\widetilde{v}}$ and hence in $\widetilde{G} \setminus \widetilde{c}$ either. Therefore the assertion holds.

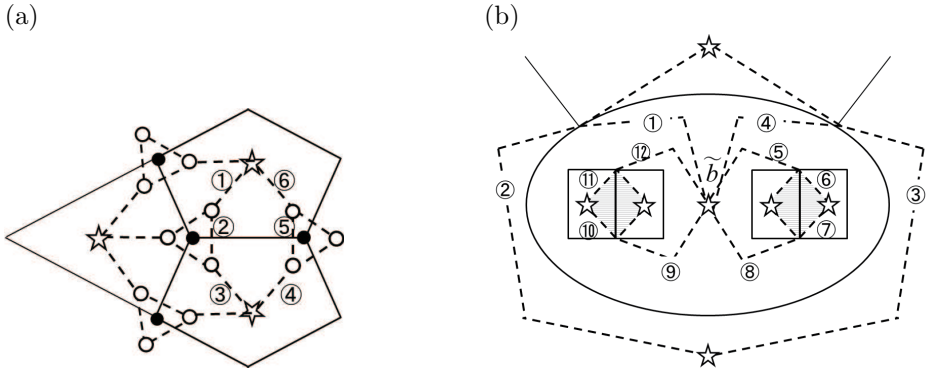
Next we consider the second assertion. Let $\widetilde{e}_+$ and $\widetilde{e}_-$ be a pair of duplicate d-edges in the line part of $\widetilde{c}$; that is, $\widetilde{e}_+ = (\widetilde{x}, \widetilde{y})$ while $\widetilde{e}_- = (\widetilde{y}, \widetilde{x})$. Consider any nonduplicate d-edge $\widetilde{e}$ of $\widetilde{c}$. Then we have two adjacent d-vertices, one is left adjacent to $\widetilde{e}$ and the other is right adjacent to $\widetilde{e}$. Let us assume without losing generality that $\widetilde{c}$ is defined by anticlockwise order. Then $\widetilde{c}_{\widetilde{v}}$ constructed by the above process starting from $\widetilde{e}$ with using the right adjacent d-vertex as $\widetilde{v}$ becomes $\widetilde{c}$. As we argued, all the right adjacent d-vertices of d-edges collected for $\widetilde{c}_{\widetilde{v}}$ are connected to $\widetilde{v}$, which holds also for the right adjacent d-vertices of $\widetilde{e}_+$ and $\widetilde{e}_-$. This means that two adjacent d-triangles of $\widetilde{e}_+$ are the right of $\widetilde{c}$, proving the second assertion. □

**Definition 3.1.11** (inside/outside of m.d-cycle)**.** *For any m.d-cycle, its* inside *(resp.,* outside*) is a set of subplanes of plane graph $\widetilde{G}$ consisting of d-triangles located in the left (resp., right) of $\widetilde{c}$. (Precisely speaking, d-edges of $\widetilde{c}$ are excluded from these subplanes.)*

Our face construction is easy if each Voronoi region has one boundary cycle. We use Figure 3.14 (a) to explain a preliminary idea of our construction for the case each Voronoi region has a single boundary cycle. As indicated by black vertices in the figure, we have a vertex of $G$ where three Voronoi regions meet (or, equivalently three Voronoi boundaries meet); let us *tentatively* call such a vertex a "branch vertex." Each branch vertex has three d-vertices incident to it (white vertices in the figure), which we call "branch d-vertices." Note that these branch d-vertices form a d-triangle, which we call a "branch d-triangle" that is in fact a face of $\widetilde{G}$. Note also that each branch d-vertex has a d-path connecting it and its boss d-vertex that is a part of the BFS d-tree; let us call this d-path a "branch d-path." We use in particular an m.d-cycle

defined by using these branch d-triangles/d-paths as shown in the figure by numbered dashed lines in the order of those numbers, which will be called a "standard" m.d-cycle. Roughly speaking, a frame graph is obtained by removing all d-vertices and d-edges except for such m.d-cycles. Then each standard m.d-cycle and the subplane that it defines becomes a face in the frame graph, which will be called a "standard face."

This construction has a problem if some Voronoi region has more than one boundary cycles; see Figure 3.14 (b) for example. We can still use only branch d-triangles and branch d-paths to define a subgraph of $\widetilde{G}$. But then we may have an m.d-cycle with many branch d-paths, which creates a face whose boundary size cannot satisfy (F4). Here we introduce a way to separate such m.d-cycles to get around this problem.




This figure shows three types of m.d-cycles. (a) (A part of) a graph consisting of three Voronoi regions each of which has a single boundary cycle that is shown by solid lines. Black vertices are branch vertices and white vertices are branch d-vertices. Dashed lines are either a branch d-triangle or a branch d-path, which will be used to define our frame graph. Here we have m.d-cycles of two types: a branch d-triangle and a "standard" m.d-cycle. An example of the latter is indicated by six numbered dashed lines oriented in the order of their numbers. (b) Solid lines are Voronoi boundaries. We omit here showing vertices, d-vertices, and branch d-triangles except for boss d-vertices. The outmost solid cycle is one of the boundary cycles of Voronoi region $\mathrm{Vr}(\widetilde{b})$. This $\mathrm{Vr}(\widetilde{b})$ contains four Voronoi regions and it has four boundary cycles between them. Dashed lines are branch d-paths. There are some "standard" m.d-cycles that defines "standard faces"; for example, those indicated by shadow parts are "standard faces." On the other hand, the graph has an m.d-cycle consisting of twelve branch d-paths labeled by numbers. This is an m.d-cycle of the third type, which is regard as a "nonstandard" m.d-cycle. The problem is that we cannot bound the length of such an m.d-cycle that is the face size in the frame graph.

Figure 3.14: Three kinds of m.d-cycles

Consider any $\widetilde{b} \in \widetilde{I}$ such that Voronoi region $\mathrm{Vr}(\widetilde{b})$ has more than one boundary cycles. Let us fix these $\widetilde{b}$ and $\mathrm{Vr}(\widetilde{b})$ for a while. Also we fix any BFS d-tree on $\mathrm{Vr}(\widetilde{b})$ with depth $O(k)$; note that it is a spanning tree of d-vertices in $\mathrm{Vr}(\widetilde{b})$. Consider any two adjacent d-vertices $\widetilde{u}$ and $\widetilde{v}$ that are not connected in the BFS d-tree. Then a d-cycle is created in the BFS d-tree by connecting $\widetilde{u}$ and $\widetilde{v}$ by $\widetilde{e} = (\widetilde{u}, \widetilde{v})$. This cycle divides the plane into two subplanes. We consider the case where this separation divides the set of boundary cycles of $\mathrm{Vr}(\widetilde{b})$.

**Definition 3.1.12** (ridge edge)**.** *Consider* $\mathrm{Vr}(\widetilde{b})$ *as a subgraph of* $G$*. An edge* $e$ *of* $\mathrm{Vr}(\widetilde{b})$ *is a* ridge edge *if the d-edge* $\widetilde{e} = \{\widetilde{u}, \widetilde{v}\}$ *of* $\mathrm{Vr}(\widetilde{b})$ *crossing* $e$ *satisfies the following:*

- $\widetilde{e}$ *is not a d-edge on the BFS d-tree of* $\mathrm{Vr}(\widetilde{b})$*.*
- *Let* $\widetilde{c}(\widetilde{e})$ *be the d-cycle induced by* $\widetilde{e}$ *and two d-paths of the BFS d-tree of* $\mathrm{Vr}(\widetilde{b})$ *containing* $\widetilde{u}$ *and* $\widetilde{v}$ *respectively. Then* $\widetilde{c}(e)$ *divides* $\mathrm{Vr}(\widetilde{b})$ *into two separated subplanes, each of which contains at least one boundary cycle of* $\mathrm{Vr}(\widetilde{b})$*.*

Below we use $R_{\widetilde{b}}$ to denote the set of ridge edges in $\mathrm{Vr}(\widetilde{b})$. Use $B_{\widetilde{b}}$ to denote the set of boundary edges of $\mathrm{Vr}(\widetilde{b})$. These sets are sometimes regarded as subgraphs of $G$.

We first point out a simple property that is immediate from the definition: namely, no BFS d-path crosses $R_{\widetilde{b}} \cup B_{\widetilde{b}}$. We then show the following two properties of $R_{\widetilde{b}}$ and $B_{\widetilde{b}}$.

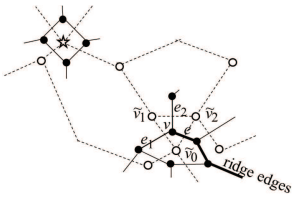
**Lemma 3.1.10.** *$R_{\widetilde{b}}$ as a subgraph of $G$ is a forest. Furthermore, every leaf of $R_{\widetilde{b}}$ is incident to some boundary edge of $B_{\widetilde{b}}$.*

*Proof.* Consider any connected component $T$ of $R_{\widetilde{b}}$. We first show that $T$ does not contain a cycle. If $T$ had a cycle, then the cycle divides $\mathrm{Vr}(\widetilde{b})$ into more than two subplanes each of which contains at least one d-vertex (since d-vertices correspond to faces of $G$). From the definition of a ridge edge, any d-edge on the BFS d-tree of $\mathrm{Vr}(\widetilde{b})$ cannot cross a ridge edge; hence, there must be a subplane whose d-vertices are not in the BFS d-tree. This contradicts to the fact that the BFS d-tree is a spanning tree of $\mathrm{Vr}(\widetilde{b})$.

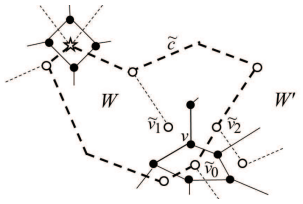
Next we show that every leaf of $T$ is incident to the boundary of $\mathrm{Vr}(\widetilde{b})$. Here we assume otherwise; that is, there is some leaf $v$ of a connected component of $T$ that is not incident to the boundary of $\mathrm{Vr}(\widetilde{b})$ and lead the contradiction.

Let $e$ be the ridge edge that is incident to $v$. We consider three faces around $v$ that are regarded as d-vertices $\widetilde{v}_0$, $\widetilde{v}_1$, and $\widetilde{v}_2$ indexed in clockwise starting from the one adjacent to $e$ so that an d-edge $\{\widetilde{v}_0, \widetilde{v}_2\}$ crosses $e$. Let $e_1$ and $e_2$ are edges incident to $v$ that are crossed by edges $\{\widetilde{v}_0, \widetilde{v}_1\}$ and $\{\widetilde{v}_1, \widetilde{v}_2\}$ respectively (Figure 3.15 (a)). Note that neither $e_1$ nor $e_2$ is a ridge edge (because $v$ is a leaf of some connected component consisting of ridge edges).

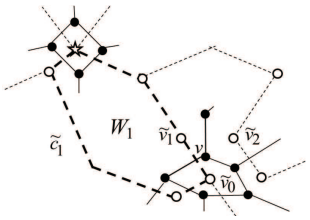
Consider d-cycle $\widetilde{c}$ induced by the BFS d-tree and an d-edge $\{\widetilde{v}_0, \widetilde{v}_2\}$ (Figure 3.15 (b)). Cycle $\widetilde{c}$ divides the plane into two subplanes; let $W$ and $W'$ denote them. Note that both contain at least one boundary cycle of $\mathrm{Vr}(\widetilde{b})$ since $e$ is a ridge edge. Without



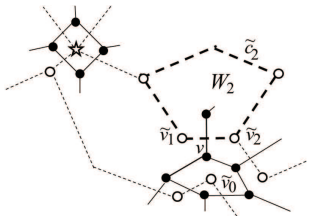

(a) $e$ is the ridge edge and $v$ is its end point connecting $\widetilde{v}_0$ and $\widetilde{v}_2$




(b) d-cycle $\widetilde{c}$ obtained by




(c) d-cycle $\widetilde{c}_1$ obtained by connecting $\widetilde{v}_0$ and $\widetilde{v}_1$ connecting $\widetilde{v}_1$ and $\widetilde{v}_2$




(d) d-cycle $\widetilde{c}_2$ obtained by

Figure 3.15: An example for the proof of Lemma 3.1.10

losing generality, we may assume that $W$ is the subplane containing $v$. Note that $W$ contains $\widetilde{v}_0$, $\widetilde{v}_1$, and $\widetilde{v}_2$.

Consider next d-cycle $\widetilde{c}_1$ induced by the BFS d-tree and an d-edge $\{\widetilde{v}_0, \widetilde{v}_1\}$, and let $W_1$ denote one of the two subplanes divided by $\widetilde{c}_1$ that is entirely contained in $W$ (Figure 3.15 (c)). Note that subplane $W_1$ has no boundary of $\mathrm{Vr}(\widetilde{b})$ because otherwise the other side of $\widetilde{c}_1$, which contains $W'$, has no boundary of $\mathrm{Vr}(\widetilde{b})$ (since $e_1$ is not a ridge edge), a contradicting the above mentioned fact that $W'$ has at least one boundary. Similarly, subplane $W_2$ defined by d-cycle $\widetilde{c}_2$ induced by the BFS d-tree and an edge $\{\widetilde{v}_1, \widetilde{v}_2\}$ (Figure 3.15 (c)) has no boundary of $\mathrm{Vr}(\widetilde{b})$. Note, however, $v$ is only one vertex of $G$ that is in $W$ and that belongs to neither $W_1$ nor $W_2$. Thus, $W$ has no boundary of $\mathrm{Vr}(\widetilde{b})$. A contradiction. □

**Lemma 3.1.11.** *$R_{\widetilde{b}} \cup B_{\widetilde{b}}$ as a graph is connected.*

*Proof.* For proving by contradiction, suppose that $R_{\widetilde{b}} \cup B_{\widetilde{b}}$ is not connected. Let $C$ and $C'$ be two connected components of $R_{\widetilde{b}} \cup B_{\widetilde{b}}$. Here we note that both $C$ and $C'$ have a Voronoi boundary cycle as its subgraph, which is immediate from the from the above lemma.

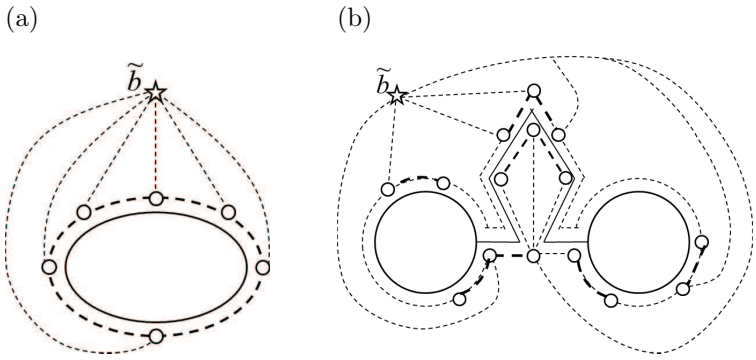
Let $\widetilde{c}$ denote a set of d-vertices of $\mathrm{Vr}(\widetilde{b})$ incident to edges of $C$. Due to the three regularity of $G$, it is easy to see that $\widetilde{c}$ defines an m.d-cycle whose orientation is determined so that $C$ is located in its inside. Note first that no d-vertex of $\mathrm{Vr}(\widetilde{b})$ belongs to the inside of $\widetilde{c}$; if otherwise, i.e., if there were some $\widetilde{v} \in \mathrm{Vr}(\widetilde{b})$ inside of $\widetilde{c}$,

then the BFS path from $\widetilde{b}$ to $\widetilde{v}$ should crosses some edge of $C$, which would not occur by definition. Thus, all d-vertices of $\mathrm{Vr}(\widetilde{b})$ are located in the outside of $\widetilde{c}$.

Among d-edges of $\widetilde{c}$, there must be some d-edges that are not on the BFS d-tree. Let $\widetilde{e}_1, \ldots, \widetilde{e}_h$ denote these d-edges. Consider any $\widetilde{e}_i$ of these d-edges with the orientation of m.d-cycle $\widetilde{c}$. Consider a pair of BFS d-paths from two end points of $\widetilde{e}_i$ to boss d-vertex $\widetilde{b}$. Then this pair of BFS d-paths and edge $\widetilde{e}_i$ defines an m.d-cycle under the orientation consistent with $\widetilde{e}_i$. Let $a(\widetilde{e}_i)$ be the inside of this m.d-cycle. Then we have the following. (The proof, which is intuitively clear, is rather tedious and it is omitted here.)

**Fact 10.** *The Voronoi region* $\mathrm{Vr}(\widetilde{b})$ *is covered by* $a(\widetilde{e}_1), \ldots, a(\widetilde{e}_h)$. *More precisely, all vertices of* $\mathrm{Vr}(\widetilde{b})$ *are located in one of subplanes* $a(\widetilde{e}_1), \ldots, a(\widetilde{e}_h)$.

Thus, there is some $a(\widetilde{e}_i)$ that contains some and hence all vertices of $C'$ because no edges of $C'$ can cross the border (i.e., some BFS d-path) of $a(\widetilde{e}_i)$. Then $\widetilde{e}_i$ satisfies the condition given in Definition 3.1.12. That is, the edge that crosses $\widetilde{e}_i$ is a ridge edge, which contradicts the choice of $\widetilde{e}_i$. □




Examples for Fact 10. In both figures, solid lines indicate edges of $C$. White vertices are d-vertices of the m.d-cycle $\widetilde{c}$ incident to $C$; thin dashed lines are d-path of the assumed BFS d-tree rooted at $\widetilde{b}$, and bold dashed lines are d-edges $\widetilde{e}_1, \ldots, \widetilde{e}_h$ of $\widetilde{c}$ that are not in the BFS d-tree. (a) A simple case where it is easy to see that $C$ is located inside of $\widetilde{c}$ and that the other side of $\widetilde{c}$ is partitioned by $a(\widetilde{e}_1), \ldots, a(\widetilde{e}_s)$. In this example, $\widetilde{e}_1, \ldots, \widetilde{e}_s$ are all and only d-edges of $\widetilde{c}$. (b) A general case where $\widetilde{c}$ has some d-edges of the BFS d-tree, and some BFS d-paths are merged before their root, i.e., boss d-vertex $\widetilde{b}$. But it is still not so difficult to see that $a(\widetilde{e}_1) \cup \cdots \cup a(\widetilde{e}_s)$ covers the outside of $\widetilde{c}$.

Figure 3.16: Examples of m.d-cycle partition

Now we define necessary notions for our frame graph formally. In the following we use $B$ and $R$ to denote respectively the set of all Voronoi boundary edges and ridge edges of $G$. They are also regarded as subgraphs of $G$. From the above two lemmas,

we see that $B \cup R$ forms a connected component. We now introduce the notion of "branch vertex" by including ridge edges, thereby introducing new branch vertices (and hence branch d-paths) that divide the original "nonstandard faces."

**Definition 3.1.13** (branch vertex)**.** *Regard $B \cup R$ as an induced subgraph of $G$. A* branch vertex *is a vertex with degree three in this graph. For each branch vertex $v$, three d-vertices of $\widetilde{G}$ incident to $v$ are* branch d-vertices*, and a triangle consisting of these d-vertices is a* branch d-triangle*. A d-path in the BFS d-tree from its boss to $\widetilde{v}$ is a* branch d-path*.*

**Definition 3.1.14** (connector)**.** *Two branch vertices are* adjacent *if they are connected by edges in $B \cup R$ with no other branch vertex on it. The path connecting adjacent branch vertices is a* connector*.*

**Remark.** *It is easy to see that each connector consists of Voronoi boundary edges only or ridge edges only. Note that it is possible that some branch vertex is adjacent to itself; that is, it is connected to itself by a a cycle connector. We fix some simple way to give a direction to each connector, and in the following, we may assume that each connector has this direction.*

We are ready to define the notion of "standard m.d-cycle", which will be used as a basis for defining "frame graph" and "standard face." We prepare some notation. Consider any connector $p$ (see Figure 3.17). Following the remark of Definition 3.1.14, we may assume that $p$ has a direction, and we use $v_{\mathrm{fst}}$ and $v_{\mathrm{last}}$ be the start and end vertices of $p$ w.r.t. this direction. Note that $v_{\mathrm{fst}}$ and $v_{\mathrm{last}}$ are an adjacent pair of branch vertices. Let us call an edge of $p$ having $v_{\mathrm{fst}}$ as its end point the *first edge*, and similarly, an edge of $p$ having $v_{\mathrm{last}}$ as its end point the *last edge*. Also let $\widetilde{e}_{\mathrm{fst}} = \{\widetilde{v}_{\mathrm{fst.r}}, \widetilde{v}_{\mathrm{fst.l}}\}$ and $\widetilde{e}_{\mathrm{last}} = \{\widetilde{v}_{\mathrm{last.r}}, \widetilde{v}_{\mathrm{last.l}}\}$ be d-edges that cross the first and the last edges respectively, where $\widetilde{v}_{\mathrm{fst.l}}$ (resp., $\widetilde{v}_{\mathrm{last.l}}$ is the one located left of the first (resp., the last) edge, and $\widetilde{v}_{\mathrm{fst.r}}$ (resp., $\widetilde{v}_{\mathrm{last.r}}$) is the one located right of the first (resp., the last) edge.

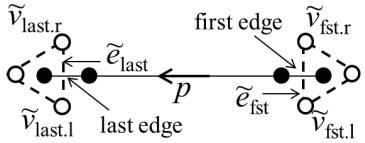



Figure 3.17: Notation for defining a standard m.d-cycle

We can easily see that both $\widetilde{v}_{\mathrm{fst.l}}$ and $\widetilde{v}_{\mathrm{last.l}}$ (similarly, both $\widetilde{v}_{\mathrm{last.r}}$ and $\widetilde{v}_{\mathrm{last.r}}$) have the same boss d-vertex.

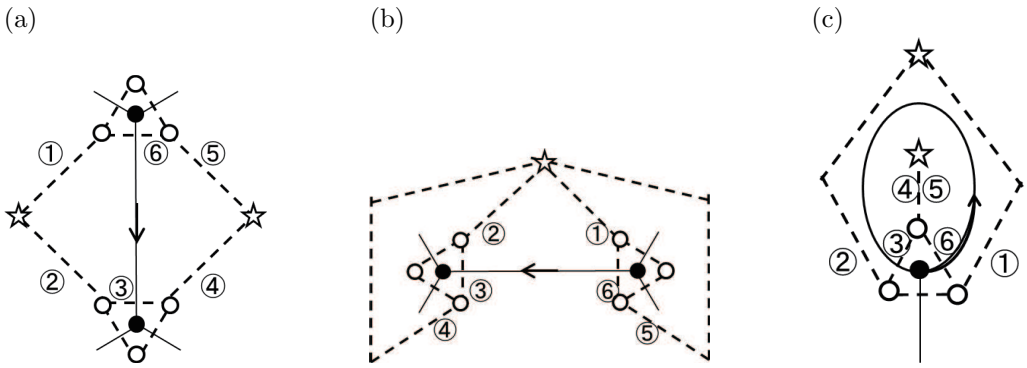
**Lemma 3.1.12.** $\mathrm{boss}(\widetilde{v}_{\mathrm{fst.l}}) = \mathrm{boss}(\widetilde{v}_{\mathrm{last.l}})$ *and* $\mathrm{boss}(\widetilde{v}_{\mathrm{fst.r}}) = \mathrm{boss}(\widetilde{v}_{\mathrm{last.r}})$.

*Proof.* Consider d-vertices incident to $p$ that are located on left side of $p$. We name them $\widetilde{v}_0 = \widetilde{v}_{\mathrm{fst.l}}, \widetilde{v}_1, \ldots, \widetilde{v}_{t-1}, \widetilde{v}_t = \widetilde{v}_{\mathrm{last.l}}$ so that $\widetilde{v}_i$ and $\widetilde{v}_{i+1}$ are adjacent. Assume that there exists $\widetilde{v}_i$ such that $\mathrm{boss}(\widetilde{v}_i) \neq \mathrm{boss}(\widetilde{v}_{i+1})$, then an edge $e$ that crosses $\{\widetilde{v}_i, \widetilde{v}_{i+1}\}$ must be an edge of a Voronoi boundary. But since $e$ is incident to $p$, the end point of $e$ that is on $p$ must be a branch vertex. This contradicts that $p$ is a connecter. Therefore,

there is no $i$ such that $\mathrm{boss}(\widetilde{v}_i) \neq \mathrm{boss}(\widetilde{v}_{i+1})$, and hence $\mathrm{boss}(\widetilde{v}_{\mathrm{fst.l}}) = \mathrm{boss}(\widetilde{v}_{\mathrm{last.l}})$. With a similar argument, we can show $\mathrm{boss}(\widetilde{v}_{\mathrm{fst.r}}) = \mathrm{boss}(\widetilde{v}_{\mathrm{last.r}})$. □

Then the notion of "standard m.d-cycle" is defined as follows.

**Definition 3.1.15** (standard m.d-cycle)**.** *(Use notation defined in Figure 3.17; also see Figure 3.18.) For any connector $p$, a* standard m.d-cycle *(*w.r.t. *$p$) is an m.d-cycle consists of (1) a branch d-path from $\widetilde{v}_{\mathrm{fst.r}}$ to its boss d-vertex, (2) a branch d-path from this boss d-vertex to $\widetilde{v}_{\mathrm{last.r}}$, (3) a branch d-triangle edge $\widetilde{e}_{\mathrm{last}}$ from $\widetilde{v}_{\mathrm{last.r}}$ to $\widetilde{v}_{\mathrm{last.l}}$, (4) a branch d-path from $\widetilde{v}_{\mathrm{last.l}}$ to its boss d-vertex, (5) a branch d-path from this boss d-vertex to $\widetilde{v}_{\mathrm{fst.l}}$, and (6) a branch d-triangle edge $\widetilde{e}_{\mathrm{fst}}$ from $\widetilde{v}_{\mathrm{fst.l}}$ to $\widetilde{v}_{\mathrm{fst.r}}$. The orientation of this m.d-cycle is defined naturally from the above order of paths.*

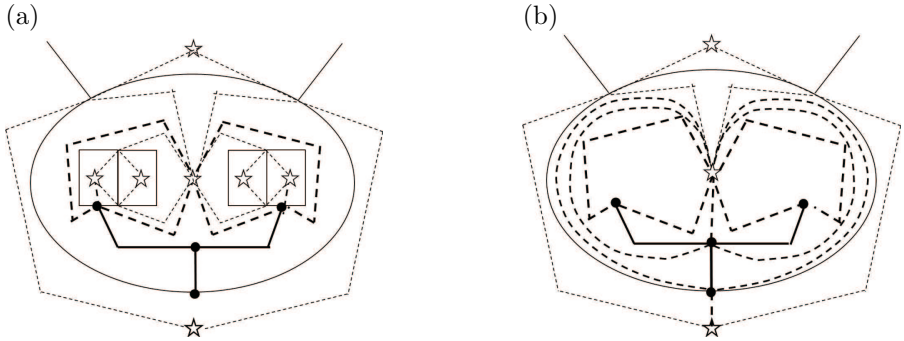



This figure shows three types of standard m.d-cycles. Each m.d-cycle is indicated by dashed lines with numbers that also define the orientation of the m.d-cycle. Under this orientation the inside of the m.d-cycle (i.e., the subplane located in the left side of the m.d-cycle) is regarded as a "face" that it defines. For simplicity we show only the cases where two branch d-paths meet at the corresponding boss d-vertex. In general, they may merge earlier, in which case the corresponding m.d-cycle creates a line part.

Figure 3.18: Standard m.d-cycles of three types

Roughly speaking, our frame graph is defined by removing all d-vertices (and d-edges) not participating in any standard m.d-cycle or branch d-triangle, and intuitively, these standard m.d-cycles define the standard faces of the frame graph. For example, the "standard face" we considered earlier corresponds to the "face" defined by Figure 3.18 (a). On the other hand, the "nonstandard face" is divided into "faces" due to the introduction of new branch vertices; see Figure 3.19 for example.

The actual situation is a bit more complicated in general because some standard m.d-cycle may have a line part; see Figure 3.20 for example. For defining a frame graph, we would like to ignore such line parts. In fact, it is computational easy to identify/ignore such line parts. We only have to ignore duplicate d-edges. This gives one way to define a frame graph.




An example of new branch vertices that introduce new "faces." We use the Voronoi region of Figure 3.14 (b). Solid lines are three boundary cycles of the Voroi region. Bold solid lines are paths consisting of ridge edges connecting these three boundary cycles. Black vertices are new branch vertices, from which new branch d-vertices (not shown here) and new branch d-triangles (not shown here) are created. Also new branch d-paths are introduced, and, for visibility, only two of them are shown as as bold dashed lines in (a) while all of them are shown as bold dashed lines in (b). We can see that the "nonstandard face" of Figure 3.14 (b) is now separated into "faces" by standard m.d-cycles.

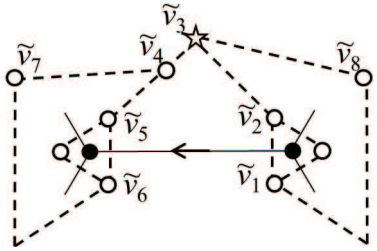
Figure 3.19: New branch vertices and standard m.d-cycles

**Definition 3.1.16** (frame graph)**.** *For each standard m.d-cycle $\widetilde{c}$ of $\widetilde{G}$, let $\widetilde{E}_{\widetilde{c}}$ be the set of d-edges that appear exactly once in $\widetilde{c}$. Let $\widetilde{E}_1$ be the union of $\widetilde{E}_{\widetilde{c}}$ for all standard m.d-cycles $\widetilde{c}$, and let $\widetilde{E}_2$ be the set of all branch d-triangle edges. Then* frame graph *of $G$ is a subgraph $\widetilde{H} = (\widetilde{U}, \widetilde{D})$ of $\widetilde{G}$, where $\widetilde{D} = \widetilde{E}_1 \cup \widetilde{E}_2$ and $\widetilde{U}$ is the set of all d-vertices that are end points of d-edges of $\widetilde{D}$.*

**Remark.** *Later we will show that $\widetilde{E}_{\widetilde{c}}$ is a set of d-cycles for any standard m.d-cycle $\widetilde{c}$. Thus, we will call it a* simplified m.d-cycle *and denote it by $(\widetilde{c})^-$. (Though we use the term "m.d-cycle", $\widetilde{E}_{\widetilde{c}}$ is a simple set of d-edges with no direction nor order.)*

We use below $\widetilde{H}$ to denote the frame graph defined above from $\widetilde{G}$. Note that $\widetilde{H}$ may not be 2-connected, due to a standard m.d-cycle of type (b) in Figure 3.18. ($\widetilde{H}$ may be even disconnected if this standard m.d-cycle has a line part as Figure 3.20.) We handle this situation as a special case. Let us use the standard m.d-cycle of Figure 3.18 (b) for our explanation; let $\widetilde{c}$ denote this m.d-cycle, and let $p$ denote the connector that defines $\widetilde{c}$. The plane is separated into three subplanes by removing $\widetilde{c}$; let $f_0$ denote the subplane inside of $\widetilde{c}$ (i.e., located left of $\widetilde{c}$ following the ordering given by the labeled numers), and let $f_1$ and $f_2$ denote two subplanes outside of $\widetilde{c}$.

We point out first that a standard m.d-cycle of this type occurs at most once in $\widetilde{G}$. As we will show below (Lemma 3.1.14), $\widetilde{c}$ is only one m.d-cycle whose inside intersects with $p$. On the other hand, there is no connector other than $p$ that $f_0$ contains because $B \cup R$ is connected, and both $f_1$ and $f_2$ have edges of $B \cup R \setminus \{p\}$. Thus, there is no




An example of a standard m.d-cycle with a line part. White vertices are d-vertices that are connected by d-edges indicated by dashed lines. A stanard m.d-cycle is defined by $(\widetilde{v}_1, \widetilde{v}_2, \widetilde{v}_3, \widetilde{v}_4, \widetilde{v}_5, \widetilde{v}_6, \widetilde{v}_7, \widetilde{v}_4, \widetilde{v}_3, \widetilde{v}_8, \widetilde{v}_1)$. It has a line part that is removed for defining a frame graph. Then the obtained face, the subplane containing the connector, has two disconnected boundary cycles.

Figure 3.20: An example of an m.d-cycle with a line part

m.d-cycle in $f_0$, which also implies that there is no standard m.d-cycle of this type in $f_1$ or $f_2$. Thus, we check whether $\widetilde{G}$ has a standard m.d-cycle of this type, and if so, we treat it in the following ad hoc way.

Consider the case where $f_0$ has more than $2\widetilde{n}/3$ d-vertices of $\widetilde{G}$. This situation is the same as the case where one of the faces of $\widetilde{H}$ has weight larger than 2/3, we can simply use the algorithm of Lipton and Tarjan as discussed in Section 1. That is, we apply the algorithm of Lipton and Tarjan to the subgraph $\widetilde{G}_{f_0}$ consisting of d-vertices located in $f_0$ or its boundary. Since $f_0$ is a part of one Voronoi region, we can use the BFS d-tree rooted its boss d-vertex to execute the algorithm of Lipton and Tarjan, and since the depth of the BFS d-tree is bounded by $O(k)$, this execution can be done in $\widetilde{O}(k)$-space ($= \widetilde{O}(\sqrt{n})$-space). Consider the case where either $f_1$ or $f_2$ has more than $2\widetilde{n}/3$ d-vertices of $\widetilde{G}$. Let us assume here that $f_1$ has more than $2\widetilde{n}/3$ d-vertices of $\widetilde{G}$. In this case we may ignore the boundary d-cycle of $f_2$ and use only the boundary d-cycle of $f_1$ to define $\widetilde{H}$. The new face introduced in $\widetilde{H}$ with this change, which is essentially the union of $f_0$ and $f_2$, has at most $\widetilde{n}/3$ d-vertices of $\widetilde{G}$, and condition (F3) is still satisfied. Finally, consider the case where none of three subplanes has more than $2\widetilde{n}/3$ d-vertices of $\widetilde{G}$. Note that there should be some subplane that has more than, say, $\widetilde{n}/4$ d-vertices. Then we may use the d-cycle boundary of this subplane as our target separator. Since it is a part of the standard m.d-cycle $\widetilde{c}$, we can bound its size (i.e., the number of d-vertices of the separator) by $O(\sqrt{k})$ from Lemma 3.1.8 (see also the discussion below).

With this special treatment for the exceptional case, we may now assume that $\widetilde{H}$ is 2-connected and satisfies (F1). Thus, our task is to show that it also satisfies conditions (F2) $\sim$ (F4). For this we need to identify faces of $\widetilde{H}$. Intuitively, it is almost clear that d-cycles from standard m.d-cycles of $\widetilde{G}$ are all kept in $\widetilde{H}$ and that every face of $\widetilde{H}$ is defined by either a branch d-triangle or such a d-cycle from a standard m.d-cycle. For the latter case, the subplane that the d-cycle defines as a face is nothing but the subplane that the corresponding standard m.d-cycle defines. Obviously, we still have to prove this intuition, which in fact is not so trivial. But here we first assume that this intuition holds to show that faces of $\widetilde{H}$ satisfy condtions (F2) $\sim$ (F4). Thus, in the following, we simply call a subplane inside of a standard m.d-cycle defines as a *standard face* expecting that it will be shown as a face of $\widetilde{H}$

later. (In general, an m.d-cycle may create more than one inside subplanes like the one in Figure 3.11. We will show later that no standard m.d-cycle creates more than one inside subplanes.)

First note that (F3) and (F4) are clear for faces defined by branch d-triangles. (F3) is also clear for standard faces because every standard face is contained in at most two Voronoi regions; thus, we can choose (any) one of the boss d-vertices of these Voronoi regions as $\widetilde{b}$ and use (at most) two BFS d-trees of these Voronoi regions to define a BFS d-tree rooted at $\widetilde{b}$ that satisfies (F3). (There is a d-path of length $O(k)$ from $\widetilde{b}$ to the boss $\widetilde{b}'$ of the other Voronoi region; we can combine this d-path and the BFS d-tree of the other Voronoi region to span all d-vertices in the other Voronoi region.) Note that each standard m.d-cycle is defined by at most four d-paths of BFS d-tree(s) and two branch d-triangle edges. Also recall Lemma 3.1.8; we may assume that the length of every d-path of BFS d-trees is bounded by $O(\sqrt{k})$. Thus, the length of standard m.d-cycle (and hence, that of the corresponding d-cycle of $\widetilde{H}$) is bounded by $O(\sqrt{k})$, satisfying condition (F4). Thus, it remains to bound the number of branch d-triangles and standard faces.

**Lemma 3.1.13.** *The number of connectors is $O(n/k)$. Also, the number of branch vertices is $O(n/k)$.*

*Proof.* We define a plane graph $G' = (V', E')$ from $R \cup B$. Let $V'$ be the set of branch vertices of $R \cup B$, and let $E'$ be the set of edges between adjacent branch vertices in $R \cup B$. We assume the planar embedding following that of $G$. Since $G'$ is plane, it satisfies Euler's formula $|F'| + |V'| = |E'| + 2$, where $F'$ is the set of faces of $G'$, which in fact is the number of Voronoi regions. , and $|V'|$ is the number of branch vertices. Since degree of any branch vertex is 3, we have $2|E'| = 3|V'|$. By substituting this to the above Euler's formula, we have $3|F'| + 2|E'| = 3|E'| + 6$, implying $|E'| = 3|F'| - 6 = O(|F'|)$. Since $|F'|$ is the number of Voronoi regions, we have $|E'|$, which is the number of connectors, is $O(n/k)$. Also we can bound the number of branch vertices by $O(n/k)$ because $2|E'| = 3|V'|$. □

Since there is a one-to-one correspondence between branch d-triangles and branch vertices, from the above lemma, it immediately follows that the number of faces from branch d-triangles are bounded by $O(n/k)$. Similarly, as we can see from Figure 3.18, each connector uniquely corresponds to a standard face, which intuition is verified below (Lemma 3.1.14 and Lemma 3.1.16). Thus, the number of standard faces is also bounded by $O(n/k)$. Therefore (provided there is no other face in $\widetilde{H}$) the number of faces of $\widetilde{H}$ is $O(n/k)$, that is, condition (F2) holds.

**Lemma 3.1.14.** *For any connector $p$, let $\widetilde{c}$ be an m.d-cycle containing $p$, and let $p'$ be a subpath of $p$ obtained by removing the first and the last edges of $p$. Then there is no edges nor vertices of $B \cup R$ except $p'$ in the inside of m.d-cycle $\widetilde{c}$.*

*Proof.* We use the notation of Figure 3.17 here with $p$ and $\widetilde{c}$ of the lemma. Since BFS d-tree of each Voronoi region does not cross any edge of $R \cup B$, only two d-edges $\widetilde{e}_{\text{fst}}$ and $\widetilde{e}_{\text{last}}$ of $\widetilde{c}$ are crossing edges of $R \cup B$. Therefore, $R \cup B$ is parted into at most three connected components by $\widetilde{c}$, and each component is in either in the left side of

$\widetilde{c}$ (that is, in the standard face) or in the right side of $\widetilde{c}$ (that is, not in the standard face). Clearly, one component of them is $p'$ and it is in the standard face, The others contain branch vertices that are not in the standard face; hence, those components are not contained in the face. □

We now show that faces of $\widetilde{H}$ are exactly the same as the set of branch d-triangles and standard faces of $\widetilde{G}$. Since there are seveal notions of "face", in order to avoid confusion, we will refer real faces of $\widetilde{G}$ as *d-faces* to distinguish from the other types of faces. Recall that $\widetilde{G}$ is triangulated; hence, all d-faces are in fact d-triangles. Furthermore, there is a one-to-one correspondence between d-faces of $\widetilde{G}$ and vertices of $G$; that is, each d-face (i.e., d-triangle) corresponds to a vertex of $G$ that it contains. In the following, let $\widetilde{H}^+$ denote a subgraph of $\widetilde{G}$ that consists of all d-vertices and d-eduges of branch d-triangles and standard m.d-cycles. Recall that $\widetilde{H}$ is obtained from $\widetilde{H}^+$ by removing d-vertices and d-edges not participating d-cycles of standard m.d-cycles or branch d-triangles.

By definition the inside of any m.d-cycle is a set of nonoverlapping subplanes. Here we first show that the inside of every standard m.d-cycle is a single subplane with one or two boundary d-cycles. First we observe that the inside of every standard m.d-cycle is empty w.r.t. $\widetilde{H}^+$.

**Lemma 3.1.15.** *Any standard m.d-cycle has no d-vertex nor d-edge of $\widetilde{H}^+$ in its inside.*

*Proof.* Consider any standard m.d-cycle $\widetilde{c}$. Recall that every d-vertex (resp., every d-edge) of $\widetilde{H}^+$ is a part of either some branch d-path or some branch d-triangle. Thus, it suffices to show that no d-vertex of a branch d-path or a branch d-triangle is in the inside of $\widetilde{c}$. From Lemma 3.1.14 no branch vertex is located in the inside of $\widetilde{c}$; hence, no branch d-vertex, i.e, no d-vertex of a branch d-triangle, belongs to the inside of $\widetilde{c}$ (see, e.g., Figure 3.17 for the relation between branch vertices and d-edges of m.d-cycles).

Thus, it remains to show that no d-vertex of a branch d-path is in the inside of $\widetilde{c}$. Suppose to the contrary that some branch d-path $\widetilde{p}$ belongs in part to the inside of $\widetilde{c}$. Since $\widetilde{p}$ is a branch d-path, one end of it is a branch d-vertex $\widetilde{u}$. Hence, $\widetilde{u}$ is not in the inside of $\widetilde{c}$ (since the inside of $\widetilde{c}$ has no branch d-vertex). Thus, $\widetilde{p}$ must cross $\widetilde{c}$; more specifically, some branch d-path $\widetilde{q}$, which is a part of $\widetilde{c}$, shares some d-vertex with $\widetilde{p}$. Note that $\widetilde{p}$ belongs to the same Voronoi region that $\widetilde{q}$ belongs to; hence, these two branch d-paths share the same boss d-vertex $\widetilde{b}$. Now traverse $\widetilde{p}$ from $\widetilde{v}$ to $\widetilde{b}$, and let $(\widetilde{v}, \widetilde{w})$ be the first d-edge that goes into the inside of $\widetilde{c}$, where $\widetilde{v}$ be the one that is in the inside of $\widetilde{c}$. Then clearly $\widetilde{w}$ is a d-vertex on $\widetilde{q}$. But then we have two d-paths from $\widetilde{w}$ to its boss $\widetilde{b}$, one following $\widetilde{q}$ and another following $\widetilde{p}$. This contradicts to the choice of our BFS d-tree. □

We introduce "d-region", a dual version of region, and show that the inside of any standard m.d-cycle is regarded as a d-region, thereby showing that it defines a single subplane. Recall that a region is a set of "well connected" faces of $G$; a d-region is a set of similarly connected d-faces of $\widetilde{G}$. We say that two d-faces $\widetilde{f}$ and $\widetilde{f}'$ of $\widetilde{G}$ are *d-face-connected* if there exists a sequence $\widetilde{f}_1, \ldots, \widetilde{f}_t$ of d-faces of $\widetilde{G}$ such that $\widetilde{f} = \widetilde{f}_1$,

$\widetilde{f}' = \widetilde{f}_t$, and $\widetilde{f}_i$ and $\widetilde{f}_{i+1}$ share one d-edge for all $i$, $1 \leq i \leq t-1$. (Since all d-faces are d-triangles, any pair of two-d-faces can share at most one d-edge.) Then a set $\Phi$ of d-faces of $\widetilde{G}$ is called a *d-region* if any two faces of $\Phi$ are d-face-connected. Also we define the *d-boundary* of $\Phi$ by the following set of d-edges.

$$\widetilde{B} \;=\; \{\, \widetilde{e} \in \widetilde{E} \mid \text{there exists exactly one face in } \Phi \text{ that contains } \widetilde{e} \,\}$$

Clearly, the properties of regions hold also for d-regions; hence, the d-boundary of a d-region is a set of d-cycles.

We then consider a bit stronger connectivity than the d-face connectivity, thereby introducing a subclass of d-regions.

**Definition 3.1.17** (strongly connected d-region)**.** *Two d-faces $\widetilde{f}$ and $\widetilde{f}'$ of $\widetilde{G}$ are* strongly d-face-connected *if there exists a sequence $\widetilde{f}_1, \ldots, \widetilde{f}_t$ of d-faces of $\widetilde{G}$ such that $\widetilde{f} = \widetilde{f}_1$, $\widetilde{f}' = \widetilde{f}_t$, and $\widetilde{f}_i$ and $f_{i+1}$ share one d-edge of $\widetilde{E} \setminus \mathrm{E}(\widetilde{H}^+)$ for all $i$, $1 \leq i \leq t-1$. A* strongly connected d-region *is a set $\Phi$ of faces of $\widetilde{G}$ such that any two d-faces of $\Phi$ are strongly d-face-connected.*

**Lemma 3.1.16.** *For any standard m.d-cycle $\widetilde{c}$, let $\Phi$ be the set of all d-faces of $\widetilde{G}$ in the inside of $\widetilde{c}$. Then $\Phi$ is a strongly connected d-region.*
**Remark.** *Clearly, any d-region is a single subplane in plane graph $\widetilde{G}$. Thus, the lemma shows that each standard m.d-cycle defines a single subplane of $\widetilde{G}$.*

*Proof.* For the lemma it suffices to show all d-faces of $\Phi$ are strongly d-face-connected because no d-face $\widetilde{f}$ that is not in $\Phi$ (i.e., located in the outside of $\widetilde{c}$) is not strongly d-face-connected because the inside of $\widetilde{c}$ is separated from the outside by removing d-edges of $\widetilde{H}^+$.

Let $p$ a connecter that defines m.d-cycle $\widetilde{c}$, and let $p'$ be the part of $p$ removing the two end points of $p$. Let $\widetilde{f}_1$ and $\widetilde{f}_2$ be two d-faces corresponding to two end points of $p'$ (i.e., two d-triangles containing those two end points); more specifically, let $\widetilde{f}_1$ (resp., $\widetilde{f}_2$) be the one with d-edge $\widetilde{e}_{\mathrm{fst}}$ (resp., $\widetilde{e}_{\mathrm{last}}$) of Figure 3.17. By definition $\widetilde{f}_1$ and $\widetilde{f}_2$ are located the inside of $\widetilde{c}$; in other words, they belong to $\Phi$. We first note that they are strongly d-face-connected. This follows from the fact that d-faces of $\widetilde{G}$ that correspond to vertices of $p'$ are strongly d-face-connected because no edges of $\widetilde{H}^+$ crosses $p'$ (since otherwise, there must be some branch vertex on $p'$). Thus, it suffices to show now that any d-face of $\Phi$ is strongly d-face-connected to either $\widetilde{f}_1$ or $\widetilde{f}_2$.

Consider any d-face $\widetilde{f}$ of $\Phi$, and let $\Phi_{\widetilde{f}}$ be a maximal strongly connected d-region containing $\widetilde{f}$. As discussed above, we have $\Phi_{\widetilde{f}} \subseteq \Phi$ because no d-face outside of $\widetilde{c}$ can be strongly d-face-connected to d-face $\widetilde{f}$ that is in the inside of $\widetilde{c}$. We prove below that $\widetilde{f}_1$ (or $\widetilde{f}_2$) is in $\Phi_{\widetilde{f}}$, thereby showing that $\widetilde{f}$ is strongly d-face-connected to $\widetilde{f}_1$ (resp., $\widetilde{f}_2$).

Let $\widetilde{B}_{\widetilde{f}}$ be the d-boundary of $\Phi_{\widetilde{f}}$. Clearly, $\widetilde{B}_{\widetilde{f}}$ consists of d-edges of $\widetilde{H}^+$; because if not, we can expand strongly connected d-region $\Phi_{\widetilde{f}}$ by adding a d-face that share this boundary d-edge in $\widetilde{E} \setminus \mathrm{E}(\widetilde{H}^+)$, which contradicts to the maximality of $\Phi_{\widetilde{f}}$.

Furthermore, we can show that any d-edge of $\widetilde{B}_{\widetilde{f}}$ belongs to $\widetilde{c}$. Suppose that $\widetilde{B}_{\widetilde{f}}$ has some d-edge $\widetilde{e}$ of $\mathrm{E}(\widetilde{H}^{+}) \setminus \mathrm{E}(\widetilde{c})$. Then two d-triangles facing $\widetilde{e}$ must belongs to $\Phi$ because one of them in $\Phi_{\widetilde{f}}$ is in $\Phi$ and the other is not separated by $\widetilde{c}$ from our assumption. Hence, $\widetilde{e}$ is located in the inside of $\widetilde{c}$, which contradicts to Lemma 3.1.15.

Since $\Phi_{\widetilde{f}}$ is a d-region, $\widetilde{B}_{\widetilde{f}}$ is a set of d-cycles. Consider one of its d-cycles. As shown above, this d-cycle consists of d-edges of $\widetilde{c}$. Recall that $\widetilde{c}$ consists of branch d-paths and two d-triangle edges $\widetilde{e}_{\mathrm{fst}}$ and $\widetilde{e}_{\mathrm{last}}$; hence, in order to from a d-cycle, it must contain one of these two d-triangle edges because a branch d-path consists of d-edges of a BFS d-tree (or two disjoint BFS d-trees). Therefore, $\widetilde{B}_{\widetilde{f}}$ contains at least $\widetilde{e}_{\mathrm{fst}}$ or $\widetilde{e}_{\mathrm{last}}$, which clearly implies $\widetilde{f}_1 \in \Phi_{\widetilde{f}}$ or $\widetilde{f}_2 \in \Phi_{\widetilde{f}}$, as desired. (Though it is not necessary for our proof, we can also show that both $\widetilde{f}_1 \in \Phi_{\widetilde{f}}$ and $\widetilde{f}_2 \in \Phi_{\widetilde{f}}$ are in $\Phi_{\widetilde{f}}$.) □

As a corollary to the above proof, any standard m.d-cycle $\widetilde{c}$ defines one d-region $\Phi$ as its inside, whose d-boundary $\widetilde{B}$ consists of d-edges of $\widetilde{c}$. Next we prove that $\widetilde{B}$ is exactly the same as the simplified m.d-cycle $(\widetilde{c})^{-}$, which also implies that $(\widetilde{c})^{-}$ is a set of d-cycles.

**Lemma 3.1.17.** *For any m.d-cycle $\widetilde{c}$. let $\Phi$ be the set of d-faces that are in the inside of $\widetilde{c}$, and let $\widetilde{B}$ be its d-boundary. Then we have $(\widetilde{c})^{-} = \widetilde{B}$.*

*Proof.* As explained above, it is enough to show (i) $\widetilde{B}$ contains all d-edges of $(\widetilde{c})^{-}$, and (ii) $\widetilde{B}$ contains no other d-edges of $\widetilde{c}$. In fact, both are easy from Lemma 3.1.9. To prove (i), consider any d-edge $\widetilde{e}$ of $(\widetilde{c})^{-}$, i.e, a d-edge of $\widetilde{c}$ that appears once in $\widetilde{c}$. Then by definition (and Lemma 3.1.9), one of its two adjacent d-triangles belongs to $\Phi$ and the other does not. Thus, $\widetilde{e}$ must be a d-boundary edge, i.e., an element of $\widetilde{B}$. To prove (ii), consider any duplicate d-edge of $\widetilde{c}$. Then from Lemma 3.1.9, its two adjacent d-triangles are either both in $\Phi$ or both not in $\Phi$. Hence, $\widetilde{e}$ cannot be a d-boundary edge. □

From this lemma it follows that standard faces are faces of $\widetilde{H}$. We can now conclude our discussion by showing that $\widetilde{H}$ has no other face than these standard faces and branch d-triangles.

**Lemma 3.1.18.** *Any face of frame graph $\widetilde{H}$ is either a standard d-face or a branch d-triangle.*

*Proof.* Consider any face of $\widetilde{H}$. Since $\widetilde{H}$ is 2-connected, the face has a single boundary d-cycle $\widetilde{c}_0$ (Fact 1). The lemma clearly holds if $\widetilde{c}_0$ is a branch d-trianble; thus, we assume that it is not a branch d-triangle. We first show that it has at least one d-edge from some branch d-triangle. This is because $\widetilde{H}$ consists of d-edges of BFS d-trees and branch d-triangles and we need at least one d-edge of some branch d-triangle since no d-cycle is created by d-edges of disjoint d-trees.

Let $\widetilde{e}_0$ and $\widetilde{f}_0$ be respectively such a d-edge and the branch d-triangle with this d-edge. Note that $\widetilde{c}_0$ and $\widetilde{f}_0$ are adjacent faces sharing d-edge $\widetilde{e}_0$. Here we overlap the original plane graph $G$ with $\widetilde{H}$. Then by definition there exists a branch vertex

in branch d-triangle $\widetilde{f}_0$ where three connector of $R \cup B$ are merged. Clearly, there should be one connector that starts with an edge crossing d-edge $\widetilde{e}_0$. Thus, $\widetilde{c}$ contains this connector. On the other hand, there must be an m.d-cycle $\widetilde{c}$ that corresponds to this connector (Definition 3.1.10 and Lemma 3.1.14). Note here that the standard face defined by $\widetilde{c}$ has no other d-edges of $\widetilde{H}^+$ (Lemma 3.1.15) in its inside. Therefore $\widetilde{c}_0$ must be identical to its simplified version $(\widetilde{c})^-$, and the face of $\widetilde{H}$ defined by $\widetilde{c}_0$ the standard face defined by $\widetilde{c}$. □

We summarize our discussion with the following lemma that follows immediate from the above lemmas.

**Theorem 3.1.19.** *Frame graph $\widetilde{H}$ satisfies conditions (F1) ∼ (F4).*

### 3.1.5 Algorithms

We explain our separator algorithm. Following the outline explained in Section **??**, the algorithm modifies a given undirected graph $G = (V, E)$ to get a frame graph and then computes its separator by using either the algorithm of Lipton and Tarjan or that of Gazit and Miller. The desired separator of $G$ is computed by a simple post-processing sub-algorithm. Here we follow arguments of previous sections and show sequence of sub-algorithms that compute modified graphs and/or additional information step by step to obtain the final separator.

In the following we state the whole procedure by seven steps and describe a sub-algorithm computing each step. We assume that an output of each sub-algorithm is added to the previous output including the original input graph $G$ that is regarded as an input to the next sub-algorithm. As remarked in Introduction, we cannot in general keep these modified graphs or additional information; but it is easy to modify these sub-algorithms so that necessary information is recomputed on the fly. (Here and throughout this section we will use notions and notation defined in the previous sections.)

1. Pre-process.
   **Output:** a triangulated and vertex expanded graph $G^{\mathrm{ve}}$ (that is simply denoted by $G$ in the following steps), its dual graph $\widetilde{G} = (\widetilde{E}, \widetilde{V})$, and combinatorial embeddings of these two graphs.

2. Computing Voronoi regions:
   **Output:** a $k$-maximal independent set $\widetilde{I}$ (Definition 3.1.5), and a list of $(\widetilde{v}, \mathrm{boss}(\widetilde{v}), d_{\mathrm{neighbor}}, d_{\mathrm{boss}})$ for all $\widetilde{v} \in \widetilde{V}$, where $\mathrm{boss}(\widetilde{v})$ is the "center" d-vertex of the Voronoi region that $\widetilde{v}$ belongs to, or more precisely, $\widetilde{b} \in \widetilde{I}$ whose $k$-neighborhood is closes to $\widetilde{v}$ (Definition 3.1.6), and $d_{\mathrm{neighbor}}$ and $d_{\mathrm{boss}}$ are the distances from $\widetilde{v}$ to respectively this $k$-neighborhood and $\widetilde{b}$.

3. Computing floor and ceiling d-cycles:
   **Output:** Modified $G$ and $\widetilde{G}$ by "contracting" the inside of floor and ceiling d-cycles, and a revised set $\widetilde{I}$ of boss d-vertices. (They are denoted by $G$, $\widetilde{G}$, $\widetilde{I}$

respectively in the following steps.) A BFS d-tree of the Voronoi region $\mathrm{V}(\widetilde{b})$ of each $\widetilde{b} \in \widetilde{I}$.

4. Computing $B \cup R$.
   **Output:** A set of branch vertices and branch d-triangles, a set $B$ of Voronoi boundary edges, and a set $R$ of ridge edges.

5. Computing a frame graph.
   **Output:** a frame graph $\widetilde{H}$ and its complete face information (Definition 2.1.5) together with the list of face weights.

6. Computing a separator of $\widetilde{H}$. **Output:** a separator of $\widetilde{H}$.

7. Post-process. **Output:** a separator of $G$.

We now show an $\widetilde{O}(\sqrt{n})$ or less space and polynomial-time algorithm for each step. (Since the polynomial-time computability is almost clear in the following discussion, we omit claiming "polynomial-time" for these algorithms.) Here we will use at several places the $O(\log n)$-space algorithm of Reingold for the undirected reachability, which is denoted by `algo_reach`.

Step 1. Pre-process

We explain the computation for this step by several sub-steps. We may assume again that the output of each sub-step as an input to the next sub-step. First compute a planar embedding of $G$. Allender and Mahajan [1] reduced the problem of computing a planar embedding to the undirected reachability. Hence, by using `algo_reach` we can compute a planar embedding of $G$ by using $O(\log n)$-space. Then based on this planar embedding, it is easy to construct *some* triangulation w.r.t. this embedding in $O(\log n)$-space. In the next sub-step, we modify this triangulated graph to $G^{\mathrm{ve}}$ by the vertex expansion explained in Section **??**, which can be done in $O(\log n)$-space. Then from the obtained $G^{\mathrm{ve}}$, its dual graph $\widetilde{G}$ and its embedding are computed in $O(\log n)$-space. We may assume that the correspondence between an edge $e$ of $G$ and its crossing d-edge $\widetilde{e}$ of $\widetilde{G}$ is also computed; more specifically, the list of pairs of $e$ and $\widetilde{e}$ is added to the output.

Step 2. Computing Voronoi regions

A tool we need here is a standard BFS search algorithm traversing at most $t$ d-vertices from a given source d-vertex $v$ of $\widetilde{G}$. It is easy to see that the standard one needs $\widetilde{O}(t)$-space. Then we have an $\widetilde{O}(k)$-space algorithm that enumerates, for a given $\widetilde{v}$, all elements of $\mathrm{N}_k(\widetilde{v})$, the set of the first $k$ closest d-vertices to $\widetilde{v}$ in $\widetilde{G}$ (Definition **??**). Here we need to be a bit careful about $\mathrm{N}_k^+(\widetilde{v})$ that contains additionally all d-vertices whose distance is the same as the one in $\mathrm{N}_k(\widetilde{v})$ with the largest distance from $\widetilde{v}$. Since $\mathrm{N}_k^+(\widetilde{v})$ can become much larger than $\mathrm{N}_k(\widetilde{v})$, we cannot keep all elements of $\mathrm{N}_k^+(\widetilde{v})$. But once we have $\mathrm{N}_k(\widetilde{v})$, we can check easily whether $\widetilde{u} \in \mathrm{N}_k^+(\widetilde{v})$ for a given $\widetilde{u}$ by checking whether it is either in $\mathrm{N}_k(\widetilde{v})$ or adjacent to some d-vertex of $\mathrm{N}_k(\widetilde{v})$. Thus, we have an $\widetilde{O}(k)$-space algorithm that checks, for given $\widetilde{u}$ and $\widetilde{v}$, whether $\mathrm{N}_k^+(\widetilde{u}) \cap \mathrm{N}_k^+(\widetilde{v}) \neq \emptyset$ by computing $\mathrm{N}_k(\widetilde{u})$ and $\mathrm{N}_k(\widetilde{v})$ and checking the membership to both $\mathrm{N}_k^+(\widetilde{u})$ and $\mathrm{N}_k^+(\widetilde{v})$ for all d-vertices of $\widetilde{G}$. Furthermore, if $\mathrm{N}_k^+(\widetilde{u}) \cap \mathrm{N}_k^+(\widetilde{v}) \neq \emptyset$, then the algorithm can

also identify the d-vertex of $\mathrm{N}_k(\widetilde{v})$ that is closest to $\widetilde{u}$, and hence, it can compute the distances from $\widetilde{u}$ to $\mathrm{N}_k(\widetilde{v})$ and to $\widetilde{v}$ respectively.

By using these tools we can design an $\widetilde{O}(k+n/k)$-space algorithm that collects d-vertices to $\widetilde{I}$ in a greedy way until there is no d-vertex $\widetilde{v}$ such that $\mathrm{N}_k^+(\widetilde{v}) \cap \mathrm{N}_k^+(\widetilde{b}) = \emptyset$ for all $\widetilde{b} \in \widetilde{I}$. Note that the working memory for keeping $\widetilde{I}$ is needed here (though the purpose of the algorithm is to produce $\widetilde{I}$ to the output tape) to achieve the greedy algorithm. Hence, we need $\widetilde{O}(k+n/k)$-space since we can bound $|\widetilde{I}| \leq n/k$.

Once $\widetilde{I}$ is obtained, the additioal information, that is, the list $\{(\widetilde{v}, \mathrm{boss}(\widetilde{v}), d_{\mathrm{neighbor}}, d_{\mathrm{boss}})\}_{\widetilde{v}\in\widetilde{V}}$ is not difficult to compute again by using the above tools. In fact, when we have $\widetilde{I}$, each $(\widetilde{v}, \mathrm{boss}(\widetilde{v}), d_{\mathrm{neighbor}}, d_{\mathrm{boss}})$ is (and hence the whole list) is $\widetilde{O}(k)$-space computable.

Step 3. Computing floor and ceiling d-cycles

Tools we need here are `algo_bcycle`, an algorithm computing all boundary cycles of a given region, and `algo_which`, an algorithm that determines, given a cycle $c$ and a vertex $v$, which subplane separated by $c$ does vertex $v$ belongs to.

Algorithm `algo_bcycle` is used in a situation where we can determine whether a give vertex is in a target region. For example, for a given d-vertex $\widetilde{b} \in \widetilde{I}$, we can determine (by using the input given to this step) whether a given vertex $v$ is in the Voronoi region $\mathrm{Vr}(\widetilde{b})$. Then we have the following simple algorithm that enumerate all boundary cycles:

**for** each vertex $v$ of $G$ in the order of their indices **do**
  **if** the following (i) and (ii) hold:
  (i) $v$ is a boundary vertex (i.e., it is in the region and
    one of its three adjacent vertices is not in the region), and
  (ii) $v$ is not connected to any boundary vertex with a smaller index
  **then**
    output all vertices of the boundary cycle from $v$ (should be a cycle by Fact **??**)
    by traversing adjacent boundary vertices from $v$
**end-for**

**Algorithm `algo_bcycle` for producing all boundary cycles**

It is easy to see that the algorithm needs $O(\log n)$-space besides the space needed for recognizing the target region.

Algorithm `algo_which` is used in a situation where a sequence of vertices of a given cycle $c$ in the order of visit clockwise/anticlockwise is given and we want to determine which side (i.e., the left or right w.r.t. the sequence and the embedding also given as an input) does a given vertex $v$ belong to. This task can be achieved by checking whether $v$ is connected to a vertex that is adjacent to the left of any fixed vertex of $c$ with a path having no vertex in $c$, and this can be done in $O(\log n)$-space by using `algo_reach`. (Recall that we assume that $G$ is connected.) Clearly the same idea worsk also for dual graph $\widetilde{G}$. By using this tool, we can also count the number of vertices located in each subplane separted by a given cycle.

Armed with these tools, we now explain an algorithm achieving the required task for this step. The computation of the algorithm is divided into five sub-steps. The first

sub-step is to produce all cores and necks. Then from these cores and necks, all their cycle boundaries are produced in the second sub-step. From these boundary cycles, we select in the third sub-step those used for floor and ceiling cycles. Then in the fourth sub-step, new graph $G$ and dual graph $\widetilde{G}$ are computed by "contracting" the inside of these floor and ceiling d-cycles. The set $\widetilde{I}$ and the list $\{(\widetilde{v}, \mathrm{boss}(\widetilde{v}), d_{\mathrm{neighbor}}, d_{\mathrm{boss}})\}_{\widetilde{v}\in\widetilde{V}}$ are also revised accordingly. Finally, the description of a BFS d-tree of each Voronoi region is computed from the revised list.

The first sub-step, i.e., enumerating d-vertices of all cores and necks, is easy. Recall that for each $\widetilde{b} \in \widetilde{I}$, its core $\mathrm{Core}(\widetilde{b})$ is simply some subset $\cup_{1\le i\le d}\mathrm{L}(\widetilde{b}, i)$ of $\mathrm{N}_k(\widetilde{b})$ (Definition **??**); hence, a simple algorithm can produce all its elements by going through the list $\{(\widetilde{v}, \mathrm{boss}(\widetilde{v}), d_{\mathrm{neighbor}}, d_{\mathrm{boss}})\}_{\widetilde{v}\in\widetilde{V}}$ passed from the previous step. Recall a neck is a connected component of $\mathrm{L}_{\mathrm{nb}}(\ell)$ of size $\le \sqrt{k}$ (see the paragraph below Lemma 3.1.6 for the definition of "neck"), where sliced graph $\mathrm{L}_{\mathrm{nb}}(\ell)$ is the set of d-vertices whose distance from the nearest $k$-neighborhood is $\ell$. Thus it easy to recognize $\mathrm{d}_{\mathrm{nb}}(\ell)$ and its connected components by using the list $\{(\widetilde{v}, \mathrm{boss}(\widetilde{v}), d_{\mathrm{neighbor}}, d_{\mathrm{boss}})\}_{\widetilde{v}\in\widetilde{V}}$.

Recall that cores and necks are regions; hence, the second sub-step is simply to apply `algo_bcycle` to the obtained cores and necks to produce boundary cycles as candidates of floor and ceiling cycles. More specifically, we check, for each cycle boundary $c$ of each core, whether its exterior, the subplane defined by $c$ that does not contain the core, has more than $2\widetilde{n}/3$ d-vertices, and if so, produce $c$ as a candidate of a floor cycle. A boundary cycle $c$ of a neck is processed similarly. We may assume that `algo_bcycle` computing a boundary cycle $c$ can also tell us whether it consists of only interior edges or only exterior edges (see the proof of Lemma 3.1.6); thus, depending on $c$'s type, we check whether its interior or exterior has more than $2\widetilde{n}/3$ d-vertices, and if so, produce $c$ as a candidate of a floor/ceiling cycle.

In the third sub-step, for each floor/ceiling cycle candidate, we check whether it is contained in the co-exterior of any other floor cycle candidate or in the co-interior of any other ceiling cycle candidate, and if not, produce its associated d-cycle (Definition 3.1.9) as a floor/ceiling d-cycle.

In the forth sub-step we simply follow the explanation given at the beginning of subsection **??** and modify $\widetilde{G}$ by "contracting" the inside of each floor/ceiling d-cycle to one d-vertex. Since this modification can be done locally, it is easy to see that $O(\log n)$-space is sufficient for computing this modification. Note that we may have to revise $\widetilde{I}$ and the list $\{(\widetilde{v}, \mathrm{boss}(\widetilde{v}), d_{\mathrm{neighbor}}, d_{\mathrm{boss}})\}_{\widetilde{v}\in\widetilde{V}}$.

In the last sub-step we compute the description of a BFS d-tree of each $\mathrm{Vr}(\widetilde{b})$. Consider any $\widetilde{b} \in \widetilde{I}$. Recall that Voronoi region $\mathrm{Vr}(\widetilde{b})$ is defined by the set of d-vertices $\widetilde{v}$ such that $\mathrm{boss}(\widetilde{v}) = \widetilde{b}$; its BFS d-tree is a spanning tree of such d-vertices rooted at $\widetilde{b}$ so that the depth of each d-vertex $\widetilde{u}$ in the tree is equal to the distance between $\widetilde{b}$ and $\widetilde{u}$. For each $\widetilde{u} \neq \widetilde{b}$ that belongs to $\mathrm{Vr}(\widetilde{b})$, we can determine its parent in our BFS d-tree as a d-vertex with the smallest index among all d-vertices adjacent to $\widetilde{u}$ that have one smaller distance from $\widetilde{b}$ that $\widetilde{u}$. We describe such a BFS d-tree by the list of $(\widetilde{v}; \widetilde{u}_1, \ldots, \widetilde{u}_t)$ for all $\widetilde{v}$ of $\mathrm{Vr}(\widetilde{b})$, where $\widetilde{u}_i$ is the $i$th child of $\widetilde{v}$ in the BFS d-tree. Clearly, some $O(\log n)$-space algorithm produces such description of a BFS d-tree for all Voronoi regions by using the list $\{(\widetilde{v}, \mathrm{boss}(\widetilde{v}), d_{\mathrm{neighbor}}, d_{\mathrm{boss}})\}_{\widetilde{v}\in\widetilde{V}}$.

Step 4. Computing $B \cup R$

A tool we need here is `algo_climb`, an algorithm that computes a d-path on a BFS d-tree of a Voronoi region specified its boss d-vertex $\widetilde{b}$; more specifically, the algorithm is given $\widetilde{b}$ and $\widetilde{v}$, and it computes a d-path from $\widetilde{v}$ to $\widetilde{b}$ in the BFS d-tree of $\mathrm{Vr}(\widetilde{b})$. This can be achieved by searching the parent of the current d-vertex (initially $\widetilde{v}$)] until the algorithm reaches the boss. The search for the parent of a given d-vertex is $O(\log n)$-space computable by using the information of BFS d-trees from the previous step. Due to Lemma 3.1.8, any d-path in any BFS-tree has at most $4\sqrt{k}+2$ d-vertices; hence, we may keep all d-vertices of the obtained d-path by using $\widetilde{O}(\sqrt{k})$-space.

With this tool, we now explain an algorithm for achieving step 4. The computation of the algorithm is divided into two sub-steps. The first sub-step is to produce the set $B$ of all Voronoi boundary edges. This can be done by applying `algo_bcycle` to a Voronoi region for each $\widetilde{b} \in \widetilde{I}$. Then in the second sub-step we use this information to produce all ridge edges. Below we explain how to determine a given edge $e$ is a ridge edge. We may assume that $e$ is not a boundary edge in $B$, and that it is in some Voronoi region $\mathrm{Vr}(\widetilde{b})$.

Recall that $e$ is a ridge edge if the d-edge $\widetilde{e} := \{\widetilde{u}, \widetilde{v}\}$ crossing $e$ satisfies the following (Definiton3.1.12):

- $\widetilde{e}$ is not a d-edge on the BFS d-tree of $\mathrm{Vr}(\widetilde{b})$.
- Let $\widetilde{c}(\widetilde{e})$ be the d-cycle induced by $\widetilde{e}$ and two d-paths of the BFS d-tree of $\mathrm{Vr}(\widetilde{b})$ containing $\widetilde{u}$ and $\widetilde{v}$ respectively. Then $\widetilde{c}(\widetilde{e})$ saparates $\mathrm{Vr}(\widetilde{b})$ into two separate subplanes, each of which contains at least one boundary cycle of $\mathrm{Vr}(\widetilde{b})$.

Thus, we decide whether $e$ is a ridge edge as follows. First identify the crossing d-edge $\widetilde{e} = \{\widetilde{u}, \widetilde{v}\}$ of $e$ (which information is given as an input to this step, see Step 1). Since $e$ is not a boundary edge, we may assume that $\widetilde{b} = \mathrm{boss}(\widetilde{u}) = \mathrm{boss}(\widetilde{v})$; thus, we use `algo_climb` to compute two d-paths from $\widetilde{u}$ and $\widetilde{v}$ to $\widetilde{b}$ in the BFS d-tree of $\mathrm{Vr}(\widetilde{b})$. Clearly, $e$ is not a ridge edge if $\widetilde{e}$ appears in these d-paths. Otherwise, we can create a d-cycle $\widetilde{c}(\widetilde{e})$ by using d-edges of these d-paths and $\widetilde{e}$, for which we need $\widetilde{O}(\sqrt{k})$-space as explained above. Then by using `algo_which`, we search for two boundary edges of $\mathrm{Vr}(\widetilde{b})$ in the both subplanes separated by $\widetilde{c}(\widetilde{e})$, and we conclude that $e$ is a ridge edge if such two boundary edges exist.

Step 5. Computing a frame graph

We explain this step by six sub-steps. The first sub-step is to enumerate all branch vertices, which can be done easily by checking every vertex of $G$ whether it is incident to three edges of $B \cup R$. Then from these branch vertices, we enumerate, in the second sub-step, connectors, i.e., paths connecting two adjacent branch vertices. The third sub-step computes all standard m.d-cycles, and then in the fourth sub-step, we compute (essentially) our target frame graph $\widetilde{H}$ by ignoring all duplicate directed d-edges in each standard m.d-cycle. Note that for the obtained $\widetilde{H}$, we also need to compute its complete face information, i.e., the list of face boundary representations and the list of all edge-connected and incident pairs of faces. Also we need to produce the list of the weights of its all faces. These lists are computed in the last sub-step.

We begin our algorithmic explanation from the second sub-step (because the first has been explained above). The task of the second sub-step is to enumerate all connectors. Essentially, the task is to enumerate all pairs $(u, v)$ of "adjacent" branch vertices $u$ and $v$ where $u$ has a smaller index than $v$; but for later explanation, we explain the way to produce the list of pairs of the "first" and the "last" edges of connectors (Figure 3.17). Recall that by "adjacent" we mean that $u$ and $v$ are connected by a path consisting of edges of $R \cup B$ with no branch vertex. For a given branch vertex $u$ (and more specifically, for an edge $\{u, u'\}$ of $B \cup R$ incident to it) we can find its adjacent branch vertex by traversing edges of $B \cup R$ until the adjacent branch vertex is reached, which can be done in $O(\log n)$-space. When an adjacent vertex $v$ is found and if $v$ has a larger index than $u$, then we output a pair $(\{u, u'\}, \{v, v'\})$ of the first and the last edge of the connector. (If $v$ has a smaller index, then the corresponding connector information $(\{v, v'\}, \{u, u'\})$ should be produced when branch vertex $v$ is examined.)

In the third sub-step, we need to compute all standard m.d-cycles. Recall that each standard m.d-cycle uniquely corresponds to some connector. Thus, for our task, it suffices to show a way to compute the standard m.d-cycle for each connector. Consider any connector that is specified by its first edge and last edge produced by the previous step, and let $\widetilde{e}_{\mathrm{fst}} = \{\widetilde{v}_{\mathrm{fst.r}}, \widetilde{v}_{\mathrm{fst.l}}\}$ and $\widetilde{e}_{\mathrm{last}} = \{\widetilde{v}_{\mathrm{last.r}}, \widetilde{v}_{\mathrm{last.l}}\}$ be d-edges that cross these first and last edges respectively (Figure 3.17). For each d-vertex of $\{\widetilde{v}_{\mathrm{fst.r}}, \widetilde{v}_{\mathrm{fst.l}}, \widetilde{v}_{\mathrm{last.r}}, \widetilde{v}_{\mathrm{last.l}}\}$, we can compute a d-path to its boss d-vertex by using `algo_climb`. Then we simply need to output their edges in an appropriate order given by Definition 3.1.15, which can be done in $O(\log n)$-space.

In the fourth sub-step we compute the set of d-edges of a frame graph $\widetilde{H}$. Recall that $\widetilde{H}$ consists of nonduplicate d-edges of standard m.d-cycles and branch d-triangle edges (Definition 3.1.16). For any standard m.d-cycle $\widetilde{c}$, let $\widetilde{E}_{\widetilde{c}}$ denote the set of d-edges appearing once in $\widetilde{c}$. This can be computed by picking up nonduplicate d-edges of $\widetilde{c}$. On the other hand, each branch d-triangle is a d-triangle around some branch vertex $v$ and it is easy to compute such branch d-triangle edges from $v$. Thus, we can compute the list of sets $\widetilde{E}_{\widetilde{c}}$ for all standard m.d-cycles $\widetilde{c}$ and the list of branch d-triangles both in $O(\log n)$-space.

At this point, we check whether $\widetilde{G}$ had a standard m.d-cycle of type (b) of Figure 3.18, which case should be treated exceptionally as explained right after Definition 3.1.16. This test can be done by checking whether there is some $\widetilde{E}_{\widetilde{c}}$ that has two or more d-cycles. If such $\widetilde{E}_{\widetilde{c}}$ exists, then either we can compute a desired separator as in Step 6 below, or we modify $\widetilde{E}_{\widetilde{c}}$ by removing d-edges so that it has only one d-cycle. Otherwise, we may assume that each $\widetilde{E}_{\widetilde{c}}$ can be regarded as a simple d-cycle.

Now in the fifth sub-step, we compute the list of all faces of $\widetilde{H}$ with their weights. Recall that every face of $\widetilde{H}$ is defined either by a standard m.d-cycle $\widetilde{c}$ (or more precisely, by $\widetilde{E}_{\widetilde{c}}$) or by a branch d-triangle. Consider any standard m.d-cycle $\widetilde{c}$. Since we may assume that $\widetilde{E}_{\widetilde{c}}$ is a simple d-cycle, we can rearrange d-vertices of $\widetilde{E}_{\widetilde{c}}$ so that it represents this d-cycle; furthermore, by using `algo_which` we can determine the order so that its left side is the face defined by $\widetilde{E}_{\widetilde{c}}$. Thus, we have an $O(\log n)$-space algorithm that produces the boundary representation of the face defined by $\widetilde{c}$. Also by using `algo_which`, we can count the number of d-vertices of $\widetilde{G}$ removed from this

face for defining $\widetilde{H}$, which is the weight of this face. Applying this process to all standard m.d-cycles, we can enumerate all boundary face representations with their weights.

Finally in the last sub-step, we check, for each pair of faces, whether they are incident and/or edge-connected, from which we can create the list of all edge-connected and incident pairs of faces.

Step 6. Computing a separator of $\widetilde{H}$

Now we have the frame graph $\widetilde{H}$ with its complete face information, and our task here is to compute a separator of $\widetilde{H}$. For obtaining our target separator, we apply either the separator algorithm of Lipton and Tarjan (Theorem **??**) or that of Miller (Corollary 3.1.3) depending on the max. weight of the faces of $\widetilde{H}$.

Consider first the case where there is no face with weight $> 2/3$. In this case we use Miller's algorithm to compute a cycle 1/3-separator of $\widetilde{H}$. From conditions (F4) and Corollary 3.1.3, it follows that the size of this separator is $O(\sqrt{n})$ as explained in Section 1. and that Miller's algorithm can be executed in $O(\sqrt{n})$ as explained in Section 1. From Fact 4, it is easy to see that the obtained separator is the desired 2/9-separator of $\widetilde{G}$ for sufficiently large $n$.

Consider next the case where there is some face $f$ with weight $> 2/3$. In this case we consider the induced subgraph $\widetilde{G}_f$ of $\widetilde{G}$ consisting of d-vertices of $\widetilde{G}$ located either on the boundary or inside of $f$. We apply the algorithm of Lipton and Tarjan to $\widetilde{G}_f$. Clearly, the obtained 1/3-separator is a separator of $\widetilde{G}$. Since $f$ has weight $> 2/3$, there must be at least $2\widetilde{n}/3$ d-vertices in $\widetilde{G}_f$; hence, each separated part of $\widetilde{G}$ in $\widetilde{G}_f$ has at least $2\widetilde{n}/9$ d-vertices. Thus, the obtained separator is a 2/9-separator of $\widetilde{G}$, and its size is at most $d = O(n^{1/4})$. Recall that the work space needed for the algorithm of Lipton and Tarjan is essentially for conducting a breadth-first-search and that the amount of required work space is linear to the depth of its BFS d-tree; hence, from condition (F3) it follows that the algorithm of Lipton and Tarjan can be executed in $\widetilde{O}(k)$-space ($= \widetilde{O}(\sqrt{n})$-space).

Step 7. Post-process

Here we consider the function $\mathrm{inv}(\cdot)$ introduced in the proof of Lemma 3.1.1. It is easy to see that this function itself is $O(\log n)$-space and polynomial-time computable. The important point here is that if $\widetilde{S}$ is a separator of $\widetilde{G}$, then so is $\mathrm{inv}(\widetilde{S})$; it also satisfies the size bounds as shown in Lemma 3.1.1. Thus, we can use $\mathrm{inv}(\widetilde{S})$ as the desired separator for the theorem.

## 3.2 Reachability algorithm given a separator

In this section we present our reachability algorithm that uses the space efficient separator algorithm (called `Separator`) given in Theorem **??**. First we define an algorithm that uses `Separator` iteratively to compute a separator family that splits a given graph into sublinear size components.

**Definition 3.2.1.** *For any undirected graph $G$ with $n$ vertices, an $r(n)$-separator*

family *of $G$ is a set $\overline{S}$ of vertices of $G$ so that the removal of $\overline{S}$ disconnects $G$ into subgraphs each of which contains at most $r(n)$ vertices.*

Although we use the term "separator family", a separator family is just a set of vertices, and hence, the *size* of a separator family is simply the number of vertices in the set.

By using `Separator`, for any $\epsilon > 0$, we can design an algorithm that produces an $n^{1-\epsilon}$-separator family of $O(n^{1/2+\epsilon/2})$ size in polynomial time and $\widetilde{O}(n^{1/2+\epsilon/2})$ space.

**Lemma 3.2.1.** *For any $\epsilon > 0$, there is an algorithm* `SepatorFamily` *that takes a planar graph as input and outputs an $n^{1-\epsilon}$-separator family of size $O(n^{1/2+\epsilon/2})$ in polynomial time and $\widetilde{O}(n^{1/2+\epsilon/2})$ space.*

*Proof.* Our key tool is the separator algorithm `Separator` of Theorem **??**. For a given undirected planar graph of size $n$, this algorithm produces a (30/31)-separator of size at most $c\sqrt{n}$ (for some constant $c > 0$); that is, removal of vertices in the separator disconnects the graph into two subgraphs of size $\leq 30n/31$. Intuitively, we repeat this procedure for some sufficient number of times so that each connected component contains at most $n^{1-\epsilon}$ vertices. The union of separators increases monotonically, and, as we will see, its size is bounded by $O(n^{1/2+\epsilon/2})$. This whole procedure can be implemented by using $\widetilde{O}(n^{1/2+\epsilon/2})$ space for keeping the obtained separators.

Let us examine this idea in more detail. Fix any input undirected graph $G = (V, E)$ with $n$ vertices. We first define an algorithm $\texttt{Separator}^+$ so that it can be used iteratively. This algorithm takes $G$, a parameter $i$ indicating the number of iterations, and a separator family $\overline{S}_i$ obtained so far, and it outputs a new separator family $\overline{S}_{i+1} := \overline{S}_i \cup \overline{S}'$. Here we may assume (by induction hypothesis) that each connected component of $G(V \setminus \overline{S}_i)$ is of size $\leq n_i := (30/31)^i n$. The set $\overline{S}'$ added by the algorithm is the union of separators $S'_1, ..., S'_\ell$ of size $c\sqrt{n_i}$. Each $S'_j$ is obtained by applying `Separator` to a connected component of $G(V \setminus \overline{S}_i)$ of size $> (30/31)n_i$. This guarantees the induction hypothesis on the size of connected components. More specifically, for every vertex $v \in V \setminus \overline{S}_i$, the algorithm first checks whether $v$ is the vertex with the smallest index in the component $G_v$ connected to $v$ in $G(V \setminus \overline{S}_i)$, and if so and if the size of $G_v$ is larger than $(30/31)^{i+1}n$, then `Separator` is applied to this component to produce a separator $S'_j$ for $G_v$.

We apply this algorithm $\texttt{Separator}^+$ for $k$ times, where $k$ is defined by

$$k \;=\; \left\lceil \frac{\epsilon}{\log(31/30)} \log n \right\rceil$$

so that

$$\frac{1}{2} n^{1-\epsilon} \;\leq\; n \left(\frac{30}{31}\right)^k \;\leq\; n^{1-\epsilon}$$

holds. Thus, after the $k$ applications of $\texttt{Separator}^+$, the size of connected components is at most $n^{1-\epsilon}$ as desired.

Next we bound the size of the obtained separator family. Note that there are at most $(31/30)^{i+1}$ components of size $\geq (30/31)^{i+1}n$; hence, there are at most

$(31/30)^{i+1}$ separators added to $\overline{S}_i$ at the $i$th iteration, and each such separator size is at most $c\sqrt{n_i} = c\sqrt{(30/31)^i n}$. Thus, the number of vertices in the final separator family $\overline{S}_{k+1}$ is bounded by

$$\begin{aligned}&\left(\frac{31}{30}\right)c\sqrt{n} + \left(\frac{31}{30}\right)^2 c\sqrt{\left(\frac{30}{31}\right)n} + \cdots + \left(\frac{31}{30}\right)^{k+1} c\sqrt{\left(\frac{30}{31}\right)^k n}\\ =\ & \frac{31c\sqrt{n}}{30}\sum_{i=0}^{k}\left(\frac{31}{30}\right)^{k/2} \le c'\sqrt{n}\left(\frac{31}{30}\right)^{k/2} \le 2c' n^{1/2+\epsilon/2}\end{aligned}$$

for some constant $c' > 0$. Finally, observe that the space used is dominated by the the space required to store the separator family. It is easy to see that the running time is bounded by a polynomial. □

Now we are ready to state our algorithm for the directed planar graph reachability problem. Consider any constant $\epsilon > 0$. We may assume that $\epsilon < 1/2$, because otherwise our target bounds, e.g., $\widetilde{O}(n^{1/2+\epsilon})$ become trivial. Let $G = (V, E)$, $s$, and $t$ be the given input; that is, $G$ is a directed graph, and $s$ and $t$ are start and goal vertices in $V$. As outlined in the introduction, our algorithm first uses `SepatorFamily` to compute an $n^{1-\epsilon}$-separator family $\overline{S}$ of size $O(n^{1/2+\epsilon/2})$ for the underlying undirected graph $\underline{G}$, and explores a path from $s$ to $t$ through vertices in $\overline{S}$. For any vertices $a$ and $b$ in $\overline{S}$, the reachability from $a$ to $b$ is determined by the reachability in $G(V' \cup \overline{S})$, where $V'$ is the set of vertices of some connected component of $\underline{G}(V \setminus \overline{S})$ that is adjacent to both $a$ and $b$ in $\underline{G}(V' \cup \overline{S})$. (Thus, we can immediately determine that $b$ is not reachable from $a$ if there is no such connected component $V'$ of $\underline{G}(V \setminus \overline{S})$.) For checking this connectivity, we may use the standard algorithm `BFS` for the reachability problem if $G(V' \cup \overline{S})$ is small enough. Although Lemma 3.2.1 guarantees that $|V' \cup \overline{S}| = O(n^{1-\epsilon})$, it is still large to execute `BFS`. Instead we use our algorithm recursively on $G(V' \cup \overline{S})$. Note further that $|V'|$ is too large to store; thus, $V'$ (and hence, $G(V' \cup \overline{S})$) must be given implicitly. In the algorithm, we specify $V'$ as a set of vertices connected to some vertex in $\underline{G}(V \setminus \overline{S})$; then we can check whether $v \in V'$ for a given $v$ by using the undirected graph reachability algorithm `UReach` in $O(\log n)$ space and polynomial time. Hence, the algorithm can be modified to handle such implicitly given graphs with the same order of space complexity (whereas some big polynomial-time overhead may be necessary).

This is the outline of our algorithm, and in Figure 1 we describe it in a pseudo code. For solving an instance $(G, s, t)$ of the planar graph reachability problem, it suffices to call `PlanarReach`$(G, s, t, n)$.

We analyze the space and time complexity of this algorithm; let $\mathcal{S}$ and $\mathcal{T}$ denote its space and time complexity functions. First note that, since $(1-\epsilon)^k \le 1/2$ for $k = O(1/\epsilon)$, the depth of recursion is $O(1/\epsilon)$, which is a constant.

We begin with the space complexity. Our goal is to show that $\mathcal{S}(n) = \widetilde{O}(n^{1/2+\epsilon})$. Consider $\mathcal{S}(\widehat{n})$ for any $\widehat{n} > n^{1/2}$. Line 5 uses space $\widetilde{O}(\widehat{n}^{1/2+\epsilon/2})$. For Line 6, we run `ImplicitBFS` on[7] $(\overline{S} \cup \{\widehat{s}, \widehat{t}\}, \overline{E})$, and its main computation can be done by using

[7]Though we use $\overline{E}$ to denote symbolically the set of edges of $G(V' \cup \overline{S})$, this set is not used in the algorithm explicitly.

1. `PlanarReach`$(\widehat{G}, \widehat{s}, \widehat{t}, n)$
   (let $\widehat{n}$ be the number of vertices of $\widehat{G}$)
2. **If** $\widehat{n} \leq n^{1/2}$
3. **then** `BFS`$(\widehat{G}, \widehat{s}, \widehat{t})$
4. **Else** (let $\widehat{r} = \widehat{n}^{1-\epsilon}$)
5. Run `SepatorFamily` to compute $\widehat{r}$-separator family $\overline{S}$
6. Run `ImplicitBFS`$((\overline{S} \cup \{\widehat{s}, \widehat{t}\}, \overline{E}), \widehat{s}, \widehat{t})$
   // `ImplicitBFS` executes in the same way as `BFS`
   // except for the case "$(a, b) \in \overline{E}$?" is queried,
   // i.e., it is asked whether $G(V' \cup \overline{S})$ has an
   // edge $(a, b)$? In this case the query is answered
   // by the following process 6.1 $\sim$ 6.5.
6.1. **For** every $x \in V$
   // $V_x$ = the set of vertices of $\underline{G}(V \setminus \overline{S})$'s
   // connected component containing $x$.
6.2. **If** `PlanarReach`$(G(V_x \cup S), a, b, n)$ is True
6.3. **then** Return True for the query
6.4. **End_For**
6.5. Return False for the query

Figure 3.21: Algorithm for the Directed Planar Reachability

$\widetilde{O}(|\overline{S}|)$ space like the standard `BFS`. On the other hand, for each query $(a, b) \in \overline{E}$ asked in the computation, we need to run Lines 6.1 $\sim$ 6.5, and the space needed for this computation is essentially $\mathcal{S}(|V_x| + |S|) \leq \mathcal{S}(2\widehat{n}^{1-\epsilon})$ for Line 6.2. Hence we get the following recurrence.

$$\mathcal{S}(\widehat{n}) = \begin{cases} \widetilde{O}\left(\widehat{n}^{1/2+\epsilon/2}\right) + \mathcal{S}\left(2\widehat{n}^{1-\epsilon}\right) & \text{if } \widehat{n} > n^{1/2}, \\ \widetilde{O}\left(n^{1/2}\right) & \text{otherwise.} \end{cases}$$

Since the recursion depth is bounded by $O(1/\epsilon)$, it is easy to see that $\mathcal{S}(n) = O(1/\epsilon)\widetilde{O}(n^{1/2+\epsilon/2}) = \widetilde{O}(n^{1/2+\epsilon/2})$, which is sufficient for our goal.

Next consider the time complexity. We need to be precise only up to a polynomial factor. Then by analysis similar to the above, we have the following recurrence.

$$\mathcal{T}(\widehat{n}) = \begin{cases} q(n)\left(p_1(\widehat{n})\mathcal{T}\left(2\widehat{n}^{1-\epsilon}\right) + p_2(\widehat{n})\right) & \text{if } \widehat{n} > n^{1/2}, \\ q(n)\widetilde{O}\left(n^{1/2}\right) & \text{otherwise.} \end{cases}$$

Here $p_1(\widehat{n})$ is the number of times we make the recursive calls of Line 6.2, and $p_2(\widehat{n})$ is the time needed by `SepatorFamily` at Line 5. On the other hand, a polynomial $q(n)$ is for the overhead when $\widehat{G}$ is given implicitly. Again from the $O(1/\epsilon)$ bound for the recursion depth, it is easy to see the bound $\mathcal{T}(n) = p(n)^{O(1/\epsilon)}$ holds for some polynomial $p(n)$.

Finally we argue the correctness of the algorithm. Consider the execution of `PlanarReach`$(G, s, t, n)$, and let $H$ denote the graph $(\overline{S} \cup \{s, t\}, \overline{E})$ given at Line 6 in this execution, where $\overline{E}$ denotes the set of pairs $(a, b)$ of $\overline{S} \cup \{s, t\}$ such that the process of Line 6.1 $\sim$ 6.5 returns true. We inductively assume that this process returns true if and only if there is a directed path from $a$ to $b$ in the induced graph $G(V_x \cup \overline{S})$ for some set of vertices $V_x$ connected to $x$ in $\underline{G}(V \setminus \overline{S})$. We claim that there is a directed path from $s$ to $t$ in $G$ if and only if there is a directed path from $s$ to $t$ in $H$. The if-part is trivial because every directed edge of $H$ corresponds to some directed path in $G$. Thus, we consider the only-if-part. Let $p$ be a path from $s$ to $t$ in $G$. We can decompose $p$ into $p_s p_1 p_2 \cdots p_l p_t$. Path $p_s$ is the part of $p$ that starts at $s$ and enters $\overline{S}$ at the very first time. Let $y_1$ be the end vertex of $p_s$. In general $p_i$ is the part of $p$ that starts at $y_i \in \overline{S}$ and ends in $y_{i+1}$ where $y_{i+1}$ is the first vertex where the path $p$ reenters $\overline{S}$ (after leaving $\overline{S}$). $p_t$ is the part of the path that starts at $y_{l+1}$ and ends in $t$. Notice that since $\overline{S}$ is a separator family, $p_i$ completely lies inside the subgraph $G(V_x \cup \overline{S})$ for some $x$. Because of this observation, $H$ has an edge $(y_i, y_{i+1})$ for every $i$, $1 \leq i \leq l$, and also edges $(s, y_1)$ and $(y_{l+1}, t)$. Hence $s y_1 y_2 \cdots y_{l+1} t$ is a path in $H$ from $s$ to $t$, as desired.

# Chapter 4

# $\widetilde{O}(\sqrt{n})$-space algorithm

## 4.1 Planar graph reachability algorithm

### 4.1.1 Preliminaries

Let $G = (V, E)$ denote an arbitrary directed graph. For any subset $U$ of $V$ we use $G[U]$ to denote the subgraph of $G$ induced by $U$. For two graphs $G_1$ and $G_2$, by $G_1 \cup G_2$ we mean the graph $(V(G_1) \cup V(G_2), E(G_1) \cup E(G_2))$. 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, $H \sqcup_G v = G[V(H) \cup \{v\}]$. Similarly, for an arbitrary set $A \subset V$, we use $H \sqcup_G A$ to denote $G[V(H) \cup A]$.

Recall that 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. We note that, in assuming that the input of our algorithm is a planar graph, we assume only that a planar embedding exists, not that it is given as part of the input.

### 4.1.2 The algorithm

We now describe our algorithm for planar graph reachability. To illustrate the idea of the algorithm we consider first the case in which the input graph $G$ is a subgraph of a bi-directed grid graph. 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. Note that both $G$ and $`G$ have $n = (2^h - 1) \times (2^h - 1)$ vertices.

Although reachability is determined by the edges of $G$, the computation is designed based on the underlying 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 **??**(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 **??**(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 $G[S \cup V^0]$. Next we can find that $w_2$ is reachable (from $v_0$) by searching vertices in $S$ that are reachable in $G[S \cup V^1]$ 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 $G[S \cup V^1]$ 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.

Since a path can cross the separator $\Theta(\sqrt{n})$ times, we cannot avoid making as many recursive calls at each level of recursion as there are vertices in the separator at that level. Consequently, without some modification the algorithm as described cannot hope to terminate in time bounded by some polynomial in $n$. In order to achieve polynomial-time computability, we introduce the idea of "budgeted recursion": the computation time allocated to individual recursive calls for checking the reachability in each subgraph is restricted in accordance with a predetermined sequence. Since the time required to trace a connecting path within an individual subgrid is not known in advance, we rely on a universality property of the sequence: eventually every subproblem will be allocated a budget sufficiently large to complete the required computation within that subproblem.

Asano and Kirkpatrick [4] describe the construction of a "universal sequence" suitable for this purpose. A similar universal sequence has been used in the context of oblivious algorithms; The notion of a universal sequence see [12], for example. Here we consider the following version.

For any $s \geq 0$, the *universal sequence* $\sigma_s$ *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 $\sigma_s$ 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. Their proof is straightforward, and is omitted here due to space constraints.

**Lemma 4.1.1.** *(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* $\sigma_s$ *such that* $d_j \leq c_{i_j}$ *holds for all* $j \in [x]$*; (b) the sequence* $\sigma_s$ *contains exactly* $2^{s-i}$ *appearances of the integer* $2^i$*, for all* $i \in [s]$*, and nothing else; and (c) the sequence* $\sigma_s$ *is computable in* $O(2^s)$*-time and* $\widetilde{O}(1)$*-space.*

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). In section 4.2 we outline how such a simple cycle-separator can be efficiently computed from any given separator. Based on this we have the following lemma.

**Lemma 4.1.2.** *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 '$G$ is used for defining the subgraphs.) ¿From technical reason[1], however, we need to add some edges to the outer faces of '$G[V^b]$ to get it triangulated under the current embedding. We can show that the number of added edges is bounded by $O(|S|)$ and that there is an algorithm `AddTriEmbed` 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 $['G]^b_{S,T}$ we mean both a graph obtained from '$G[V^b]$ 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

[1]The algorithm `CycleSep` is defined based on the separator algorithm of Lemma 4.2.1 that assumes a triangulated graph as input. Thus, in order to apply `CycleSep` to divide '$G[V^b]$ further, we need to get it triangulated.

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 it is easy to identify a depth $t$ subarea by using $\mathbf{S}$; for a given $\mathbf{S}$, we can determine whether $v \in V$ 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 grid-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 confirmed. Besides these data in the global variables, the procedure takes arguments $\mathbf{S}$ and $\ell$, 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 $G[\texttt{A} \cup V_{\mathbf{S}}]$ 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 $\text{`}G$ 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 graph reachability algorithm: (1) Set `A` $\leftarrow \{v_0, v_*\}$, `R`$[v_0] \leftarrow$ **true**, and `R`$[v_*] \leftarrow$ **false**; and (2) Execute `ExtendReach`$(\langle\,\rangle, 2^{\lceil \log n \rceil})$, and then output `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 $\ell$ (together with data kept in its global variables). Let $V_{\mathbf{S}}$ be the set of vertices of the subarea $[\text{`}G]_{\mathbf{S}}$ specified by $\mathbf{S}$. There are two cases. If $V_{\mathbf{S}}$ has less than $144c_{\text{sep}}^2$ vertices, then the procedure updates the value of `R` in a straightforward way. As we will see later $G[\texttt{A} \cup V_{\mathbf{S}}]$ 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 $[\text{`}G]_{\mathbf{S}}$ 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 $\sigma_s$ to control the length of paths in recursive calls.

The correctness of Algorithm 1 is demonstrated in Lemma 4.1.3 below. ¿From this, as summarized in Theorem 4.1.4, it is clear that our algorithm correctly determines

**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 $v \in$ `A`.

**Task:** For each vertex $v \in$ `A`, set `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 $u \in$ `A` whose value `R`$[u]$ before the current procedure execution equals `true`.
// Invariant: `A` $= \{v_0, v_*\} \cup \bigcup_{i \in [t]} S_i$. `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 $v \in$ `A` **do**
4: `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 `AddTriEmbed` to create a new cycle separator $S_{t+1}$ of $[\!`G]_{\mathbf{S}}$ and its additional triangulation edge lists $T_{t+1}^0$ and $T_{t+1}^1$;
8: $S'_{t+1} \leftarrow S_{t+1} \setminus \mathtt{A}$; `A` $\leftarrow$ `A` $\cup S'_{t+1}$;
9: `R`$[v] \leftarrow$ `false` for each vertex $v \in S'_{t+1}$;
10: **for** each $c_i = 2^{s_i}$ in the universal sequence $\sigma_s$ (where $i \in [2^{s+1} - 1]$) **do**
11: `ExtendReach`$(\langle (b_1, S_1, T_1), \ldots, (b_2, S_t, T_t), (0, S_{t+1}, T_{t+1}^0) \rangle, c_i)$;
12: `ExtendReach`$(\langle (b_1, S_1, T_1), \ldots, (b_2, S_t, T_t), (1, S_{t+1}, T_{t+1}^1) \rangle, c_i)$;
13: **end for**
14: `A` $\leftarrow$ `A` $\setminus S'_{t+1}$;
15: **end if**

the reachability of vertex $v_*$ from vertex $v_0$ in the input graph $G$.

**Lemma 4.1.3.** *For any input instance $G$, $v_0$, and $v_*$ of the planar graph reachability problem, 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* `A` *before the execution,* `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 $u \in$* `A` *whose value* `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 `A`, (ii) $h \leq 2^s$, and (iii) `R`$[u_0]$ = `true` before executing the procedure. For the lemma, it suffices to show that `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 4.1.1(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 $\sigma_s$ 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:

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

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 4.1.4.** *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 graph 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 4.1.3. For the complexity analysis, we consider the essential part, that is, the execution of $\mathtt{ExtendReach}(\langle\,\rangle, 2^{\lceil \log n\rceil})$.

As a key step for estimating the space and time bounds, we show here a bound on 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 $t_{\max} \leq c_{\mathrm{depth}} \log n$ holds with some constant $c_{\mathrm{depth}}$. Consider any depth $t$ recursive call of $\mathtt{ExtendReach}$; in other words, the execution of $\mathtt{ExtendReach}(\mathbf{S}, \ell)$ with a sequence $\mathbf{S}$ of length $t$. (Thus, the inital call of $\mathtt{ExtendReach}$ 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 $n_t \geq 513c_{\mathrm{sep}}^2$. Then two smaller subareas of $[`G]_{\mathbf{S}}$ are created and $\mathtt{ExtendReach}$ is recursively executed on them. Let $n_{t+1}$ denote the number of vertices of a larger one of these two smaller subareas. Then by Lemma 4.1.2 we have $n_{t+1} \leq 30n_t/31 + c_{\mathrm{sep}}\sqrt{n_t} \leq 31n_t/32$ since $n_t \geq 513c_{\mathrm{sep}}^2$. Hence, $t_{\max}$ is bounded by $c_{\mathrm{depth}} \log n$ as desired because $n(30/31)^{c_{\mathrm{depth}} \log n} < 513c_{\mathrm{sep}}^2$ for some constant $c_{\mathrm{depth}}$.

We bound the memory space used in the execution $\mathtt{ExtendReach}(\langle\,\rangle, 2^{\lceil \log n\rceil})$. For this, 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 4.1.2 we have

$$\begin{aligned}|\mathtt{A} \setminus \{v_0, v_*\}| &\leq \sum_{i\in[t_{\max}]} |S_i| \leq \sum_{i\in[t_{\max}]} c_{\mathrm{sep}}\sqrt{n_i} \\ &\leq \sum_{i\in[t_{\max}]} c_{\mathrm{sep}}\sqrt{\left(\frac{31}{32}\right)^{i-1} n} \leq (c_{\mathrm{sep}}\sqrt{n})\left(\sum_{i\geq 0}\left(\frac{31}{32}\right)^{i/2}\right) = O(\sqrt{n}),\end{aligned}$$

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 $\sigma_s$ (Lemma 4.1.1(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. □ □

## 4.2 Cycle-separators

### 4.2.1 Intuition of cycle separator

We expand here the notion of a cycle-separator and Lemma 4.1.2 used in the previous section. Throughout this section, we consider only undirected graphs. In particular, we fix any sufficiently large planar undirected graph $G = (V, E)$ and discuss a cycle-separator for $G$; all symbols using $G$ are used to denote some graph related to $G$.

Roughly speaking, a cycle-separator is a separator $S$ consisting of cycles. In this paper, we assume some orientation for each cycle, and our cycle-separators is required to separate $G$ into two subgraphs by considering a part located left (resp., right) of cycles (*cf.* Figure **??**).

Recall that we do not assume that our input graph comes equipped with a planar embedding. This is unnecessary for our purposes since Allender and Mahajan [1]

showed that the problem of computing a planar embedding can be reduced to the undirected graph reachability problem. Hence, by using the algorithm `UReach` of Reingold, we can compute a planar embedding of $G$ by using $O(\log n)$-space. Also it is also easy to to obtain some triangulation w.r.t. this embedding, and thus we may assume an $O(\log n)$-space algorithm that computes a triangulated planar embedding for a give planar graph. In our reachability algorithm, this $O(\log n)$-space algorithm is used (implicitly) before starting the actual computation and so in the description that follows we proceed as though our input graph $G$ is given with some triangulated planar embedding.

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*. Imai et al. has shown an algorithm that computes a 30/31-*separator* by using $O(\sqrt{n})$-space and in polynomial-time. Though we use such a separator algorithm as a blackbox, we introduce some modification so that we can 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 $\tau$ 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 30/31-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 $|\mathrm{V}((G)^0_{S\tau})| \leq 30|V|/31$ and $|\mathrm{V}((G)^1_{S\tau})| \leq 30|V|/31$ hold with $\tau = \{v_1, \ldots, v_k\}$. In summary, we have the following separator algorithm that is the basis of our cycle-separator algorithm.

**Lemma 4.2.1.** *There exists an $\widetilde{O}(\sqrt{n})$-space and polynomial-time algorithm that yields a 30/31-labeled-separator of size $\leq c_{\mathrm{sep}}\sqrt{n}$ for a given planar graph, where $c_{\mathrm{sep}}$ is some constant, which has been used in the previous section.*

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*. With this orientation, we define the left and the right of the cycle $C$. Our main technical lemma [[ here we should cite the expanded version in the archive]] is to show a way to compute a set $S'$ of cycles from a given separator $S$ that can be used as a separator in $\widetilde{O}(|S|)$-space and polynomial-time. More specifically, by using cycles in $S'$, we define two subsets $V^0$ and $V^1$ of $V$ as sets of vertices respectively located left and right of the cycles. Then they satisfy Lemma 4.1.2; that is, $G[V^0]$ and $G[V^1]$ are subgraphs covering $G$ and sharing only vertices in $S'$, which corresponds to '$G^0$, '$G^1$, and the separator $S$ in the grid case.

Furthermore, their size is (approximately) bounded by $2n/3$, and since $S' \subseteq S$ as a set, we have $|S'| \leq c_{\mathrm{sep}}\sqrt{n}$. This $S'$ is called a *cycle-separator* in this paper. We also provide a way to identify graphs $G[V^0]$ and $G[V^1]$, which is used as a basis of an algorithm identifying $[`G]_{\mathbf{S}}$.

### 4.2.2 Definition and construction of a cycle-separator

We give some explanation omitted in Section 3. We restate some statements for the sake of readability.

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 [10, 9] and Gazit and Miller [7], we can design $O(\sqrt{n})$-space and polynomial-time separator algorithm that yields a 30/31-*separator* [9]. We use such a separator algorithm as a black box in this paper, but we introduce here some 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 $\tau$ 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 30/31-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 $|\mathrm{V}((G)^0_{S\tau})| \leq 30|V|/31$ and $|\mathrm{V}((G)^1_{S\tau})| \leq 30|V|/31$ hold with $\tau = \{v_1, \ldots, v_k\}$. The following lemma summarize these observations.

**Lemma 4.2.2.** *There exists an $\widetilde{O}(\sqrt{n})$-space and polynomial-time algorithm that yields a 30/31-labeled-separator of size $\leq c_{\mathrm{sep}}\sqrt{n}$ for a given planar graph, where $c_{\mathrm{sep}}$ 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.

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 B.3.

**Lemma 4.2.3.** *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) *$S' \subseteq S$, $\mathrm{V}((G)_{S'\tau'}) \subseteq \mathrm{V}((G)_{S\tau}) \cup S$ 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

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

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 $E^{1-b}(S')$ (*cf.* Figure **??**). We will show (i.e., Lemma 4.2.5) that $G[V^b_{S'}]$ indeed represents $(G)_{S'\tau'} \sqcup_G S'$.

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 4.2.4.** *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 $(G)_{S'\tau'}$.*

*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 $E^1_{(u,v)} = \{\{u,w_1\},\dots,\{u,w_i\}\}$ 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 $w_2,\dots,w_i$, 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 4.2.5.** *For each $b \in \{0,1\}$, we have $\mathrm{V}((G)_{S'\tau'}) \cup S' = V^b_{S'}$.*

*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 $\{C_1,\dots,C_h\}$ 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 $\tau'$; that is, $G[V^b_{S'}] = (G)_{S'\tau'} \sqcup_G S'$ 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 4.1.2 is immediate. By using the algorithm given in Lemma 4.2.1, 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 $|V^b_{S'}| \leq |\mathrm{V}((G)_{S\tau})| + |S| \leq (2/3)|V| + c_{\mathrm{sep}}\sqrt{n}$ holds (Lemma 4.2.3(1)). The computation can be done using $O(\sqrt{n})$-space and in polynomial-time.

### 4.2.3 Depth $t$ subarea

In this subsection we use $S$ to denote a cycle-separator. Intuitively, we can use $G[V_S^0]$ and $G[V_S^1]$ 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 $G[V_S^b]$ is triangulated; but we need to triangulate its outer face. The following lemma gives a way to do this triangulation.

**Lemma 4.2.6.** *There exists a polynomial-time algorithm (which we refer as* `AddTriEmbed`*) that, for given undirected graph $G$ with its planar embedding, its cycle-separator $S$, and $b \in \{0, 1\}$, computes edges added to triangulate $G[V_S^b]$ 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 $[G]_{S,T}^b$ denote a graph obtained from $G[V_S^b]$ by adding triangulation edges specified by $T$.*

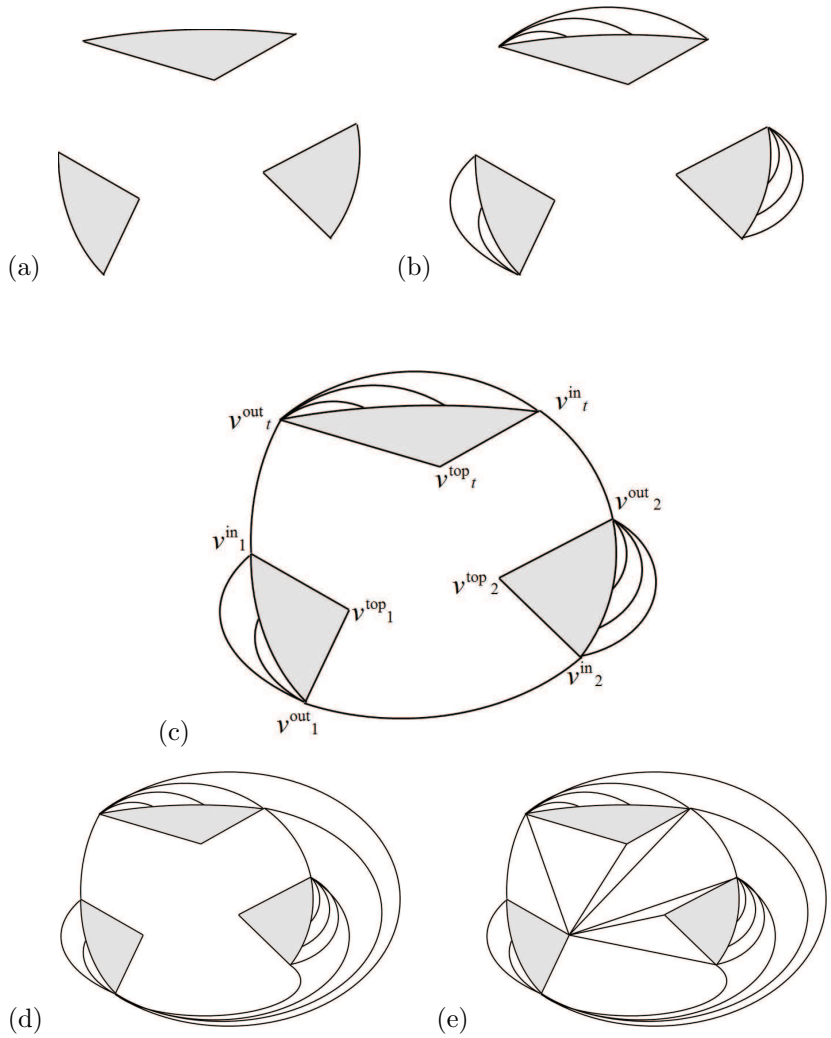
*Proof.* We explain a way that the algorithm `AddTriEmbed` adds new edges to vertices in $S$. The way to modify $\mathrm{N}(v)$ follows easily, and we omit explaining it. In the algorithm, we need to identify which vertex (resp., edge) of $G$ belongs to $G[V_S^b]$; we provide $O(\log n)$-space algorithms for these tasks in Lemma 4.2.7.

The algorithm adds edges for every "face" of $G[V_S^b]$, namely, a connected component of $G$ that is separated from $G[V_S^b]$ by cutting edges in $E^{1-b}(S)$. Consider one of such faces, and let $F$ denote it. The algorithm identifies all cycles, say, $C_1, \ldots, C_r$, of $S$ facing to $F$ (Figure 4.1(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 4.1(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 $C_1, \ldots, C_r$.

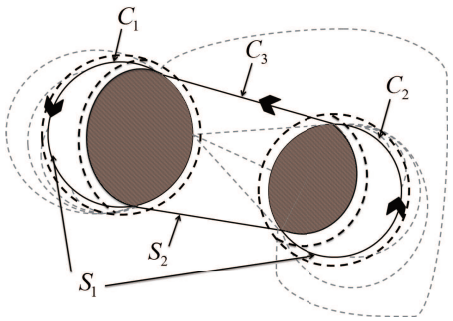
Consider now triangle $C_1$. We assume (by reversing the order if necessary) that the face is on the left side of $C_1$ under the cycle representation of its three vertices, and let $\langle u, v, w \rangle$ be this cycle representation. We name $u$, $v$, and $w$ as $v_1^{\mathrm{out}}$, $v_1^{\mathrm{in}}$, and $v_1^{\mathrm{top}}$ respectively. Similarly, we give names to three vertices of the other cycles. Then the algorithm adds an edge connecting $v_1^{\mathrm{out}}$ and $v_2^{\mathrm{in}}$, $v_2^{\mathrm{out}}$ and $v_3^{\mathrm{in}}$, ..., $v_t^{\mathrm{out}}$ and $v_1^{\mathrm{in}}$ (Figure 4.1(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 $v_1^{\mathrm{out}}$ and all other vertices of $C_{\mathrm{out}}$; Figure 4.1(d)). For the latter face, edges are added to connect $v_1^{\mathrm{out}}$ and all other vertices of $C_{\mathrm{in}}$; Figure 4.1(e)).




This figure shows how edges are added at each step of the triangulation algorithm explained in Lemma 4.2.6. 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 4.1: 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 4.2: The idea of testing whether $v$ exists in $[G]^{01}_{(S_1,T_1),(S_2,T_2)}$

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:

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

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 4.2.7.** *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 4.2. 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

$$v \in V_t \;\Leftrightarrow\; v \text{ is connected to some vertex in } S_t \text{ in } G[E \cup \widetilde{T}_t \setminus \widetilde{E}_t] \tag{4.1}$$

holds inductively.

The case where $t = 1$ is explained right after the proof of Lemma 4.2.5. (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 (4.1) 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 (4.1).

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 (4.1) 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. □

### 4.2.4 Existence and computability of cycle separator

We give a proof of our main technical lemma.

First, define $S'$ and $\tau'$. 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 $E'_{\mathrm{all}}$ be the set of all such border edges. We define $S'$ to be the set of end points of $E'_{\mathrm{all}}$. 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 $\tau'$ by removing all $v_j$ 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 $E'_{\text{all}}$ 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.

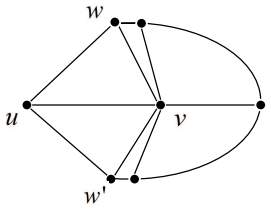
**Claim 1.** *Degree of any vertex of $G[E']$ is at least 2.*

*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 4.3); 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 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




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

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 4.4(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 4.4(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

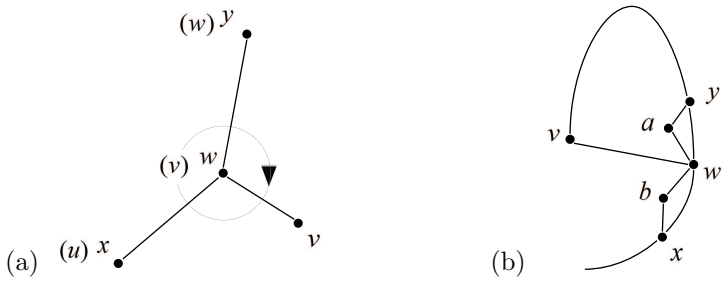



Figure 4.4: Situation when $w$ is selected twice (for Claim 3)

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 $E'_{\mathrm{all}}$ is chosen in $E'$ for some $K'_i$. Hence, every edge in $E'_{\mathrm{all}}$ appears in some cycle; on the other hand, only edges in $E'_{\mathrm{all}}$ 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 $E'_{\mathrm{all}}$, 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 4.2.3.

# Chapter 5

# Conclusion

In this chapter, we summarize the results of this dissertation, and discuss open problems and future work.

In the third chapter, we first showed a polynomial time $\widetilde{O}(\sqrt{n})$-space algorithm for computing a planar separator of a graph. We also provided a planar reachability algorithm that uses the separator and showed the tradeoff between time and space. The reachability algorithm, from chapter three, works in sublinear space and polynomial time but it needs super polynomial time to work in $\widetilde{O}(\sqrt{n})$ space.

In the forth chapter, we introduced the notion of a universal sequence and an algorithm for the planar reachability problem which works in $\widetilde{O}(\sqrt{n})$ space and polynomial time.

It should be noted that, though restricted to grid graphs, the problem studied by Asano et al. in [2, 4] is the shortest path problem, a natural generalization of the reachability problem. In order to keep the discussion in this thesis as simple as possible, and focus on the key ideas, we have restricted our attention here to the graph reachability problem. However, it is not hard to see that our algorithms for the reachability problem can be modified to solve shortest path problem (with a modest increase in the polynomial time bound).

Similarly, the focus in Asano et al. in [2, 4] is on space efficient and yet practically useful algorithms, and they also include time-space tradeoffs. In this thesis, on the other hand, our motivation has been to extend a graph class that is solvable in $\widetilde{O}(\sqrt{n})$-space and polynomial-time, while the specific time complexity of the algorithms is not so important so long as it is polynomial. In fact, since the algorithm of Reingold for the undirected reachability is used heavily in this thesis, we need a very large polynomial to bound our algorithms' running time. So there is wide room for improvement on the time complexity of these algorithms.

Since we use Reingold's undirected reachability algorithm, our algorithms have no natural implementation in the NNJAG model. While the worst-case instances for NNJAG given in [6] 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 a model in which our algorithm can be naturally implemented and show a bound of the space efficiency of

the computation.

# Bibliography


[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, in *Proc. of the 23th Canadian Conf. on Comp. Geometry* (CCCG'93), 2011.

[3] T. Asano, D. Kirkpatrick, K. Nakagawa and O. Watanabe $O(\sqrt{n})$-Space and Polymomial-time Algorithm for the Planar Directed Graph Reachability Problem, in *Proc. of 39th International Symposium on Mathematical Foundations of Computer Science* PartII 45–56, 2014.

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

[5] 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*, IEEE Computer Society press, 27–33, 1992.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

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

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

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

[13] O. Reingold, Undirected connectivity in log-space, *J. ACM*, 55(4), 1–24, 2008.

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

[15] R.Diestel, *Graph Theory*, Springer, Graduate Texts in Mathematics, 2010.