The directed graph reachability problem takes as input an \( n \)-vertex directed graph \( G = (V,E) \), and two distinguished vertices \( s \) and \( t \). The problem is to determine whether there exists a path from \( s \) to \( t \) in \( G \). This is a canonical complete problem for class NL. Asano et al. proposed an \( \tilde{O}(\sqrt{n}) \) space and polynomial time algorithm for the directed grid and planar graph reachability problem. The main result of this paper is to show that the directed graph reachability problem restricted to grid graphs can be solved in polynomial time using only \( \tilde{O}(n^{1/3}) \) space.

1 Introduction

The graph reachability problem, for a graph \( G = (V,E) \) and two distinct vertices \( s,t \in V \), is to determine whether there exists a path from \( s \) to \( t \). This problem characterizes many important complexity classes. The directed graph reachability problem is a canonical complete problem for the nondeterministic log-space class, NL. Reingold showed that the undirected graph reachability problem characterizes the deterministic log-space class, L\(^{10}\). As with P vs. NP problem, whether L=NL or not is a major open problem. This problem is equivalent to whether the directed graph reachability problem is solvable in deterministic log-space. There exist two fundamental solutions for the directed graph reachability problem, breadth first search, denoted as BFS, and Savitch’s algorithm. BFS runs in \( O(n) \) space and \( O(m) \) time, where \( n \) and \( m \) are the number of vertices and edges, respectively. For Savitch’s algorithm, we use only \( O(\log^2 n) \) space but require \( \Theta(n \log n) \) time. BFS needs short time but large space. Savitch’s algorithm uses small space but super polynomial time. A natural question is whether we can make an efficient deterministic algorithm in both space and time for the directed graph reachability problem. In particular, Wigderson proposed a problem that does there exist an algorithm for the directed graph reachability problem that uses polynomial time and \( O(n^\varepsilon) \) space, for some \( \varepsilon < 1 \)\(^{13}\), and this question is still open. The best known polynomial time algorithm, shown by Barns, Buss, Ruzzo and Schieber, uses \( O(n/2^{\log n}) \) space \(^4\).

For some restricted graph classes, better results are known. Stolee and Vinodchandran showed that for any \( 0 < \varepsilon < 1 \), the reachability problem for directed acyclic graph with \( O(n^\varepsilon) \) sources and embedded on a surface with \( O(n^\varepsilon) \) genus can be solved in polynomial time and \( O(n^\varepsilon) \) space \(^{11}\). A natural and important restricted graph class is the class of planar graphs. The planar graph reachability problem is hard for L, and in the unambiguous log-space class, UL\(^5\), which is a subclass of NL. Imai et al. gave an algorithm using \( O(n^{1/2+\varepsilon}) \) space and polynomial time for the planar graph reachability problem \(^2\,^{8}\). Moreover Asano et al. devised a efficient way to control the recursion, and proposed a polynomial time and \( \tilde{O}(\sqrt{n}) \) space algorithm for the planar graph

\(^1\)In this paper “\( \tilde{O}(s(n)) \) space” means \( O(s(n)) \) words intuitively and precisely \( O(s(n) \log n) \) space.
reachability problem [3]. In this paper, we focus on the grid graph reachability problem, where grid graphs are special cases of planar graphs. Allender et al. showed the planar graph reachability problem is log-space reducible to the grid graph reachability problem [1]. By using the algorithm of Asano et al., we can solve the grid graph reachability problem in $\tilde{O}(\sqrt{n})$ space and polynomial time. The main result of this paper is to show an $\tilde{O}(n^{1/3})$ space and polynomial time algorithm for the directed grid graph reachability problem.

Theorem 1 ([3]). There exists an algorithm that decides directed planar graph reachability in polynomial time and $\tilde{O}(\sqrt{n})$ space. (We refer to this algorithm by PlanarReach in this paper.)

2 Preliminaries and an Outline of the Algorithm

We will use the standard notions and notations for algorithms, complexity measures, and graphs without defining them. We consider mainly directed graphs, and a graph is assumed to be a directed graph unless it is specified as an undirected graph. Throughout this paper, for any set $X$, $|X|$ denotes the number of elements in $X$. We refer to the maximum and minimum elements of $X$ as $\max X$ and $\min X$, respectively. Consider any directed graph $G = (V, E)$. For any $u, v \in V$, a directed edge $e$ from $u$ to $v$ is denoted as $e = (u, v)$; on the other hand, the tail $u$ and the head $v$ of $e$ are denoted as $t(e)$ and $h(e)$, respectively. For any $U \subseteq V$, let $G[U]$ denote the subgraph of $G$ induced by $U$.

Recall that a grid graph is a graph whose vertices are located on grid points, and whose vertices are adjacent only to their immediate horizontal or vertical neighbors. We refer to a vertex on the boundary of a grid graph as a rim vertex. For any grid graph $G$, we denote the set of the rim vertices of $G$ as $R_G$.

Computational Model

For discussing sublinear-space algorithms formally, we use the standard multi-tape Turing machine model. A multi-tape Turing machine consists of a read-only input tape, a write-only output tape, and a constant number of work tapes. The space complexity of this Turing machine is measured by the total number of cells that can be used as its work tapes.

For the sake of explanation, we will follow a standard convention and give a sublinear-space algorithm by a sequence of constant number of sublinear-space subroutines $A_1, \ldots, A_k$ such that each $A_i$ computes, from its given input, some output that is passed to $A_{i+1}$ as an input. Note that some of these outputs cannot be stored in a sublinear-size work tape; nevertheless, there is a standard way to design a sublinear-space algorithm based on these subroutines. The key idea is to compute intermediate inputs every time when they are necessary. For example, while computing $A_i$, when it is necessary to see the $j$th bit of the input to $A_i$, simply execute $A_{i-1}$ (from the beginning) until it yields the desired $j$th bit on its work tape, and then resume the computation of $A_i$ using this obtained bit. It is easy to see that this computation can be executed in sublinear-space. Furthermore, while a large amount of extra computation time is needed, we can show that the total running time can be polynomially bounded if all subroutines run in polynomial-time.

Outline of the Algorithm

We show the outline of our algorithm. Our algorithm uses the algorithm PlanarReach for the planar graph reachability. We assume both $\sqrt{n}$ and $n^{1/3}$ are integers for simplicity. Let $G$ be an input $\sqrt{n} \times \sqrt{n}$ grid graph with $n$ vertices.

1. Separate $G$ into $n^{1/3} \times n^{1/3}$ small grid graphs, or “blocks”. There are $n^{1/3}$ blocks, and each block contains $n^{2/3}$ vertices.
2. Transform each block $B$ into a special planar graph, “gadget graph”, with $O(n^{1/3})$ vertices. The reachability among the vertices in $R_B$ should be unchanged. The total number of vertices in all blocks becomes $O(n^{2/3})$.

3. We apply the algorithm $\text{PlanarReach}$ to the transformed graph of size $O(n^{2/3})$, then the reachability is computable in $O(\sqrt{n^{2/3}}) = O(n^{1/3})$ space.

In step 1 and 2, we reduce the number of vertices in the graph $G$ while keeping the reachability between the rim vertices of each block so that we can solve the reachability problem of the original graph. Then to this transformed graph we apply $\text{PlanarReach}$ in step 3, which runs in $\tilde{O}(n^{1/3})$ space.

**Theorem 2.** There exists an algorithm that computes the grid graph reachability in polynomial-time and $\tilde{O}(n^{1/3})$ space.

The start vertex $s$ (resp., the end vertex $t$) may not be on the rim of any block. In such a situation, we make an additional block so that $s$ (resp., $t$) would be on the rim of the block. This operation would not increase the time and space complexity. In this paper, we assume that $s$ (resp., $t$) is on the rim of some block.

### 3 Graph Transformation

In this section, we explain an algorithm that modifies each block and analyze time and space complexity of the algorithm. Throughout this section, we let a directed graph $G_0 = (V_0, E_0)$ denote a block of the input grid graph, and let $V_0^{\text{rim}}$ denote the set of its rim vertices. We use $N$ to denote the number of vertices of the input grid graph and $n$ to denote $|V_0^{\text{rim}}|$, which is $O(N^{1/3})$; note, on the other hand, that we have $|V_0| = O(n^2) = O(N^{2/3})$. Our task is to transform this $G_0$ to a plane “gadget graph”, an augmented plane graph, $\tilde{G}_p$ with $O(n) = O(N^{1/3})$ vertices including $V_0^{\text{rim}}$ so that the reachability among vertices in $V_0^{\text{rim}}$ on $G_0$ remains the same on $\tilde{G}_p$.

There are two steps for this transformation. We first transform $G_0$ to a circle graph $G_0^{\text{cir}}$, and then obtain $\tilde{G}_p$ from the circle graph.

#### 3.1 Circle Graph

We introduce the notion of “circle graph”. A circle graph is a graph embedded on the plane so that all its vertices are placed on a cycle and all its edges are drawn inside of the cycle. Note that a circle graph may not have an edge between a pair of adjacent vertices on the cycle. We introduce some basic notions on circle graphs. Consider any circle graph $G = (V, E)$, and let $C$ be a cycle on which all vertices of $V$ are placed. For any $u, v \in V$, a clockwise tour (resp., anti-clockwise tour) is a part of the cycle $C$ from $u$ to $v$ in a clockwise direction (resp., in an anti-clockwise direction). We use $C^{\text{cl}}[u, v]$ (resp., $C^{\text{acl}}[u, v]$) to denote this tour (Figure 1(a)). When we would like to specify the graph $G$, we use $C^{\text{cl}}[u, v]$ (resp., $C^{\text{acl}}[u, v]$). The tour $C^{\text{cl}}[u, v]$, for example, can be expressed canonically as a sequence of vertices $(v_1, \ldots, v_k)$ such that $v_1 = u$, $v_k = v$, and $v_2, \ldots, v_{k-1}$ are all vertices visited along the cycle $C$ clockwise. We use $C^{\text{cl}}(u, v)$ and $C^{\text{cl}}(u, v)$ (resp., $C^{\text{cl}}(u, v)$ and $C^{\text{acl}}(u, v)$) to denote the sub-sequences $(v_2, \ldots, v_{k-1})$ and $(v_1, \ldots, v_{k-1})$ respectively. Note here that it is not necessary that $G$ has an edge between adjacent vertices in such a tour. The length of the tour is simply the number of vertices on the tour. An edge $(u, v)$ of $G$ is called a chord if $u$ and $v$ are not adjacent on the cycle $C$. For any chord $(u, v)$, we may consider two arcs, namely, $C^{\text{cl}}[u, v]$ and $C^{\text{acl}}[u, v]$; but in the following, we will simply use $C[u, v]$ to denote one of them that is regarded as the arc of the chord $(u, v)$ in the context. When necessary, we will state, e.g., “the arc $C^{\text{cl}}[u, v]$” for specifying which one is currently regarded as the arc. A gap-d (resp., gap-d+) chord is a chord
Figure 1: An example of the notions on chords. (a) a figure showing a chord, arcs, a lower area, an upper area, (b) a figure showing crossing chords (e₁ and e₂) and semi-crossing chords (e₃ and e₄) and (c) separating chords (e₃ separates e₁ and e₂).

(u, v) whose arc C[u, v] is of length d + 2 (resp., length ≥ d + 2). For any chord (u, v), the subplane inside of the cycle C surrounded by the chord (u, v) and the arc C[u, v] is called the lower area of the chord; on the other hand, the other side of the chord within the cycle C is called the upper area (see Figure 1(a)). A lowest gap-d⁺ chord is a gap-d⁺ chord that has no other gap-d⁺ chord in its lower area. We say that two chords (u₁, v₁) and (u₂, v₂) cross if they cross in the circle C in a natural way (see Figure 1(b)). Formally, we say that (u₁, v₁) crosses (u₂, v₂) if either (i) u₂ is on the tour Cclk[u₁, v₁] and v₂ is on the tour Cclk[u₁, v₁], or (ii) v₂ is on the tour Cclk[u₁, v₁] and u₂ is on the tour Cclk[u₁, v₁]. Also, we say that (u₁, v₁) semi-crosses (u₂, v₂) if either (i) u₂ is on the tour Ccl[u₁, v₁] and v₂ is on the tour Ccl[u₁, v₁], or (ii) v₂ is on the tour Ccl[u₁, v₁] and u₂ is on the tour Ccl[u₁, v₁] (see Figure 1(b)). Note that clearly crossing implies semi-crossing. In addition, we say that a chord (u₁, v₁) separates two chords (u₂, v₂) and (u₃, v₃) if the endpoints of two chords v₂ and v₃ are separated by the chord (u₁, v₁) (see Figure 1(c)). Formally, (u₁, v₁) separates (u₂, v₂) and (u₃, v₃) if either (i) v₂ is on the tour Ccl[u₁, v₁] and v₃ is on the tour Ccl[u₁, v₁], or (ii) v₃ is on the tour Ccl[u₁, v₁] and v₂ is on the tour Ccl[u₁, v₁]. We say that k chords (u₁, v₁), (u₂, v₂), . . . , (uₖ, vₖ) are traversable if the following two conditions are satisfied:

1. (u₁, v₁) semi-crosses (u₂, v₂),
2. ∀i ∈ [3, k], ∃p, q < i, (uᵢ, vᵢ) separates (uᵢ, vᵢ) and (uᵢ, vᵢ).

Now for the graph G₀ = (V₀, E₀), we define the circle graph Gcir₀ = (Vcir₀, Ecir₀) by

\[
Vcir₀ = V₁rim, \quad \text{and} \quad Ecir₀ = \{ (u,v) | \exists \text{path from } u \text{ to } v \text{ in } G₀ \},
\]

where we assume that the rim vertices of Vcir₀ (= V₁rim) are placed on a cycle C₀ as they are on the rim of the block in the grid graph. Then it is clear that Gcir₀ keeps the same reachability relation among vertices in Vcir₀ = V₁rim. Recall that G₀ has O(n²) vertices. Thus, by using PlanarReach, we can show the following lemma.

**Lemma 1.** Gcir₀ keeps the same reachability relation among vertices in Vcir₀ = V₁rim. That is, for any pair u, v of vertices of Vcir₀, v is reachable from u in Gcir₀ if and only if it is reachable from u in G₀. There exists an algorithm that transforms G₀ to Gcir₀ in O(n) space and polynomial-time in n.
Lemma 2. For a circle graph $G_0^{\text{cir}} = (V_0^{\text{cir}}, E_0^{\text{cir}})$ obtained from a block grid graph $G_0$, if there are traversable edges $(u_1, v_1), (u_2, v_2), \ldots, (u_k, v_k) \in E_0^{\text{cir}}$, then $(u_1, v_k) \in E_0^{\text{cir}}$.

Proof. We show that $v_k$ is reachable from $u_1$ in $G_0$ by induction on $k$. First, we consider the case $k = 2$, namely $(u_1, v_1)$ semi-crosses $(u_2, v_2)$. $G_0$ contains a path $p_{u_1,v_1}$ which goes from $u_1$ to $v_1$. Also, $G_0$ contains a path $p_{u_2,v_2}$ which goes from $u_2$ to $v_2$. Since $G_0$ is planar and $u_1, v_1, u_2, v_2$ are the rim vertices and the edges are semi-crossing, there exists a vertex $w$ which is common in $p_{u_1,v_1}$ and $p_{u_2,v_2}$ in $G_0$. Since $w$ is reachable from $u_1$ and $v_2$ is reachable from $w$, there exists a path from $u_1$ to $v_2$.

Next, we assume that the lemma is true for all sequences of traversable edges of length less than $k$. By the definition, there exist two edges $(u_p, v_p)$ and $(u_q, v_q)$ that the edge $(u_k, v_k)$ separates $(p, q < k)$. We have two paths $p_{u_1,v_p}$ from $u_1$ to $v_p$ and $p_{u_1,v_q}$ from $u_1$ to $v_q$ in $G_0$ by the induction hypothesis. Also we have a path $p_{u_k,v_k}$ from $u_k$ to $v_k$. Since $(u_k, v_k)$ separates $(u_p, v_p)$ and $(u_q, v_q)$, $v_p$ and $v_q$ are on the different sides of arcs of the edge $(u_k, v_k)$. If $u_1$ and $v_p$ are on the same arc of $(u_k, v_k)$, the paths $p_{u_1,v_p}$ and $p_{u_k,v_k}$ have a common vertex $w'$ (see Figure 2(a)). On the other hand, if $u_1$ and $v_q$ are on the same arc of $(u_k, v_k)$, the paths $p_{u_1,v_p}$ and $p_{u_k,v_k}$ have a common vertex $w'$ (see Figure 2(b)). Thus there exists a path from $u_1$ to $v_k$ via $w'$ in $G_0$. 

3.2 Gadget Graph

We introduce the notion of “gadget graph”. A gadget graph is a graph that is given a “label set” to each edge.

Definition 1. A gadget graph $\mathcal{G}$ is a graph defined by a tuple $(\mathcal{V}, \mathcal{E}, \mathcal{K}, \mathcal{L})$, where $\mathcal{V}$ is a set of vertices, $\mathcal{E}$ is a set of edges, $\mathcal{K}$ is a path function that assigns an edge or $\perp$ to each edge, and $\mathcal{L}$ is a level function that assigns a label set to each edge. A label set is a set $\{i_1 \to o_1, i_2 \to o_2, \ldots, i_k \to o_k\}$ of labels where each label $i_j \to o_j$, $i_j, o_j \in \mathbb{R} \cup \{\infty\}$, is a pair of in-level and out-level.

Remark. For an edge $(u, v) \in \mathcal{E}$, we may use expressions $\mathcal{K}(u, v)$ and $\mathcal{L}(u, v)$ instead of $\mathcal{K}((u, v))$ and $\mathcal{L}((u, v))$ for simplicity.
Our goal is to transform a given circle graph (obtained from a block grid graph) \( G_0^{\text{cir}} = (V_0^{\text{cir}}, E_0^{\text{cir}}) \) in which all vertices in \( V_0^{\text{cir}} \) are placed on a cycle \( C \) to a plane gadget graph \( \tilde{G}_p = (\tilde{V}_p^{\text{out}} \cup \tilde{V}_p^{\text{in}}, \tilde{E}_p, \tilde{K}_p, \tilde{L}_p) \) where \( \tilde{V}_p^{\text{out}} \) is the set of outer vertices that are exactly the vertices of \( V_0^{\text{cir}} \) placed in the same way as \( G_0^{\text{cir}} \) on the cycle \( C \), and \( \tilde{V}_p^{\text{in}} \) is the set of inner vertices placed inside of \( C \). All edges of \( \tilde{E}_p \) are also placed inside of \( C \) under our embedding. The inner vertices of \( \tilde{V}_p^{\text{in}} \) are used to replace crossing points of edges of \( E_0^{\text{cir}} \) to transform to a planar graph (see Figure 3). We would like to keep the “reachability” among vertices in \( \tilde{V}_p^{\text{out}} \) in \( \tilde{G}_p \) while bounding \( |\tilde{V}_p^{\text{in}}| = O(n) \).

We explain how to characterize the reachability on a gadget graph. Consider any gadget graph \( \tilde{G} = (\tilde{V}, \tilde{E}, \tilde{K}, \tilde{L}) \), and let \( x \) and \( y \) be any two vertices of \( \tilde{V} \). Intuitively, the reachability from \( x \) to \( y \) is characterized by a directed path on which we can send a token from \( x \) to \( y \). Suppose that there is a directed path \( p = (e_1, \ldots, e_m) \) from \( x \) to \( y \). We send a token through this path. The token has a level, which is initially \( \infty \) when the token is at vertex \( x \). (For a general discussion, we use a parameter \( \ell_s \) for the initial level of the token.) When the token reaches the tail vertex \( t(e_j) \) of some edge \( e_j \) of \( p \) with level \( \ell \), it can “go through” \( e_j \) to reach its head vertex \( h(e_j) \) if \( \tilde{L}(e_j) \) has an available label \( i_j \rightarrow o_j \) such that \( i_j \leq \ell \) holds for its in-level \( i_j \). If the token uses a label \( i_j \rightarrow o_j \), then its level becomes the out-level \( o_j \) at the vertex \( h(e_j) \). If there are several available labels, then we naturally use the one with the highest out-level. If the token can reach \( y \) in this way, we consider that a “token tour” from \( x \) to \( y \) is “realized” by this path \( p \). Technically, we introduce \( \tilde{K} \) so that some edge can specify the next edge. We consider only a path \( p = (e_1, \ldots, e_m) \) as “valid” such that \( e_{i+1} = \tilde{K}(e_i) \) for all \( e_i \) such that \( \tilde{K}(e_i) \neq \perp \). We characterize the reachability from \( x \) to \( y \) on gadget graph \( \tilde{G} \) by using a valid path realizing a token tour from \( x \) to \( y \).

**Definition 2.** For any gadget graph \( \tilde{G} = (\tilde{V}, \tilde{E}, \tilde{K}, \tilde{L}) \), and for any two vertices \( x, y \) of \( \tilde{V} \), there exists a token tour from \( x \) to \( y \) with initial level \( \ell_s \) if there exists a sequence of edges \( (e_1, \ldots, e_m) \) that satisfies

1. \( x = t(e_1) \) and \( y = h(e_m) \),
2. \( h(e_i) = t(e_{i+1}) \) (1 ≤ \( i < m \)),
3. if \( \tilde{K}(e_i) \) is not \( \perp \) (1 ≤ \( i < m \)), then \( e_{i+1} = \tilde{K}(e_i) \),
4. there exist labels \( i_1 \rightarrow o_1 \in \tilde{L}(e_1), \ldots, i_m \rightarrow o_m \in \tilde{L}(e_m) \) such that \( \ell_s \geq i_1 \) and \( o_t \geq i_{t+1} \) for all \( 1 \leq t < m \).
At the beginning of our algorithm, we obtain a gadget graph \( \tilde{G}_0 = (\tilde{V}_0, \tilde{E}_0, \tilde{K}_0, \tilde{L}_0) \) whose base graph is equal to \( G_0^{\text{ir}} \), and \( \tilde{K}_0(e) = \perp, \tilde{L}_0(e) = \{0 \to \infty\} \) for every \( e \in \tilde{E}_0 \). It is obvious that \( G_0^{\text{ir}} \) and \( \tilde{G}_0 \) have the same reachability. Namely, there exists a token tour from \( x \) to \( y \) for \( x, y \in \tilde{V}_0 \) in \( \tilde{G}_0 \) if and only if there exists an edge \((x, y) \in \tilde{E}_0\).

We explain first the outline of our transformation from \( \tilde{G}_0 \) to \( \tilde{G}_p \). We begin by finding a chord \( e_* = (u, v) \) with gap \( \geq 2 \) having no other gap-2\(^+\) chord in its lower area, that is, one of the lowest gap-2\(^+\) chords. (If there is no gap-2\(^+\) chord, then the transformation is terminated.) For this \( e_* \) and its lower area, we transform them into a planar part and reduce the number of crossing points as follows (see Figure 4): (i) Consider all edges of \( \tilde{G}_0 \) crossing this chord \( e_* = (e_1, e_2, e_3) \) in Figure 4. Create a new inner vertex \( v_* \) of \( \tilde{G}_0 \) on the chord and bundle all crossing edges going through this vertex \( v_* \); that is, we replace all edges crossing \( e_* \) by edges between their end points in the lower area of \( e_* \) and \( v_* \), and edges between \( v_* \) and their end points in the upper area of \( e_* \). (ii) Introduce new inner vertices for edges crossing gap-1 chords in the lower area of \( e_* \) (\( w \) in Figure 4). (iii) Add appropriate label sets to those newly introduced edges so that the reachability is not changed by this transformation. At this point we regard the lower area of \( e_* \) as processed, and remove this part from the circle graph part of \( \tilde{G}_0 \) by replacing the arc \( C[u, v] \) by a tour \((u,v_*,v)\) to create a new circle graph part of \( \tilde{G}_1 \). We then repeat this transformation step on the circle graph part of \( \tilde{G}_1 \). In the algorithm, \( U_t \) is the vertices of the circle graph part of \( \tilde{G}_t \); thus \( \tilde{G}_t[U_t] \) indicates the circle graph part of \( \tilde{G}_t \). Note that \( e_* \) is not removed and becomes a gap-1 chord in the next step.

We explain step (ii) for \( \tilde{G}_0 \) in more detail. Since \( e_* \) is a gap-2\(^+\) chord, there exist only gap-1 chords or edges whose one end point is \( v_* \) in the lower area of \( e_* \). If there are two edges \( e_0 \) and \( e_1 \) that cross each other, we replace the crossing point by a new inner vertex \( u \) (see Figure 5(a), (b)). The edge \( e_i \) becomes two edges \((t(e_i), u)\) and \((u, h(e_i))\) \((i = 0, 1)\), and we set \( \tilde{K}_1(t(e_i), u) = (u, h(e_i)) \). The edges might be divided into more than two segments (see Figure 5(c)). We call the edge of \( \tilde{G}_0 \) original edge of the divided edges. By the path function, we must move along the original edge. An edge \( e \) might have a reverse direction edge \( \tilde{e} = (h(e), t(e)) \) (see Figure 5(d)). In this case, \( e \) and \( \tilde{e} \) share a new vertex for resolving crossing points. For \( \tilde{G}_t[U_t] \) \((t > 0)\), we process the lower area in the same way. We refer to this algorithm as \text{MakePlanar}, and the new inner vertices created by \text{MakePlanar} in step \( t \) as \( V_t^{\text{MP}} \).

The detailed process of step (iii) is written in Algorithm 2 and Algorithm 3 describes the entire process of step (i), (ii) and (iii). The following lemma shows that an output graph of Algorithm 1 has small size.

**Lemma 3.** Algorithm 1 terminates creating a planar graph of size \( O(n) \).

**Proof.** In the beginning of the algorithm, \(|U_0| = n\) and \(|U_t|\) decreases by at least 1 for each iteration.
since the picked edge $e_s^*$ is a gap-$2^+$ chord. Hence the algorithm stops after at most $n$ iterations and the number of the new inner vertices made at line 7, or $v^*_s$, is also at most $n$. If a gap-$k$ chord is picked, we make at most $2k - 1$ new inner vertices by MakePlanar, namely $|V_{\text{MP}}^t| \leq 2k - 1$, since there exist only gap-1 chords in the lower area of the picked edge. The total number of inner vertices becomes at most
\[
n + \sum_{i=1}^{t}(2k_i - 1) = n + 2 \sum_{i=1}^{t} k_i - t \leq n + 2 \times 2n = 5n
\]
where $t$ is the number of iterations and $k_i$ means that a gap-$k_i$ chord was picked in the $i$-th iteration. After all, $|V_p^{\text{out}} \cup V_p^{\text{in}}| \leq n + 5n = 6n$.

Now we explain Algorithm 2 describing how to assign labels to $\widetilde{G}_{t+1}$ constructed in Algorithm 1.

For each outer vertex $v \in V^{\text{out}}$, we keep three attributes $p^t(v)$, $t^t_{\text{in}}(v)$ and $t^t_{\text{out}}(v)$, and we call them parent, in-level and out-level respectively. We calculate these values from line 2 to 7 and line 25 to 27. $p^t(v)$ is a vertex belonging to the circle graph part of $\widetilde{G}_t$, namely $p^t(v) \in U_t$. From the algorithm, we can show that there are token tours from $v$ to $p^t(v)$ and/or from $p^t(v)$ to $v$. For the token tour from $v$ to $p^t(v)$, the final level of the token becomes $t^t_{\text{in}}(v)$. On the other hand, for the token tour from $p^t(v)$, it is enough to have $t^t_{\text{out}}(v)$ as an initial level to reach $v$. We will show these facts implicitly in the proof of Lemma 5.

At the beginning of each iteration of Algorithm 1, we choose a lowest gap-$2^+$ chord $e_s^*$. We collect vertices in $U_t$ which are endpoints for some edges crossing with $e_s^*$, and we refer to the vertices among which they are in the lower area of $e_s^*$ as $S^t$ and the vertices in the upper area of $e_s^*$ as $S^u$ (see Figure 6(a) and line 2). Next we collect vertices whose parents are in $S^t$ (resp., $S^u$), and we denote them by $T^t$ (resp., $T^u$) (line 3). Let $x'$ and $y'$ be vertices whose parents are $t(e_s^*)$ and $h(e_s^*)$ respectively. We assign indices to the vertices in $T^u$ and $T^t$ such that the nearer to $x'$ a vertex is located, the larger index the vertex has (see Figure 6(b)). We regard $T^t$ as a sequence $(t^t_1, t^t_2, \ldots, t^t_{|T^t|})$, and $T^u$ as a sequence $(t^u_1, t^u_2, \ldots, t^u_{|T^u|})$. For each vertex $t^t_i$ in $T^t$, we calculate $t^t_{\text{in}}(t^t_i)$ and $t^t_{\text{out}}(t^t_i)$ according to reachability among vertices in $T^t$ and $T^u$ in $G^\text{cir}_0$. When $t^u_j$ has the maximum index among vertices that $t^t_i$ can reach in $T^u$, we let $t^t_{\text{in}}(t^t_i) = j + i/n$. When $t^u_j$ has the minimum index among vertices which can reach $t^t_i$ in $T^u$, we let $t^t_{\text{out}}(t^t_i) = j + i/n$. The term $i/n$ is for breaking ties. See Figure 7 $T^t = \{t^t_1, t^t_2, t^t_3\}$, $T^u = \{t^u_1, t^u_2, t^u_3\}$ and the edges are derived from $\mathcal{E}_{0}^\text{cir}$. The vertex $t^u_3$ can reach $t^t_1$, $t^t_2$ and $t^t_3$. Thus $t^t_{\text{in}}(t^t_3) = \max(1, 2, 3) + 3/n = 3 + 3/n$. The vertices $t^t_2$ and $t^t_3$ can reach $t^t_2$. Thus $t^t_{\text{out}}(t^t_2) = \min(2, 3) + 2/n = 2 + 2/n$. In the next for-loop, we change the in- and out-levels so that the in-level of the larger indexed vertex is larger.

Figure 5: Examples of vertices made by MakePlanar.
Algorithm 1

Input: A circle graph $G_0^{cir} = (V_0^{cir}, E_0^{cir})$ obtained from a block graph.

Task: Output a plane gadget graph $G_p = (V_p^{out} \cup \bar{V}_p^{in}, E_p, K_p, \bar{L}_p)$ which satisfies $V_p^{out} = V_0^{cir}$ and the reachability among vertices in $V_p^{out}$ in $G_p$ is the same as $G_0^{cir}$.

1. initialize $t = 0 //$ loop counter
2. $G_0 = (\bar{V}_0^{out} \cup \bar{V}_0, E_0, K_0, \bar{L}_0)$ where $\bar{V}_0^{out} \leftarrow V_0^{cir}, \bar{V}_0 \leftarrow \emptyset, E_0 \leftarrow E_0^{cir}, K_0(e) \leftarrow \bot, \bar{L}_0(e) \leftarrow \{0 \rightarrow \infty\}$ for each $e \in E_0^{cir}$, and $U_0 \leftarrow \bar{V}_0^{out}$
3. for every $v \in \bar{V}_0^{out}$, $\ell^0_{in}(v) \leftarrow 0, \ell^0_{out}(v) \leftarrow \infty, p^0(v) \leftarrow v$
4. while $G_t[U_t]$ has a lowest gap-2$^+$ chord do
5. pick a lowest gap-2$^+$ chord $e_t^*$
6. make a new vertex $v_t^*$
7. $\bar{V}_{t+1} \leftarrow \bar{V}_t \cup \{v_t^*\}$
8. $\bar{E}_{t+1} \leftarrow (\bar{E}_t \cup \{(t(e), v_t^*) \mid e \text{ crosses } e_* \text{ or } e = e_*\}) \cup \{e \mid e \text{ crosses } e_*\}$
9. $U_{t+1} \leftarrow (U_t \cup \{v_t^*\}) \cup G_t[U_t](t(e_t^*), h(e_t^*))$
10. use MakePlanar to make the lower area of $e_t^*$ planar and update $\bar{V}_{t+1}, \bar{E}_{t+1}$ and $\bar{K}_{t+1}$
11. change the labels by using Algorithm 2 for keeping reachability
12. output $G_{t+1}[C_{G_t[U_t]}(t(e_t^*), h(e_t^*)) \cup \{v_t^*\} \cup V_{MP}^{out}]$, which is the lower area of $e_t^*$
13. $t \leftarrow t + 1$
14. end while
15. use MakePlanar to make $G_t[U_t]$ planar and assign labels by line 17-24 of Algorithm 2
16. output $G_t[U_t \cup V_{MP}^{out}]$

than the out-level of the smaller indexed vertex. If there exists a vertex $t^e_j$ such that $i > j$ and $\ell^{t^e_j+1}_{in}(t^e_j) > \ell^{t^e_i+1}_{in}(t^e_i)$, then we let $\Delta = (\ell^{t^e_j+1}_{in}(t^e_j) - j/n) - (\ell^{t^e_i+1}_{in}(t^e_i) - i/n)$ and add $\Delta$ to $\ell^{t^e_j+1}_{out}(t^e_j)$ and $\ell^{t^e_i+1}_{out}(t^e_i)$. For preserving the magnitude relationship between in- and out-levels of $t^e_i$ and those of $t^e_k$ ($k > i$), we also add $\Delta$ to $\ell^{t^e_k+1}_{in}(t^e_k)$ and $\ell^{t^e_k+1}_{out}(t^e_k)$. In Figure 7, we have $\ell^{t^e_2+1}_{in}(t^e_2) < \ell^{t^e_1+1}_{out}(t^e_1)$. Thus we add 1 = $\ell^{t^e_2+1}_{out}(t^e_2) - 1/n - (\ell^{t^e_1+1}_{in}(t^e_2) - 2/n)$ to $\ell^{t^e_1+1}_{in}(t^e_2)$. Moreover, we add 1 to $\ell^{t^e_2+1}_{out}(t^e_2), \ell^{t^e_2+1}_{in}(t^e_2)$ and $\ell^{t^e_2+1}_{out}(t^e_2)$ so that we keep the magnitude relationship. Here we state a lemma.

**Lemma 4.** For any $t$, if $i > j$, then $\ell^{t^e_i+1}_{out}(t^e_i) > \ell^{t^e_j+1}_{out}(t^e_j)$.

Back to Algorithm 2. From line 8 to 16, we assign labels to edges newly appearing in $G_{t+1}$. Figure 8 is an example of how to assign label sets based on Figure 7. The vertex $a$ is the parent of $t^a_3$, $b$ is the parent of $t^a_1$ and $t^a_2$, $c$ is the parent of $t^a_2$ and $t^a_3$ and $d$ is the parent of $t^a_1$. Let $v$ be any vertex in $T^e$. For edges in the lower area of $e_t^*$, the edge $(p^e(v), v_t^*)$ has a label $\ell^0_{in}(v) \rightarrow \ell^0_{out}(v)$ (line 9), and the edge $(v_t^*, p^e(v))$ has a label $\ell^0_{out}(v) \rightarrow \ell^0_{out}(v)$ (line 10). In Figure 8, the edge $(c, v_t^*)$ has labels $\ell^0_{in}(t^a_2) \rightarrow \ell^{t^e_2+1}_{in}(t^a_2)$ and $\ell^{t^e_2+1}_{in}(t^a_2) \rightarrow \ell^{t^e_2+1}_{in}(t^a_3)$. The edge $(v_t^*, c)$ has a label $\ell^{t^e_2+1}_{out}(t^a_2) \rightarrow \ell^{t^e_2+1}_{out}(t^a_3)$ and the edge $(v_t^*, d)$ has a label $\ell^{t^e_2+1}_{out}(t^a_3) \rightarrow \ell^{t^e_2+1}_{out}(t^a_3)$. Consider edges in the upper area of $e_t^*$. Let $v$ be any vertex in $W^e$. The edge $(p^e(v), v_t^*)$ has a label $\ell^0_{in}(v) \rightarrow \ell^{t^e_{max}}_{max}$ where $\ell^{t^e_{max}}$ is the maximum in-level of vertices in $T^e$ that can reach $v$ (line 13). The edge $(v_t^*, p^e(v))$ has a label $\ell^{t^e_{min}}_{out}(v)$ where $\ell^{t^e_{min}}$ is the minimum out-level of vertices in $T^e$ that $v$ can reach (line 14). In Figure 8, the edge $(a, v_t^*)$ has a label $\ell^{t^e_{min}}_{in}(t^a_3) \rightarrow \ell^{t^e_{out}+1}_{out}(t^a_2)$ since $t^a_3$ can reach $t^a_1$ and $t^a_2$, and $\ell^{t^e_{out}+1}(t^a_1) < \ell^{t^e_{out}+1}(t^a_2)$ (see Figure 7). The edge $(v_t^*, b)$ has a label $\ell^{t^e_{in}+1}_{in}(t^a_2) \rightarrow \ell^{t^e_{out}+1}(t^a_3)$ since $t^a_2$ and $t^a_3$ can reach $t^a_1$, and $\ell^{t^e_{out}+1}(t^a_3) < \ell^{t^e_{in}+1}_{in}(t^a_3)$ (see Figure 7). The edges $(\ell(e_t^*, v_t^*)$ and $(v_t^*, h(e_t^*))$ have only one label $\infty \rightarrow 0$, which prohibits using these edges (line 16).

From line 17 to 24, we assign labels to edges made by MakePlanar. For every edge $(u, v)$ in the lower area of $e_t^*$, the edge $(u, v)$ might be divided into some edges, for instance $(u, w_1), (w_1, w_2), \ldots (w_k, v)$.
Algorithm 2

Task: Set $L_{t+1}$ so that $G_{t+1}$ has the same reachability as $G_t$

1: For every edge $e$ appearing in both $G_t$ and $G_{t+1}$, let $L_{t+1}(e) = \bar{L}_t(e)$.
2: $S^t$ (resp., $S^u$) $\leftarrow \{v \in U_t \mid \exists e \in \bar{E}_t \text{ s.t. } e \text{ crosses } e^t, t(e) = v \text{ or } h(e) = v\}$, and $v$ is at the lower (resp., upper) area of $e^t$.
3: $T^t$ (resp., $T^u$) $\leftarrow \{v \in V^t_0 \mid p^t(v) \in S^t \text{ (resp. } S^u)\}$
4: Fix any vertices $x^t, y^t \in V^t_0$ such that $p^t(x^t) = t(x^t)$, $p^t(y^t) = h(x^t)$.
5: Set an order to $T^t$ according to the order appearing in $C^v_{G^u_0}[y^t, x^t]$. We regard $T^t$ as a sequence $(t^t_1, t^t_2, \ldots, t^t_{|T^t|})$ (see Figure 6(b)).
6: Set an order to $T^u$ in the same way as $T^t$ but according to the tour along the other arc. We also regard $T^u$ as a sequence $(t^u_1, t^u_2, \ldots, t^u_{|T^u|})$ (see Figure 6(b)).
7: Use Algorithm 3 for calculating $\ell^t_{in}(v)$ and $\ell^t_{out}(v)$ for all $v \in T^t$.
8: for $u \in S^t$ do
9: $\{\ell^t_{in}(v), v^t_{in}\} \leftarrow \{\ell^t_{in}(v) \to \ell^t_{in}(v) \mid p^t(v) = u\}$
10: $\{\ell^t_{out}(v), u\} \leftarrow \{\ell^t_{out}(v) \to \ell^t_{out}(v) \mid p^t(u) = v\}$
11: end for
12: for $u \in S^u$ do
13: $\{\ell^t_{in}(t^u_i), v^t_{in}\} \leftarrow \{\ell^t_{in}(t^u_i) \to \max_{t^u_j \in T^u} \{\ell^t_{out}(t^u_j) \mid (t^u_i, t^u_j) \in E^0_0 \} \mid t^u_i \in T^u \text{ and } p^t(t^u_i) = u\}$
14: $\{\ell^t_{out}(v^t_i), u\} \leftarrow \min_{t^u_j \in T^u} \{\ell^t_{in}(t^u_j) \mid (t^u_j, t^u_i) \in E^0_0 \} \to \ell^t_{out}(t^u_j) \mid t^u_i \in T^u \text{ and } p^t(t^u_i) = u\}$
15: end for
16: $\{\ell^t_{in}(t(e^t_i)), v^t_{in}\} \leftarrow \{\infty \to 0\}, \{\ell^t_{in}(t(e^t_i), h(e^t_i)) \leftarrow \{\infty \to 0\}$
17: for all edge $e$ created by MakePlanar do
18: Let $e'$ be the original edge of $e$
19: if $t(e) = t(e')$ then
20: $\{\ell^t_{in}(e), a \to b \mid a \to b \in L_t(e')\}$
21: else
22: $\{\ell^t_{in}(e), b \to a \mid a \to b \in \bar{L}_t(e')\}$
23: end if
24: end for
25: for $v \in \{u \in V^t \mid \exists w \in U_t \text{ s.t. } w \text{ is at the lower area of } e^t \text{ and } p^t(u) = w\}$ do
26: $p^t+1(v) = v^t_i$
27: end for
28: Unchanged $\ell^t_{in}(\cdot), \ell^t_{out}(\cdot)$ and $p^t(\cdot)$ will be taken over to $\ell^{t+1}_{in}(\cdot), \ell^{t+1}_{out}(\cdot)$ and $p^{t+1}(\cdot)$.

Algorithm 3

Task: Calculate $\ell^{t+1}_{in}(v)$ and $\ell^{t+1}_{out}(v)$ for all $v \in T^t$.

1: for $i \in [1, |T^t|]$ do
2: $\ell^{t+1}_{in}(t^t_i) \leftarrow \max\{j \mid (t^t_i, t^t_j) \in E^0_0, t^t_j \in T^u\} + i/n$
3: $\ell^{t+1}_{out}(t^t_i) \leftarrow \min\{j \mid (t^t_i, t^t_j) \in E^0_0, t^t_j \in T^u\} + i/n$
4: end for
5: for $i = 1$ to $|T^t|$ do
6: $\Delta \leftarrow \max(0, \max\{\ell^{t+1}_{out}(t^t_j) - j/n \mid 1 \leq j < i\}) - (\ell^{t+1}_{in}(t^t_i) - i/n)$
7: for $k \in [i, |T^t|]$ do
8: $\ell^{t+1}_{in}(t^t_k) \leftarrow \ell^{t+1}_{in}(t^t_k) + \Delta$
9: $\ell^{t+1}_{out}(t^t_k) \leftarrow \ell^{t+1}_{out}(t^t_k) + \Delta$
10: end for
11: end for
by MakePlanar. In this case, when \((u, v)\) has a label \(a \rightarrow b\), \((u, w_1)\) has a label \(a \rightarrow b\) and the other edges have labels \(b \rightarrow b\) (see Figure 9).

From line 25 to 27, we update the parents of the vertices whose parents are in the lower area of \(e_t^l\). For each vertex \(v\) in \(V_{0ir}^t\) which \(p^t(v)\) is in the lower area of \(e_t^l\), we let \(p^{t+1}(v) = v_t^s\).

For a gadget graph \(G = (\bar{V}, \bar{E}, \bar{K}, \bar{L})\), we use \((v_1, \ell_1) \Rightarrow (v_2, \ell_2) \Rightarrow \cdots \Rightarrow (v_m, \ell_m)\) to denote a token tour from \(v_1\) to \(v_m\) in \(G\) with having a level \(\ell_i\) at \(v_i \in \bar{V}\) for any \(1 \leq i \leq m\). Needless to say, if such a tour exists, \((v_i, v_{i+1}) \in \bar{E}\) and \(\ell_i \rightarrow \ell_{i+1} \in \bar{L}(v_i, v_{i+1})\) where \(\ell_i \geq \ell_i^l\) for any \(1 \leq i < m\). In addition, when we would like to show which available labels we used, we write, for example, \((v_i, \ell_i; \ell_i^l \rightarrow \ell_{i+1}^l) \Rightarrow (v_{i+1}, \ell_{i+1}^l)\), which means the available label \(\ell_i^l \rightarrow \ell_{i+1}^l\) was used. The following lemma shows that paths in \(\bar{G}_0\) remain in \(\bar{G}_t\) for every \(t\).

**Lemma 5.** For any \(t\) in Algorithm 4 if there exists an edge from \(x\) toward \(y\) in \(\bar{G}_0\), then there exists a token tour from \(x\) to \(y\) in \(\bar{G}_t\) whose length is at most \(2t + 1\).

**Proof.** We prove the lemma by showing that if \((x, y) \in \bar{E}_0\) then one of the following two statements holds in \(\bar{G}_t\) for any \(t\):

(i) there exists a token tour of length at most \(2t + 1\) from \(x\) to \(y\) which uses no chords appearing in \(\bar{G}_t[U_t]\) (see Figure 10(i)).

(ii) there exists a token tour \(t_{x,y} = (x, \infty) \Rightarrow \cdots \Rightarrow (p^t(x), \ell_{in}^t(x); \ell_{in}^t(x) \rightarrow \ell_{out}^t(y)^+) \Rightarrow (p^t(y), \ell_{in}^t(y)^+; \ell_{in}^t(y)^- \rightarrow \ell) \Rightarrow \cdots \Rightarrow (y, \infty)\) where \(\ell_{in}^t(x)^- \leq \ell_{in}^t(x), \ell_{in}^t(y)^- \leq \ell_{out}^t(y)^+ \leq \ell_{out}^t(y)^- \leq \ell_{out}^t(y)^+\). In addition, this tour uses no chords appearing in \(\bar{G}_t[U_t]\) except \((p^t(x), p^t(y))\), and its length is at most \(2t + 1\) (see Figure 10(ii)).

We prove by induction on \(t\). We have a tour \((x, \infty; 0 \rightarrow \infty) \Rightarrow (y, \infty)\) in \(\bar{G}_0\). Thus \(\bar{G}_0\) satisfies the statement (i) if \(x\) and \(y\) are consecutive on the cycle, and otherwise satisfies the statement (ii).

Assume that the statement (i) holds in \(\bar{G}_t\). The tour from \(x\) to \(y\) appears also in \(\bar{G}_{t+1}\) and satisfies the statement (i) in \(\bar{G}_{t+1}\). Now, we suppose the statement (ii) holds in \(\bar{G}_t\). We first consider the case that the chord \((p^t(x), p^t(y))\) does not cross \(e_+^l\). When \((p^t(x), p^t(y))\) is in the lower area of \(e_+^l\), the tour \(t_{x,y}\) satisfies statement (i) in \(\bar{G}_{t+1}\). When \((p^t(x), p^t(y))\) is in the upper area of \(e_+^l\) or equal to \(e_+^l\), the tour \(t_{x,y}\) satisfies statement (ii) in \(\bar{G}_{t+1}\). Next, we assume that the chord \((p^t(x), p^t(y))\) crosses \(e_+^l\). There are two cases:
Figure 7: How to calculate in and out levels.

| $\ell_{in}^{t+1}$ | $\ell_{out}^{t+1}$ |
|-------------------|-------------------|
| $3 + 3/n$         | $2 + 2/n$         |
| $1 + 2/n$         | $2 + 1/n$         |
| $-$               | $-$               |
| $4 + 3/n$         | $2 + 2/n$         |
| $-$               | $3 + 2/n$         |
| $-$               | $2 + 1/n$         |

Figure 8: How to assign labels to edges.

a = $p'(t_3^n)$
b = $p'(t_2^n) = p'(t_1^n)$
c = $p'(t_3) = p'(t_2)$
d = $p'(t_1)$

Figure 9: How to assign labels to edges made by MakePlanar.
Lemma 6. For any $t$ and $x, y \in V_0^{\text{cir}}$, if there exists a token tour from $x$ to $y$ in $\hat{G}_t$, then there exists a traversable edge sequence $(e_1, \ldots, e_k)$ in $G_0^{\text{cir}}$ such that $t(e_1) = x$ and $h(e_k) = y$.

Before proving this, we prepare several lemmas, i.e., Lemma 7 to Lemma 11.

Figure 10: Two cases of token tours in the proof of Lemma 5.

(I) $p'(x) \in S^t$ and $p'(y) \in S^u$: We have $x \in T^t$ and $y \in T^u$. There exists a label $\ell_{in}^t(x) \rightarrow \ell_{in}^{t+1}(x) \in \tilde{L}^{t+1}(p'(x), v_s^t)$ (cf. line 9 of Algorithm 2). There also exists a label $\ell_{min} \rightarrow \ell_{out}^t(y) \in \tilde{L}^{t+1}(v_s^t, p'(y))$ where $\ell_{min} = \min_{t \in T^r} \{ \ell_{in}^{t+1}(t^t) \mid (t^t, y) \in E_0^{\text{cir}} \}$ (cf. line 14 of Algorithm 2). Since $x \in T^t$ and $(x, y) \in E_0^{\text{cir}}$, we have $\ell_{min} \leq \ell_{in}^{t+1}(x)$. Thus, in $\hat{G}_{t+1}$, there exists a token tour $(x, \infty) \Rightarrow \cdots \Rightarrow (p'(x), \ell_{in}^t(x); \ell_{in}^{t+1}(x) \rightarrow \ell_{in}^{t+1}(x)) \Rightarrow (v_s^t, \ell_{in}^{t+1}(x); \ell_{min} \rightarrow \ell_{out}^t(y)) \Rightarrow (p'(y), \ell_{out}^t(y)) \Rightarrow \cdots \Rightarrow (y, \infty)$. If the edge $(p'(x), v_s^t)$ has a crossing point in the lower area of $e_s^t$, we have to modify the part $(p'(x), \ell_{in}^t(x); \ell_{in}^{t+1}(x) \rightarrow \ell_{in}^{t+1}(x)) \Rightarrow (v_s^t, \ell_{in}^{t+1}(x))$ to $(p'(x), \ell_{in}^t(x); \ell_{in}^{t+1}(x) \rightarrow \ell_{in}^{t+1}(x)) \Rightarrow (u, \ell_{in}^{t+1}(x); \ell_{in}^{t+1}(x) \rightarrow \ell_{in}^{t+1}(x)) \Rightarrow (v_s^t, \ell_{in}^{t+1}(x))$ where $u$ is a vertex created by MakePlanar.

(II) $p'(x) \in S^u$ and $p'(y) \in S^t$: We have $x \in T^u$ and $y \in T^t$. There exists a label $\ell_{in}^t(x) \rightarrow \ell_{max} \in \tilde{L}^{t+1}(p'(x), v_s^t)$ where $\ell_{max} = \max_{t \in T^r} \{ \ell_{out}^{t+1}(t^t) \mid (x, t^t) \in E_0^{\text{cir}} \}$ (cf. line 13 of Algorithm 2). Since $y \in T^t$ and $(x, y) \in E_0^{\text{cir}}$, we have $\ell_{max} \geq \ell_{out}^{t+1}(y)$. There also exists a label $\ell_{out}^{t+1}(y) \rightarrow \ell_{out}^t(y) \in \tilde{L}^{t+1}(v_s^t, p'(y))$ (cf. line 10 of Algorithm 2). Thus, in $\hat{G}_{t+1}$, there exists a token tour $(x, \infty) \Rightarrow \cdots \Rightarrow (p'(x), \ell_{in}^t(x); \ell_{in}^{t+1}(x) \rightarrow \ell_{in}^{t+1}(x)) \Rightarrow (v_s^t, \ell_{max}; \ell_{out}^{t+1}(y) \rightarrow \ell_{out}^t(y)) \Rightarrow (p'(y), \ell_{out}^t(y)) \Rightarrow \cdots \Rightarrow (y, \infty)$ in $\hat{G}_{t+1}$. If the edge $(v_s^t, p'(y))$ has a crossing point in the lower area of $e_s^t$, we have to modify the part $(v_s^t, \ell_{max}; \ell_{out}^{t+1}(y) \rightarrow \ell_{out}^t(y)) \Rightarrow (p'(y), \ell_{out}^t(y))$ to $(v_s^t, \ell_{max}; \ell_{out}^{t+1}(y) \rightarrow \ell_{out}^t(y)) \Rightarrow (u, \ell_{out}^t(y); \ell_{out}^{t+1}(y) \rightarrow \ell_{out}^t(y)) \Rightarrow (p'(y), \ell_{out}^t(y))$ where $u$ is a vertex created by MakePlanar.

In both cases, the length of the new tour is longer than that of $t_{x,y}$ by at most 2, thus it is at most $2(t + 1) + 1$. We have $p^{t+1}(x) = v_s^t$ in case (I) and $p^{t+1}(y) = v_s^t$ in case (II). Thus the new tour has only one chord $(p^{t+1}(x), p^{t+1}(y))$ appearing in $\hat{G}_{t+1}[U_{t+1}]$, and the chord has a label $\ell_{in}^t(x) \rightarrow \ell_{out}^t(y)$. Therefore the new tour satisfies statement (ii).

The following lemma shows the other direction: if there exists a token tour from $x$ to $y$ in the gadget graph, then there exists a path from $x$ to $y$ in the circle graph. From Lemma 2 it is enough to prove the following Lemma.
Lemma 8. In Algorithm 3 in step t, for \( t'_p, t'_q \in T^\ell \), if \( t'^{\ell+1}_{in}(t'_p) \geq t'^{\ell+1}_{out}(t'_q) \), then \( t'^{\ell+1}_{in}(t'_p) \geq t'^{\ell+1}_{out}(t'_q) \).

Proof. When \( p > q \), this lemma holds from Lemma 7. Consider the case \( p \leq q \). Let \( t'^{\ell+1}_{in}(t'_p) = i + p/n \) and \( t'^{\ell+1}_{out}(t'_q) = j + q/n \). Since \( t'^{\ell+1}_{in}(t'_p) \geq t'^{\ell+1}_{out}(t'_q) \) and \( p \leq q \), we have \( i \geq j \). Assume \( t'^{\ell+1}_{in}(t'_p) < t'^{\ell+1}_{out}(t'_q) \). In order that \( t'^{\ell+1}_{in}(t'_p) < t'^{\ell+1}_{out}(t'_q) \) holds, some positive integer \( \Delta \) should be added to \( t'^{\ell+1}_{in}(t'_p) \) at line 9, and not added to \( t'^{\ell+1}_{out}(t'_q) \) at line 8 of Algorithm 3. Thus, there should be a vertex \( t'_s \) such that \( q \geq r \geq p \) and \( \Delta = \max\{ t'^{\ell+1}_{out}(t'_k) - k/n \mid 1 \leq k < r \} - (t'^{\ell+1}_{in}(t'_s) - r/n) > 0 \). Since \( \Delta \) is positive, there exists a vertex \( t'_s \) such that \( r > s \) and \( t'^{\ell+1}_{in}(t'_s) < t'^{\ell+1}_{out}(t'_s) \). Since
\( r \geq p \) and \( q \geq s \), we have \( t^k_{in}(t_p^k) \geq t^{k+1}_{in}(t_p) \) and \( t^k_{out}(t_q^s) \geq t^{k+1}_{out}(t_q) \) from Lemma \[7\]. Now \( t^{k+1}_{in}(t_p) \geq t^{k+1}_{out}(t_q) \) holds, thus \( t^{k+1}_{in}(t_p) \geq t^{k+1}_{out}(t_q) \) and \( \Delta \) becomes non-positive. This is a contradiction. Thus \( t^k_{in}(t_p) \geq t^{k+1}_{out}(t_q) \) holds.

For every label in \( G_t \) and \( k \leq t \), there are three types: (i) \( \ell^k_{in}(x) \to \ell^{k+1}_{in}(x) \) (cf. line 9), (ii) \( t^k_{out}(x) \to t^{k+1}_{out}(x) \) (cf. line 10) and (iii) \( t^k_{in}(x) \to t^{k}_{out}(y) \) (cf. line 13, 14) for \( x, y \in V_{0}^{\text{cir}} \). We define a source vertex and a sink vertex for any types of labels.

(i) source vertex is \( x \). When \( i = \max\{j \mid (x, t^k_j) \in E_{0}^{\text{cir}}, t^k_j \in T^u\} \), sink vertex is \( t^k_i \).

(ii) sink vertex is \( x \). When \( i = \min\{j \mid (t^k_y, t^k_i \in E_{0}^{\text{cir}}, t^k_j \in T^u\} \), source vertex is \( t^k_i \).

(iii) source vertex is \( x \) and sink vertex is \( y \).

For a label \( L \), we refer to an edge in \( G_{0}^{\text{cir}} \) from \( L \)'s source vertex to \( L \)'s sink vertex as a source edge of \( L \). It is obvious that any source edge exists in \( G_{0}^{\text{cir}} \).

Lemma 9. We consider any token tour of length 2 going through \( v^k_x \): \( (x, a'; a \to b) \Rightarrow (v^k_x, b; c \to d) \Rightarrow (y, d) \) in \( G_t \) where \( t < t' \).

1. \( (x, v^k_i) \) is in upper area, and \( (v^k_i, y) \) is in upper area of \( e^k \). Let \( (t^k_q, t^k_p) \) be \( a \to b \)'s source edge, and we let \((t^k_q, t^k_p)\) be \( c \to d \)'s source edge. We have \( p < q \).

2. \( (x, v^k_i) \) is in upper area, and \( (v^k_i, y) \) is in lower area of \( e^k \). Let \( (t^k_q, t^k_p) \) be \( a \to b \)'s source edge, and we let \((t^k_q, t^k_p)\) be \( c \to d \)'s source edge. We have \( p < q \).

3. \( (x, v^k_i) \) is in lower area, and \( (v^k_i, y) \) is in upper area of \( e^k \). Let \( (t^k_p, t^k_q) \) be \( a \to b \)'s source edge, and we let \((t^k_q, t^k_p)\) be \( c \to d \)'s source edge. We have \( p < q \).

4. \( (x, v^k_i) \) is in lower area, and \( (v^k_i, y) \) is in lower area of \( e^k \). Let \( (t^k_p, t^k_q) \) be \( a \to b \)'s source edge, and we let \((t^k_q, t^k_p)\) be \( c \to d \)'s source edge. We have (i) \( i < j \) and \( p < q \), (ii) \( i < j \) and \( p < q \), or (iii) \( i < j \) and \( p < q \).

The indices \( i, j, p \) and \( q \) are based on the sequences \( T^u \) and \( T^k \) made in Algorithm \[2\] in step \( t \).

Proof.

1. We have \( b = \ell^{k+1}_{out}(t^k_p) \) and \( c = \ell^{k+1}_{in}(t^k_q) \). From the rule of token tours, \( \ell^{k+1}_{in}(t^k_p) \geq \ell^{k+1}_{in}(t^k_q) \) holds. If \( p < q \), we have \( \ell^{k+1}_{out}(t^k_q) < \ell^{k+1}_{out}(t^k_p) \) from Lemma \[4\]. Thus we have \( p \geq q \).

2. We have \( b = \ell^{k+1}_{out}(t^k_p) \) and \( c = \ell^{k+1}_{in}(t^k_q) \). From the rule of token tours, \( \ell^{k+1}_{in}(t^k_p) \geq \ell^{k+1}_{out}(t^k_q) \) holds. If \( p < q \), we have \( \ell^{k+1}_{out}(t^k_q) < \ell^{k+1}_{out}(t^k_p) \) from Lemma \[7\]. Thus we have \( p \geq q \).

3. We have \( b = \ell^{k+1}_{in}(t^k_p) \) and \( c = \ell^{k+1}_{in}(t^k_q) \). From the rule of token tours, \( \ell^{k+1}_{in}(t^k_p) \geq \ell^{k+1}_{in}(t^k_q) \) holds. If \( p < q \), we have \( \ell^{k+1}_{in}(t^k_q) < \ell^{k+1}_{in}(t^k_p) \) from Lemma \[7\]. Thus we have \( p \geq q \). From the definition of source edge, \( t^k_i \) has the maximum index among vertices that \( t^k_p \) can reach. Assume \( i < j \). Now we have \( p \geq q \) and \( i < j \). Thus the edges \((t^k_p, t^k_i)\) and \((t^k_q, t^k_j)\) are semi-crossing. From Lemma \[2\], the edge \((t^k_p, t^k_i)\) is in \( E_{0}^{\text{cir}} \). This is contrary to the fact that \( i \) is the maximum index. Thus \( i \geq j \) holds.

4. We have \( b = \ell^{k+1}_{in}(t^k_p) \) and \( c = \ell^{k+1}_{out}(t^k_q) \). From the rule of token tours, \( \ell^{k+1}_{in}(t^k_p) \geq \ell^{k+1}_{out}(t^k_q) \) holds. We will show that \( i < j \) and \( p \geq q \) do not hold simultaneously. Assume \( i < j \) and \( p < q \). We have \( i + p/n = \ell^{k+1}_{in}(t^k_j) = \ell^{k+1}_{out}(t^k_i) = j + q/n \). From Lemma \[7\], we have \( \ell^{k+1}_{in}(t^k_p) < \ell^{k+1}_{out}(t^k_q) \) since \( p < q \), and we cannot follow the tour. Thus, there are three possible relationships: (i) \( i \geq j \) and \( p \geq q \), (ii) \( i \geq j \) and \( p < q \), (iii) \( i < j \) and \( p \geq q \).
Lemma 10. For any three vertices \(u, v, w \in U_t\), if \((u, v, w)\) is in clockwise (resp., anti-clockwise) order in \(\tilde{G}_t[U_t]\), then \((x, y, z)\) is also in clockwise (resp., anti-clockwise) order in \(G_0^{\text{circ}}\) for any \(x, y, z \in V_0^{\text{circ}}\) such that \(p^t(x) = u, p^t(y) = v\) and \(p^t(z) = w\) (see Figure 12(a)).

Proof. We prove by induction on \(t\). Since the parent of every vertex is itself in step 0, the Lemma is true in step 0. Let \(u, v\) and \(w\) be vertices such that \((u, v, w)\) is in clockwise (resp., anti-clockwise) order in \(\tilde{G}_t[U_t]\). Fix any three vertices \(x, y, z\) such that \(p^{t+1}(x) = u, p^{t+1}(y) = v\) and \(p^{t+1}(z) = w\). When none of \(u, v\) or \(w\) is \(v_t^e\), \((u, v, w)\) is in clockwise (resp., anti-clockwise) order also in \(\tilde{G}_t[U_t]\). Thus \((x, y, z)\) is in clockwise (resp., anti-clockwise) order from the induction hypothesis. Let \(u = v_t^e\). Since \(p^t(x)\) is in the lower area of \(e_t^e\), \((p^t(x), u, v)\) is in clockwise (resp., anti-clockwise) order in \(\tilde{G}_t[U_t]\) (see Figure 12(b)). From the induction hypothesis, \((x, y, z)\) is in clockwise (resp., anti-clockwise) order in \(G_0^{\text{circ}}\). In cases \(v = v_t^e\) or \(w = v_t^e\), the Lemma is proved in the same way.

Lemma 11. For any \(k \leq t\) such that \(v_t^k \in U_t\), let \(x\) and \(y\) be vertices such that \(p^k(x) = t(e_t^e)\) and \(p^k(y) = h(e_t^e)\) respectively. \(p^t(x), v_t^k\) and \(p^t(y)\) are consecutive in \(\tilde{G}_t[U_t]\).

Proof. Fix any \(k\). We prove by induction on \(t\). When \(t = k\), it is obvious that \(p^t(x), v_t^k\) and \(p^t(y)\) are consecutive. Assume the Lemma is true for a fixed \(t\). If \(v_t^k\) is not an endpoint of \(e_t^e\), we have \(p^t(x) = p^{t+1}(x)\) and \(p^t(y) = p^{t+1}(y)\). Thus \(p^{t+1}(x), v_t^k\) and \(p^{t+1}(y)\) are consecutive in \(\tilde{G}_{t+1}[U_{t+1}]\) from the induction hypothesis. When \(v_t^k\) is an endpoint of \(e_t^e\) and \(p^t(x)\) (resp., \(p^t(y)\)) is on \(C_{\tilde{G}_{t}[U_t]}(t(e_t^e), h(e_t^e))\), we have \(p^{t+1}(x) = v_t^k\) (resp., \(p^{t+1}(y) = v_t^k\)). Since \(v_t^k\) and \(v_t^k\) are adjacent, \(p^{t+1}(x), v_t^k\) and \(p^{t+1}(y)\) are consecutive in \(\tilde{G}_{t+1}[U_{t+1}]\).

Now we prove Lemma 6 (restated). For any \(t\) and \(x, y \in V_0^{\text{circ}}\), if there exists a token tour from \(x\) to \(y\) in \(\tilde{G}_t\), then there exists a traversable edge sequence \((e_1, \ldots, e_k)\) in \(G_0^{\text{circ}}\) such that \(t(e_1) = x\) and \(h(e_k) = y\).

Proof. Let \(t_{x,y}\) be a token tour \((v_1, \ell_1; f_1 \to \ell_2) \Rightarrow (v_2, \ell_2; f_2 \to \ell_3) \Rightarrow \cdots \Rightarrow (v_m, \ell_m)\) such that \(v_1 = x\) and \(v_m = y\). We modify \(t_{x,y}\). If the edges \((v_i, v_{i+1}), \ldots, (v_{i+d-1}, v_{i+d})\) are made by \texttt{MakePlanar} and they have the same original edge, namely \(K(i, v_{i+1}) = (v_{j+1}, v_{j+2})\) \((i \leq j < i + d - 1)\), we change the partial tour \((v_i, \ell_i; f_i \to \ell_{i+1}) \Rightarrow (v_{i+1}, \ell_{i+1}; f_{i+1} \to \ell_{i+2}) \Rightarrow \cdots \Rightarrow (v_{i+d}, \ell_{i+d})\) to \((v_i, \ell_i; f_i \to \ell_{i+1}) \Rightarrow \cdots \Rightarrow (v_{i+d}, \ell_{i+d})\). Figure 12: Relations of vertices and their parents.
$\ell_{i+1} \Rightarrow (v_{i+d}, \ell_{i+1})$. Note that $\ell_{i+1}$ is equal to $\ell_{i+d}$. Next, we remove redundant moves. Consider a partial tour of length 2 ($v_i, \ell_i; f_i \rightarrow \ell_{i+1}$ $\Rightarrow (v_{i+1}, \ell_{i+1}; f_{i+1} \rightarrow \ell_{i+2})$ $\Rightarrow (v_{i+2}, \ell_{i+2}; f_{i+2} \rightarrow \ell_{i+3})$). When $v_i = v_{i+2}$ and $f_i \geq \ell_{i+2}$, we regard this move as a redundant move. We have $\ell_i \geq f_i$ and $\ell_{i+2} \geq f_{i+2}$. If the move is redundant, $\ell_i \geq f_{i+2}$ holds. We change the partial tour ($v_i, \ell_i; f_i \rightarrow \ell_{i+1}$ $\Rightarrow (v_{i+1}, \ell_{i+1}; f_{i+1} \rightarrow \ell_{i+2})$ $\Rightarrow (v_{i+2}, \ell_{i+2}; f_{i+2} \rightarrow \ell_{i+3})$) to ($v_i, \ell_i; f_{i+2} \rightarrow \ell_{i+3}$) $\Rightarrow (v_{i+3}, \ell_{i+3})$. Again, we let the changed token tour be $t_{x,y} = (v_1, \ell_1; f_1 \rightarrow \ell_2) \Rightarrow (v_2, \ell_2; f_2 \rightarrow \ell_3) \Rightarrow \cdots \Rightarrow (v_m, \ell_m)$.

We construct a traversable edge sequence. We put source edges of the labels appearing in $t_{x,y}$. Let this edge sequence be $(e_1, \ldots, e_{m-1})$ where $e_i$ is a source edge of the label $f_i \rightarrow \ell_{i+1}$. For each $i$ ($2 \leq i < m$), $v_i$ corresponds to $v^k_i$ for some $1 \leq k < t$ since we removed vertices made by MakePlanar from the tour. For every $i$ ($2 \leq i < m$), we take an edge $e'_i \in E^m_0$ such that $p^k(t(e'_i)) = t(e^k_i)$ and $p^k(h(e'_i)) = h(e^k_i)$ where $k$ is derived from $v_i = v^k_i$. There exist several ways to choose $e'_i$. We show that if we select $e'_i$'s appropriately, the edge sequence $(e_1, e'_2, e_3, e_4, \ldots, e_{m-2}, e'_{m-1}, e_{m-1})$ becomes traversable.

It is obvious that $e_1$ crosses $e'_2$. We have to show that all edges except for $e_1$ and $e'_2$ separate two edges appearing before themselves. By induction, suppose we fixed $e'_j$ ($i + 2 \leq j < m$). We show which pair of edges $e_{i+1}$ separates. Consider the partial tour of length 2 ($v_i, \ell_i; f_i \rightarrow \ell_{i+1}$ $\Rightarrow (v_{i+1}, \ell_{i+1}; f_{i+1} \rightarrow \ell_{i+2})$ $\Rightarrow (v_{i+2}, \ell_{i+2})$, and let $v_{i+1} = v^k_{i+1}$. We suppose that this partial tour corresponds to case 1 of Lemma 9, namely the edge $(v_i, v_{i+1})$ is in upper area and the edge $(v_{i+1}, v_{i+2})$ is in upper area of $e_s^k$. We let $(t^u_i, t^l_p)$ be $f_i \rightarrow \ell_{i+1}$’s source edge, and $(t^u_q, t^l_j)$ be $f_{i+1} \rightarrow \ell_{i+2}$’s source edge. From Lemma 9, we have $p \geq q$. Thus $e_{i+1} = (t^l_q, t^u_j)$ separates $e_{i+1}$ and $e_i = (t^l_i, t^u_p)$ for any choice of $e'_i$. In both cases (i) $i \geq j$ and (ii) $i < j$, $t^l_p$ and $h(e_{i+1})$ are on the opposite side of $(t^l_q, t^u_j)$. When the partial tour corresponds to case 2, 3, 4-(i) or 4-(ii), $e_{i+1}$ separates $e'_i$ and $e_i$ for any choice of $e'_i_+$ (see Figure 13).

Figure 13: Relations of two source edges.
this case, we have $\ell_{in}^{k+1}(t') \geq \ell_{out}^{k+1}(t')$ from Lemma \ref{lem:lower_bound}. Assume $e''_s$ and $e''_t$ has the opposite direction, namely $t''_i$, $t''_p$, $t''_q$ and $t''_q$ had indices $i'$, $j'$, $p'$ and $q'$ respectively in step $k$ such that $i' > j'$ and $p' \leq q'$. Since $i' > j'$, $\ell_{in}^{k+1}(t') - p/n \geq i'$ and $\ell_{out}^{k+1}(t') - q/n \leq j'$, we have $t\ell_{in}^{k+1}(t') \geq t\ell_{out}^{k+1}(t')$. From Lemma \ref{lem:lower_bound} $\ell_{in}^{k+1}(t') \geq \ell_{out}^{k+1}(t')$. Thus, when $v_i = v_{i+2}$, this is a redundant move and does not appear in $t_{xy}$.

Again, let $v_i$ and $v_{i+1}$ be $v''_k$ and $v''_s$ respectively. Let $e'$ be an edge such that $p^k(t(e')) = t(e''_s)$ and $p^k(h(e')) = h(e''_s)$. Now we consider the case $v_i \neq v_{i+2}$, hence there are two cases for a location of $p^k(h(e'))$ from Lemma \ref{lem:2_cases}.

1. $p^k(h(e'))$ is on $C_{G\overline{t}[U_i]}[t(e''_s), v_{i+2}]$: Consider the four vertices $p^k(h(e'))$, $v_{i+2}$, $h(e''_s)$ and $p^k(t(e_{i+1}))$. A possible order on $U_i$ of the four vertices is $(p^k(h(e'))$, $v_{i+2}$, $h(e''_s)$, $p^k(t(e_{i+1}))$. From Lemma \ref{lem:ordering} $e_{i+1}$ separates $e'$ and $e'_{i+1}$ for any choice of $e'$ (see Figure \ref{fig:case4}).

2. $p^k(h(e'))$ is equal to $v_{i+2}$: If $e_{i+1}$ separates $e'$ and $e'_{i+1}$, we set $e'$ as $e'_{i}$ (see Figure \ref{fig:case5} (a)). However, $e_{i+1}$ might not separate $e'$ and $e'_{i+1}$. In this case, $e'$ crosses $e_{i+1}$. From Lemma \ref{lem:crossing} there exists an edge $(t(e'), h(e_{i+1}))$ in $E_{0}^{ir}$ (see Figure \ref{fig:case5} (b)). We choose $(t(e'), h(e_{i+1}))$ as $e'_{i}$ and $e_{i+1}$ separates $e'_{i}$ and $e'_{i+1}$ in $G_{0}^{ir}$.

Suppose we fixed $e'_{j}$ ($i + 2 \leq j < m$). We show how to select $e'_{i+1}$ and which pair of edges $e_{i+2}$ separates. Consider any partial token tour of length $2$ $(v_i, \ell_{i}; f_i \rightarrow \ell_{i+1}) \Rightarrow (v_{i+1}, \ell_{i+1}; f_{i+1} \rightarrow \ell_{i+2}) \Rightarrow (v_{i+2}, \ell_{i+2})$. Assume $v_i = v''_s$, $v_{i+1} = v''_k$ and $v_{i+2} = v''_t$. Note that $e_i$ and $e_{i+1}$ are source edges of $f_i \rightarrow \ell_{i+1}$ and $f_{i+1} \rightarrow \ell_{i+2}$ respectively, and $e'_{i+2}$ is an edge such that $p^k(t(e'_{i+2})) = t(e''_s)$ and $p^k(h(e'_{i+2})) = h(e''_s)$. We let $e'$ be a source edge of a label of $e''_s$. When $t > k$:

1. $v''_s$ is on $U_i$: From Lemma \ref{lem:2_cases}, there are three cases for a location of $p^k(h(e'))$.

   (i) $p^k(h(e'))$ is neither $h(e''_s)$ nor $t(e''_s)$: Consider the four vertices $p^k(t(e'_{i+2}))$, $p^k(h(e''_s))$, $p^k(h(e'_{i+2}))$ and $p^k(h(e_{i+1}))$. A possible order on $U_i$ of the four vertices is $(p^k(t(e'_{i+2})), p^k(h(e''_s)), p^k(h(e'_{i+2})), p^k(h(e_{i+1})))$. From Lemma \ref{lem:ordering} $e'_{i+2}$ separates $e_{i+1}$ and $e'$ (see Figure \ref{fig:case4}). We select $e'$ as $e'_{i+1}$.
(ii) \( p'(h(e')) \) is equal to \( h(e'_i) \): When \( e'_{i+2} \) separates \( e_{i+1} \) and \( e' \) (see Figure 18(a)), we select \( e' \) as \( e'_{i+1} \). \( e'_{i+2} \) might not separate \( e_{i+1} \) and \( e' \) (see Figure 18(b)). In this case, \( e' \) crosses \( e'_{i+2} \). From Lemma 2, there exists an edge \((t(e'), h(e'_{i+2}))\) in \( E_0^{cir} \). We choose \((t(e'), h(e'_{i+2}))\) as \( e'_{i+1} \) and \( e'_{i+2} \) separates \( e_{i+1} \) and \( e'_{i+1} \) in \( G_0^{cir} \).

(iii) \( p'(h(e')) \) is equal to \( t(e'_i) \): When \( e'_{i+2} \) separates \( e_{i+1} \) and \( e' \) (see Figure 18(a)), we select \( e' \) as \( e'_{i+1} \). \( e'_{i+2} \) might not separate \( e_{i+1} \) and \( e' \) (see Figure 18(b)). In step \( k \), \( h(e_i) \) and \( h(e'_{i+2}) \) was in \( T^u \). Let \( i \) and \( j \) be indices of \( h(e_i) \) and \( h(e'_{i+2}) \) respectively, that is, \( h(e_i) = t^u_i \) and \( h(e'_{i+2}) = t^u_j \). If \( i < j \), then \( e_i \) crosses \( e'_{i+2} \), and \( t(e_i) \) can reach \( h(e'_{i+2}) \) (see Figure 18(c)). Since \( t^u_i \) has the maximum index among vertices \( t(e_i) \) can reach, this is a contradiction. Therefore we have \( i \geq j \). Thus \( e'_{i+2} \) separates \( e_i \) and \( e' \) (see Figure 18(d)). We select \( e' \) as \( e'_{i+1} \).

When \( k > t \):

1. \((v^*_k, v^*_k)\) and \((v^*_k, v^*_k)\) are in the same side of \( e^*_k \): \( p^k(t(e'_{i+2})) \), \( p^k(h(e_{i+1})) \) and \( p^k(h(e'_{i+2})) \) are consecutive on \( U_k \) from Lemma 11. The parent of \( h(e_i) \) is on the other side of \( e^*_k \). From Lemma 10, \( e'_{i+2} \) separates \( e_{i+1} \) and \( e_i \) (see Figure 20). We select \( e' \) as \( e'_{i+1} \).

2. \((v^*_k, v^*_k)\) and \((v^*_k, v^*_k)\) are in the opposite side of \( e^*_k \): From Lemma 11 there are three cases for a location of \( p^k(h(e'_{i+2})) \) and \( p^k(t(e'_{i+2})) \).
(i) Neither $p^k(h(e_{i+2}'))$ nor $p^k(t(e_{i+2}'))$ is $h(e_k^*)$: Consider the four vertices $p^k(t(e_{i+2}'))$, $p^k(h(e_{i+1}'))$ and $p^k(e')$. A possible order on $U_k$ of the four vertices is $(p^k(t(e_{i+2}'))), p^k(h(e_{i+1}')), p^k(h(e_{i+2}')), p^k(e')$ (see Figure 21). From Lemma 10, $e_{i+2}'$ separates $e_{i+1}$ and $e'$. We select $e'$ as $e_{i+1}'$.

(ii) $p^k(h(e_{i+2}'))$ is equal to $h(e_k^*)$: When $e_{i+2}'$ separates $e_{i+1}$ and $e'$ (see Figure 22(a)), we select $e'$ as $e_{i+1}'$. $e_{i+2}'$ might not separate $e_{i+1}$ and $e'$ (see Figure 22(b)). In this case, $e'$ crosses $e_{i+2}'$. From Lemma 2, there exists an edge $(t(e'), h(e_{i+2}'))$ in $E^0_{cir}$. We choose $(t(e'), h(e_{i+2}'))$ as $e_{i+1}'$, and $e_{i+2}'$ separates $e_{i+1}$ and $e_{i+1}'$ in $G^0_{cir}$.

(iii) $p^k(t(e_{i+2}'))$ is equal to $h(e_{k}^*)$: When $e_{i+2}'$ separates $e_{i+1}$ and $e'$ (see Figure 23(a)), we select $e'$ as $e_{i+1}'$. $e_{i+2}'$ might not separate $e_{i+1}$ and $e'$ (see Figure 23(b)). Assume the edge $(v^*_s, v^*_k)$ is in the upper area of $e_{k}^*$. In step $k$, $h(e_i)$ and $h(e_{i+2}')$ was in $T^k$. Let $p$ and $q$ be indices of $h(e_i)$ and $h(e_{i+2}')$ respectively, that is, $h(e_i) = t_p^k$ and $h(e_{i+2}') = t_q^k$. If $p < q$, then $e_i$ crosses $e_{i+2}'$, and $t(e_i)$ can reach $h(e_{i+2}')$ (see Figure 23(c)). Since $t_p^k$ has the maximum index among vertices $t(e_i)$ can reach, this is a contradiction. Therefore we have $p \geq q$. Thus $e_{i+2}'$ separates $e_i$ and $e'$ (see Figure 23(d)). We select $e'$ as $e_{i+1}'$. In the case $(v^*_s, v^*_k)$ is in the lower area of $e_{k}^*$, we could show that $e_{i+2}'$ separates $e_i$ and $e'$ in
Figure 18: $t > k$: $v_s^*$ is not on $U_t$, $p^t(h(e'))$ is equal to $h(e'_i)$.
Figure 19: $t > k$: $v^*_t$ is not on $U_t$, $p^t(h(e'))$ is equal to $t(e'_t)$. 
Figure 20: $k > t$: $(v^k_*, v^k_*)$ and $(v^k_*, v^t_*)$ are in the same side of $e^k_*$. 

Figure 21: $k > t$: $(v^k_*, v^k_*)$ and $(v^k_*, v^t_*)$ are in the opposite side of $e^k_*$, neither $p^k(h(e^t_*)$ nor $p^k(t(e^t_*)$ is $h(e^k_*)$.

the almost same way.

When we consider which pair of edges $e_{i+1}$ separates, we might choose a specific $e'_i$ (see Figure 15(b)). When we consider which pair of edges $e'_{i+1}$ separates, we also might choose specific $e'_i$ (see Figure 18(b) and 22(b)). If the cases of Figure 15(b) and Figure 18(b) occur simultaneously, $h(e'_i)$ must be $v_{i+2}$, but the edge $(v^k_*, t(e'_i))$ has no available label. In the case of Figure 22(b), the edge $e_i$ is in the upper area of $e'_i$. Thus, these cases never occur simultaneously. From the above, the constructed edge sequence is traversable.

We analyze the space and time complexity of Algorithm 1. Note that, for saving computation space, we do not implement the Algorithm straightforwardly in some points. We begin with the space complexity. We regard the circle graph $G^0_\text{cir} = (V^0_\text{cir}, E^0_\text{cir})$ as the input. For every $v \in V^0_\text{cir}$, we keep three attributes $\ell^t_\text{in}(v)$, $\ell^t_\text{out}(v)$ and $p^t(v)$ in step $t$. The in- and out-levels are rational numbers that have the form of $i + j/n$. Thus we keep two integers $i$ and $j$ for each in- and out-level. We use $\tilde{O}(n)$ space for preserving them. In step $t$, we also keep $U_t$ by using $\tilde{O}(n)$ space. We need $\tilde{G}_t[U_t]$, 

23
Figure 22: $k > t$: $(v_s^k, v_s^k)$ and $(v_s^k, v_s^k)$ are in the opposite side of $e_s^k$, $p^k(h(e_t^k))$ is equal to $h(e_t^k)$.

but we do not keep $E_t$ explicitly. For $u,v \in U_t$, whether there exists an edge $(u,v)$ in $G_t[U_t]$ is equivalent to whether there exists an edge $(x,y)$ in $E_0^t$ such that $p^t(x) = u$ and $p^t(y) = v$. Since $E_0^t$ is included in the input, we could calculate it with $\tilde{O}(1)$ space. We keep no other information throughout the Algorithm. The number of edges in $G_t[U_t]$ is at most $2|U_t|^2 = O(n^2)$. Thus, for line 4 and 5, we can find a lowest gap-2 chord by using $O(1)$ space. For line 7 and 9, we use only $\tilde{O}(1)$ space for updating $\tilde{V}_t$ and $\tilde{U}_t$. For line 8, we ignore the edges in the upper area of $e_s^k$ (these edges belong to $G_{t+1}[U_{t+1}]$, thus we have no need to keep them). For the edges in the lower area of $e_s^k$, since there exist only gap-1 chords in the area, the number of edges in the area is $O(n)$. We use $\tilde{O}(n)$ space for temporarily keeping them. In MakePlanar (line 10), we look through them, and find crossing points and resolve them and set $\tilde{K}_{t+1}(\cdot)$ by using $\tilde{O}(n)$ space.

Now we consider Algorithm 3. The number of edges in $G_t[U_t]$ is at most $2|U_t|^2 = O(n^2)$. Thus, for line 2, we can find $S^{t}$ and $S^t$ by using $O(1)$ space, and we use $\tilde{O}(n)$ space for keeping them. For line 3 to 6, since $|T^t|, |T^u| = O(n)$, we also use $O(n)$ for keeping $T^t$ and $T^u$. In addition, we use $\tilde{O}(n)$ space for calculating $\ell_{in}^{t+1}(v)$ and $\ell_{out}^{t+1}(v)$ for all $v \in T^t$. In Algorithm 3, we use $\tilde{O}(1)$ space for each operation and the length of for-loops is $O(n)$. Thus we use $\tilde{O}(1)$ space in all. For line 8 to 11, we only refer to in- and out-levels that we are keeping. For line 12 to 15, we do not keep and ignore the labels belonging to edges in the upper area. For line 16, we use $\tilde{O}(1)$ space. For line 17 to 24, we check whether an edge in the lower area was divided by MakePlanar and we use additional $\tilde{O}(1)$
Figure 23: \(k > t\): \((v_*^k, v_*^k)\) and \((v_*^k, v_*^t)\) are in the opposite side of \(e_*^k\), \(p^k(t(e_{i+2}')) = h(e_*^k)\).
We go back to Algorithm 1. For line 12, we output the information of the vertices, edges, labels and values of the path function in the lower area of $e^i$. Here we have to calculate the labels on the gap-1 chords (other information is preserved now). Let the gap-1 chord be $(v_i^p, v_i^q)$. If $p < q$, this edge was made in step $q$ and the labels on the edge were calculated at line 13 of Algorithm 2. Thus, for any $v_i \in V_{0}^{c_i}$ such that $p^i(v) = v_i^p$, we calculate $\ell_{out} = \max_{t_i \in V_{0}^{c_i}, p^i(t_i) = v_i^p} \{\ell_{out}(t_i) \mid (v_i, t_i) \in e_{0}^{c_i}\}$, and $\ell_{in}(v) \rightarrow \ell_{out}$ becomes one of the labels on the edge (if the vertex $v$ is not in $T^u$, $\ell_{out}$ is not defined and a label for $v$ does not exist). On the other hand, if $p > q$, this edge was made in step $p$ and the labels on the edge were calculated at line 14 of Algorithm 2. Thus, for any $v_i \in V_{0}^{c_i}$ such that $p^i(v) = v_i^q$, we calculate $\ell_{in} = \min_{t_i \in V_{0}^{c_i}, p^i(t_i) = v_i^q} \{\ell_{in}(t_i) \mid (t_i, v) \in e_{0}^{c_i}\}$, and $\ell_{in} \rightarrow \ell_{out}(v)$ becomes one of the labels on the edge (if the vertex $v$ is not in $T^u$, $\ell_{in}$ is not defined and a label for $v$ does not exist). We use additional $\widetilde{O}(1)$ space for these calculation. For line 15, we trace line 10 to 12. In total, we use $\widetilde{O}(n)$ space.

Next consider the time complexity. In Lemma 3, we proved that the while-loop at line 4 stops after at most $n$ steps. Since the sizes of $U_i$, $S^i$, $T^i$ and $T^u$ are all $O(n)$, every operation in the Algorithm takes poly($n$) time. Thus this algorithm runs in polynomial time.

Lemma 12. Algorithm 1 runs in polynomial time with using $\widetilde{O}(n)$ space.

From Lemma 5, 6 and 12 we can obtain desired $G_p$ with $\widetilde{O}(n) = \widetilde{O}(N^{1/3})$ space and polynomial time.

4 Apply PlanarReach to a Gadget Graph

By applying PlanarReach to the obtained plane gadget graph $G_p$ with $O(N^{2/3})$ vertices, we can prove Theorem 2. In this section, we explain how to apply PlanarReach to a plane gadget graph, which has labels in edges. We have to modify PlanarReach slightly. We now describe the outline of the algorithm PlanarReach. The notion of “separator” is central to the algorithm.

Definition 3. For any undirected graph $G = (V, E)$ 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 $\rho \cdot |V|$. The size of 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 $O(\sqrt{n})$ [7, 9], and we refer an algorithm which obtains such a separator as Separator.

Let $G = (V, E)$, $s$ and $t$ be the given input; that is, $G$ is a directed graph, and $s$ and $t$ are the start and goal vertices in $V$. We assume that $t$ is reachable from $s$ in $G$, and explain that the algorithm confirms it. We use $G$ to denote an underlying undirected graph of $G$. The algorithm first uses Separator to compute a separator $S$ of size $O(\sqrt{n})$ for $G$, and suppose $G$ is divided into two subgraphs $G[V_0]$ and $G[V_1]$ by $S$ ($V_0 \cap V_1 = \emptyset, V_0 \cup V_1 \cup S = V$). Let us fix a path $p$ from $s$ to $t$. The path $p$ is divided into some $k$ subpaths $p_1, p_2, \ldots, p_k$ by $S$. Note that the end vertex $u_i$ of $p_i$ is on $S$ and whole path $p_i$ is in either one of $G[V_0 \cup S]$ and $G[V_1 \cup S]$. Suppose $p_1$ is in $G[V_0 \cup S]$. By searching in $G[V_0 \cup S]$, we can find $u_1$ is reachable from $s$. The algorithm remembers it and searches $G[V_1 \cup S]$ from $u_1$ in the next step. Then we can find $p_2$, namely $u_2$ is reachable from $u_1$ and $s$. By repeating this procedure, we can confirm $u_i$ is reachable from $s$ for any $i$. More precisely, for each vertex $v \in S$, we keep a boolean value which represents reachability from $s$. In each searching step, we start the search from vertices in $S$ whose boolean values are true. We use this reachability algorithm recursively when searching $G[V_b \cup S]$ ($b \in \{0, 1\}$).
algorithm. In the actual algorithm, we have to control the recursion more carefully, but this is enough for explaining where to modify the algorithm for gadget graphs.

Algorithm 4 PlanarReach($G = (V, E), V_s, R[V_s], V_t$)

\textbf{Input:} A planar graph $G$, start vertices $V_s$, a boolean array $R[V_s]$ for $V_s$, end vertices $V_t$.

\textbf{Task:} Return a boolean array $R[V_t]$ for $V_t$. For any $v \in V_t$, $R[v]$ is true if and only if $v$ is reachable from some vertex $u \in V_s$ such that $R[u]$ is true.

1: if the size of $V$ is small enough then
2: use a standard BFS algorithm and compute $R[V_t]$.
3: return $R[V_t]$
4: else
5: Run Separator and obtain a separator $S$ ($G$ is divided into $G[V_0]$ and $G[V_1]$).
6: $R[S] = \text{PlanarReach}(G[V_0 \cup S \cup V_s], V_s, R[V_s], S)$
7: while unsearched paths remain do
8: $R[S] = \text{PlanarReach}(G[V_0 \cup S], S, R[S], S)$
9: $R[S] = \text{PlanarReach}(G[V_1 \cup S], S, R[S], S)$
10: end while
11: return $\text{PlanarReach}(G[V_1 \cup S \cup V_t], S, R[S], V_t)$
12: end if

Now, we explain where to modify. Let $\tilde{G}_p = (\tilde{V}_p, \tilde{E}_p, \tilde{K}_p, \tilde{L}_p)$ be an input plane gadget graph of PlanarReach and $N$ be the number of vertices of an input grid graph of Algorithm 1. Consider a gadget graph $\tilde{G}_p' = (V_p', E_p', K_p', L_p')$ which is a subgraph of $G_p$. While we execute PlanarReach, for every $v \in S$, we have to keep a boolean value whether $v$ is reachable from $s$ with using $O(|S|)$ space. For $\tilde{G}_p'$, instead of the boolean value, we keep the maximum level that a token starting from $s$ could have at $v$. When $v$ is equal to $t(K'_p(e))$ for some edge $e$, we should keep a specific level that a token can have at $v$ when the token used the edge $e$ last. Such a vertex is made by MakePlanar, and we should keep at most two specific levels for a vertex. Thus we use $\tilde{O}(|S|)$ space for preserving them, and we can still obtain an $\tilde{O}(N^{1/3})$ space algorithm.

For $\tilde{G}_p'$, we use Algorithm 5 like Bellman-Ford algorithm instead of BFS. Algorithm 5 takes as input $\tilde{G}_p'$, a start vertex $s$, an initial level $\ell_s$ and an edge restriction $r \in \tilde{E}_p \cup \{\perp\}$. For any $v \in \tilde{V}_p'$,
the algorithm computes the maximum level that a token starting from s with a level \( \ell_s \) can have at v. When v is equal to \( t(\tilde{K}_p'(e)) \) for some edge e, the algorithm calculates the maximum level that a token can have at v when the token used the edge e last. In Algorithm 5, \( A[v_e] \) means that the maximum level that a token can have at v with using the edge e last, and \( A[v_{\perp}] \) means that the maximum level that a token can have at v with using an edge e last such that \( \tilde{K}_p'(e) = \perp \). At the end of \( t \)-th while-loop, for any \( v \in \tilde{V}_p' \), \( A[v_s] \) has the maximum level which we can have at v within \( t \) steps by starting from s with level \( \ell_s \). At line 4, we use two mappings \( k \) and \( \tilde{K}^{-1} \), and they are defined as follows:

\[
k(e) = \begin{cases} 
\perp & \text{if } \tilde{K}_p'(e) = \perp, \\
e & \text{otherwise}
\end{cases} \quad \tilde{K}^{-1}(e) = \begin{cases} 
e' & \exists e', \tilde{K}_p'(e') = e \\
\perp & \text{otherwise}
\end{cases}
\]

Since the value \( A[\cdot] \) changes no more than two times with the same label, the while-loop will terminate in \( |L_p'| \) steps where \( |L_p'| = |\bigcup_{e \in \tilde{E}_p' \setminus \{\perp\}} \tilde{L}_p'(e)| \). Thus the computation time for Algorithm 5 is \( O(|L_p'|^2) \). An edge has at most \( O(N^{1/3}) \) labels, thus the algorithm runs in polynomial time of \( N \).

**Algorithm 5**

**Input:** A gadget graph \( \tilde{G}_p' = (\tilde{V}_p', \tilde{E}_p', \tilde{K}_p', \tilde{L}_p') \), start vertex \( s \), initial level \( \ell_s \), edge restriction \( r \in \tilde{E}_p \cup \{\perp\} \).

1. initialize \( A[v_{\perp}] = A[v_e] = -1 \) for every \( v \in \tilde{V}_p' \) and \( e \in \tilde{E}_p' \) such that \( h(e) = v \) except for \( s \) and let \( A[s_r] = \ell_s \)
2. while A was changed in the previous loop do
   3. for all \( e \in \tilde{E} \) do
      4. \( A[h(e)_{k(e)}] \leftarrow \max(A[h(e)_{k(e)}], \max\{b \mid a \rightarrow b \in \tilde{L}_p'(e), A[t(e)_{\tilde{K}^{-1}(e)}] \geq a\}) \)
   5. end for
3. end while
6. output \( A \)

**5 Conclusion**

We presented an \( \tilde{O}(n^{1/3}) \) space algorithm for the grid graph reachability problem. The most natural question is whether we can apply our algorithm to the planar graph reachability problem. Although the directed planar reachability is reduced to the directed reachability on grid graphs [1], the reduction blows up the size of the graph by a large polynomial factor and hence it is not useful. Moreover, it is known that there exist planar graphs that require quadratic grid area for embedding [12]. However we do not have to stick to grid graphs. We can apply our algorithm to graphs which can be divided into small blocks efficiently. For instance we can use our algorithm for king’s graphs [6]. More directly, for using our algorithm, it is enough to design an algorithm that divides a planar graph into small blocks efficiently.

**References**

[1] Eric Allender, David A Mix Barrington, Tanmoy Chakraborty, Samir Datta, and Sambuddha Roy. Planar and grid graph reachability problems. Theory of Computing Systems, 45(4):675–723, 2009.
[2] Tetsuo Asano and Benjamin Doerr. Memory-constrained algorithms for shortest path problem. In *CCCG*, 2011.

[3] Tetsuo Asano, David Kirkpatrick, Kotaro Nakagawa, and Osamu Watanabe. $\tilde{O}(\sqrt{n})$-space and polynomial-time algorithm for planar directed graph reachability. In *International Symposium on Mathematical Foundations of Computer Science*, pages 45–56. Springer, 2014.

[4] Greg Barnes, Jonathan F Buss, Walter L Ruzzo, and Baruch Schieber. A sublinear space, polynomial time algorithm for directed st connectivity. *SIAM Journal on Computing*, 27(5):1273–1282, 1998.

[5] Chris Bourke, Raghunath Tewari, and NV Vinodchandran. Directed planar reachability is in unambiguous log-space. *ACM Transactions on Computation Theory (TOCT)*, 1(1):4, 2009.

[6] Gerard Jennhwa Chang. Algorithmic aspects of domination in graphs. *Handbook of Combinatorial Optimization*, pages 221–282, 2013.

[7] Hillel Gazit and Gary L Miller. A parallel algorithm for finding a separator in planar graphs. In *Foundations of Computer Science, 1987., 28th Annual Symposium on*, pages 238–248. IEEE, 1987.

[8] Tatsuya Imai, Kotaro Nakagawa, Aduri Pavan, NV Vinodchandran, and Osamu Watanabe. An $O(n^{1/2+\varepsilon})$-space and polynomial-time algorithm for directed planar reachability. In *Computational Complexity (CCC), 2013 IEEE Conference on*, pages 277–286. IEEE, 2013.

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

[10] Omer Reingold. Undirected connectivity in log-space. *Journal of the ACM (JACM)*, 55(4):17, 2008.

[11] Derrick Stolee and NV Vinodchandran. Space-efficient algorithms for reachability in surface-embedded graphs. In *Computational Complexity (CCC), 2012 IEEE 27th Annual Conference on*, pages 326–333. IEEE, 2012.

[12] Leslie G Valiant. Universality considerations in vlsi circuits. *IEEE Transactions on Computers*, 100(2):135–140, 1981.

[13] Avi Wigderson. The complexity of graph connectivity. *Mathematical Foundations of Computer Science 1992*, pages 112–132, 1992.