$\widetilde{O}(n^{1/3})$-Space Algorithm for the Grid Graph Reachability Problem

Ryo Ashida
Department of Mathematical and Computing Science, Tokyo Institute of Technology
(c/o Professor Osamu Watanabe)
Tokyo, Japan
ashida1@is.titech.ac.jp

Kotaro Nakagawa
JMA SYSTEMS Corporation
Tokyo, Japan
kootaroonakagawa@gmail.com

Abstract

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 $\widetilde{O}(\sqrt{n})$ space[1] 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 $\widetilde{O}(n^{1/3})$ space.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases graph reachability, grid graph, graph algorithm, sublinear space algorithm

Digital Object Identifier 10.4230/LIPIcs.SoCG.2018.5

Related Version A full version of this paper is available at `http://arxiv.org/abs/1803.07097`

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[8]. 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

[1] In this paper "$\widetilde{O}(s(n))$ space" means $O(s(n))$ words intuitively and precisely $O(s(n) \log n)$ space.

© Ryo Ashida and Kotaro Nakagawa;
licensed under Creative Commons License CC-BY

there exist an algorithm for the directed graph reachability problem that uses polynomial time and $O(n^\varepsilon)$ space, for some $\varepsilon < 1$? [11], and this question is still open. The best known polynomial time algorithm, shown by Barns, Buss, Ruzzo and Schieber, uses $O(n/2^{\sqrt{\log n}})$ 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 [9]. 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, 7]. Moreover Asano et al. devised a efficient way to control the recursion, and proposed a polynomial time and $\widetilde{O}(\sqrt{n})$ space algorithm for the planar graph 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 $\widetilde{O}(\sqrt{n})$ space and polynomial time. The main result of this paper is to show an $\widetilde{O}(n^{1/3})$ 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 $\widetilde{O}(\sqrt{n})$ 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 a 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 jth bit of the input to A_i, simply execute A_{i-1} (from the beginning) until it yields the desired jth 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 PlanarReach to the transformed graph of size $O(n^{2/3})$, then the reachability is computable in $\widetilde{O}\left(\sqrt{n^{2/3}}\right) = \widetilde{O}(n^{1/3})$ 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 PlanarReach in step 3, which runs in $\widetilde{O}(n^{1/3})$ space.

▶ **Theorem 2.** *There exists an algorithm that computes the grid graph reachability in polynomial-time and $\widetilde{O}(n^{1/3})$ 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^{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 $|V_0^{\mathrm{rim}}|$, 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, $\widetilde{G}_{\mathsf{p}}$ with $O(n) = O(N^{1/3})$ vertices including V_0^{rim} so that the reachability among vertices in V_0^{rim} on G_0 remains the same on $\widetilde{G}_{\mathsf{p}}$.

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

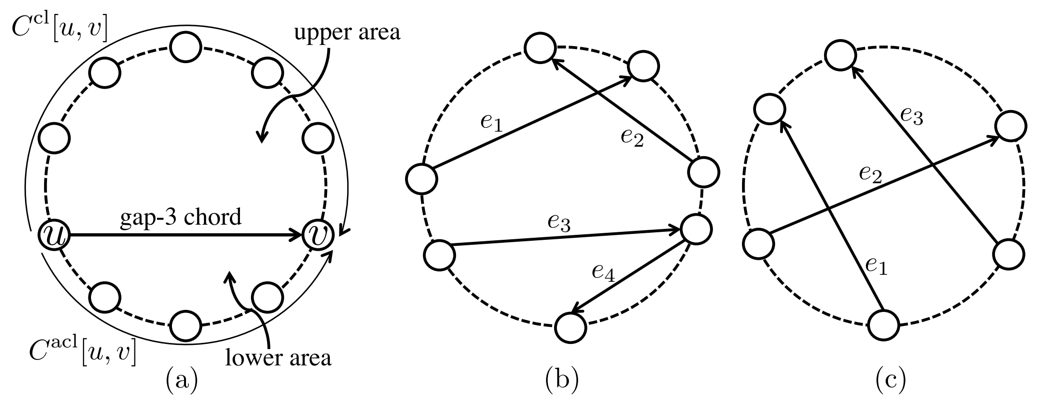

Figure 1 An example of the notions on chords. (a) a figure showing a chord, arcs, a lower area, an upper area, (b) a figure showing crossing chords (e_1 and e_2) and semi-crossing chords (e_3 and e_4) and (c) separating chords (e_3 separates e_1 and e_2).

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 $C^{\rm cl}[u,v]$ (resp., $C^{\rm acl}[u,v]$) to denote this tour (Figure 1(a)). When we would like to specify the graph G, we use $C^{\rm cl}_G[u,v]$ (resp., $C^{\rm acl}_G[u,v]$). The tour $C^{\rm cl}[u,v]$, 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 $C^{\rm cl}(u,v)$ and $C^{\rm cl}[u,v)$ (resp., $C^{\rm acl}(u,v)$ and $C^{\rm acl}[u,v)$) 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, $C^{\rm cl}[u,v]$ and $C^{\rm acl}[u,v]$; 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 $C^{\rm cl}[u,v]$" for specifying which one is currently regarded as the arc. A *gap-d* (resp., *gap-d^+*) *chord* is a chord (u, v) whose arc $C[u,v]$ is of length $d+2$ (resp., length $\geq d+2$). 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 (u_1, v_1) and (u_2, v_2) *cross* if they cross in the circle C in a natural way (see Figure 1(b)). Formally, we say that (u_1, v_1) *crosses* (u_2, v_2) if either (i) u_2 is on the tour $C^{\rm cl}(u_1,v_1)$ and v_2 is on the tour $C^{\rm acl}(u_1,v_1)$, or (ii) v_2 is on the tour $C^{\rm cl}(u_1,v_1)$ and u_2 is on the tour $C^{\rm acl}(u_1,v_1)$. Also, we say that (u_1, v_1) *semi-crosses* (u_2, v_2) if either (i) u_2 is on the tour $C^{\rm cl}[u_1,v_1]$ and v_2 is on the tour $C^{\rm acl}[u_1,v_1]$, or (ii) v_2 is on the tour $C^{\rm cl}[u_1,v_1]$ and u_2 is on the tour $C^{\rm acl}[u_1,v_1]$ (see Figure 1(b)). Note that clearly crossing implies semi-crossing. In addition, we say that a chord (u_1, v_1) *separates* two chords (u_2, v_2) and (u_3, v_3) if the endpoints of two chords v_2 and

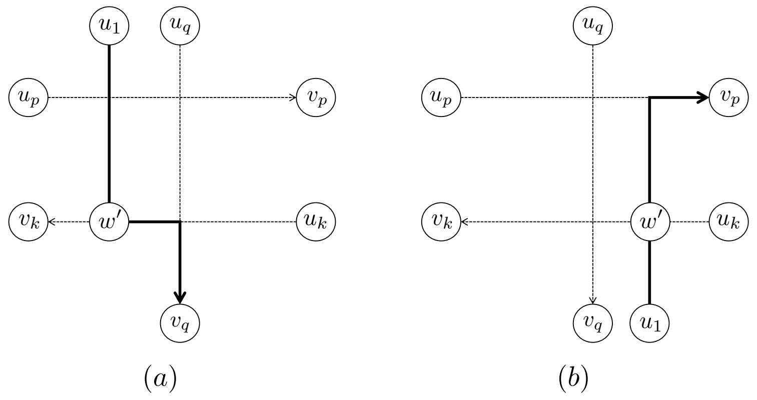

Figure 2 A common vertex w' of a path from u_k to v_k and a path from u_1 to v_q or v_p for some $p, q < k$.

v_3 are separated by the chord (u_1, v_1) (see Figure 1(c)). Formally, (u_1, v_1) *separates* (u_2, v_2) and (u_3, v_3) if either (i) v_2 is on the tour $C^{\mathrm{cl}}[u_1, v_1]$ and v_3 is on the tour $C^{\mathrm{acl}}[u_1, v_1]$, or (ii) v_3 is on the tour $C^{\mathrm{cl}}[u_1, v_1]$ and v_2 is on the tour $C^{\mathrm{acl}}[u_1, v_1]$. We say that k chords (u_1, v_1), $(u_2, v_2), \ldots, (u_k, v_k)$ are *traversable* if the following two conditions are satisfied:

1. (u_1, v_1) semi-crosses (u_2, v_2),
2. $\forall i \in [3, k]$, $\exists p, q < i$, (u_i, v_i) separates (u_p, v_p) and (u_q, v_q).

Now for the graph $G_0 = (V_0, E_0)$, we define the circle graph $G_0^{\mathrm{cir}} = (V_0^{\mathrm{cir}}, E_0^{\mathrm{cir}})$ by

$$\begin{aligned} V_0^{\mathrm{cir}} &= V_0^{\mathrm{rim}}, \text{ and} \\ E_0^{\mathrm{cir}} &= \{\, (u, v) \mid \exists \text{path from } u \text{ to } v \text{ in } G_0 \,\}, \end{aligned}$$

where we assume that the rim vertices of V_0^{cir} $(= V_0^{\mathrm{rim}})$ are placed on a cycle C_0 as they are on the rim of the block in the grid graph. Then it is clear that G_0^{cir} keeps the same reachability relation among vertices in $V_0^{\mathrm{cir}} = V_0^{\mathrm{rim}}$. Recall that G_0 has $O(n^2)$ vertices. Thus, by using PlanarReach, we can show the following lemma.

▶ **Lemma 3.** *G_0^{cir} keeps the same reachability relation among vertices in $V_0^{\mathrm{cir}} = V_0^{\mathrm{rim}}$. That is, for any pair u, v of vertices of V_0^{cir}, v is reachable from u in G_0^{cir} if and only if it is reachable from u in G_0. There exists an algorithm that transforms G_0 to G_0^{cir} in $O(n)$-space and polynomial-time in n.*

The notion of traversable is a key for discussing the reachability on G_0^{cir}. Based on the following lemma, we use a traversable sequence of edges for characterizing the reachability on the circle graph G_0^{cir}.

▶ **Lemma 4.** *For a circle graph $G_0^{\mathrm{cir}} = (V_0^{\mathrm{cir}}, E_0^{\mathrm{cir}})$ obtained from a block grid graph G_0, if there are traversable edges (u_1, v_1), $(u_2, v_2), \ldots, (u_k, v_k) \in E_0^{\mathrm{cir}}$, then $(u_1, v_k) \in E_0^{\mathrm{cir}}$.*

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, and 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 and v_2 is reachable from w, there exists a path from u_1 to v_2.

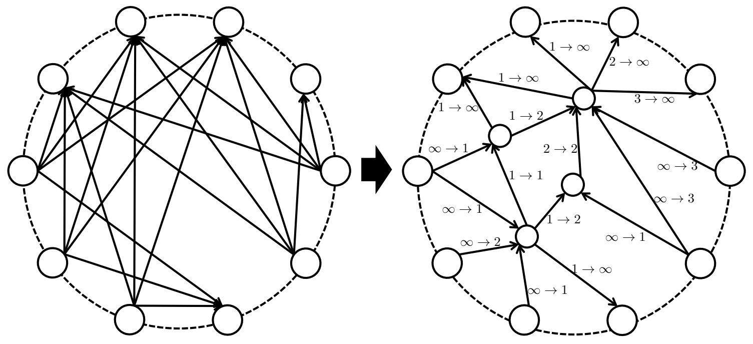

Figure 3 An example of the transformation from a circle graph to a gadget graph.

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_q} 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 5.** A *gadget graph* $\widetilde{G}$ is a graph defined by a tuple $(\widetilde{V}, \widetilde{E}, \widetilde{K}, \widetilde{L})$, where $\widetilde{V}$ is a set of vertices, $\widetilde{E}$ is a set of edges, $\widetilde{K}$ is a *path function* that assigns an edge or $\perp$ to each edge, and $\widetilde{L}$ 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 $(u, v) \in \widetilde{E}$, we may use expressions $\widetilde{K}(u, v)$ and $\widetilde{L}(u, v)$ instead of $\widetilde{K}((u, v))$ and $\widetilde{L}((u, v))$ for simplicity.

Our goal is to transform a given circle graph (obtained from a block grid graph) $G_0^{\mathrm{cir}} = (V_0^{\mathrm{cir}}, E_0^{\mathrm{cir}})$ in which all vertices in V_0^{cir} are placed on a cycle C to a *plane* gadget graph $\widetilde{G}_{\mathsf{p}} = (\widetilde{V}_{\mathsf{p}}^{\mathrm{out}} \cup \widetilde{V}_{\mathsf{p}}^{\mathrm{in}}, \widetilde{E}_{\mathsf{p}}, \widetilde{K}_{\mathsf{p}}, \widetilde{L}_{\mathsf{p}})$ where $\widetilde{V}_{\mathsf{p}}^{\mathrm{out}}$ is the set of *outer vertices* that are exactly the vertices of V_0^{cir} placed in the same way as G_0^{cir} on the cycle C, and $\widetilde{V}_{\mathsf{p}}^{\mathrm{in}}$ is the set of *inner vertices* placed inside of C. All edges of $\widetilde{E}_{\mathsf{p}}$ are also placed inside of C under our embedding. The inner vertices of $\widetilde{V}_{\mathsf{p}}^{\mathrm{in}}$ are used to replace crossing points of edges of E_0^{cir} to transform to a planar graph (see Figure 3). We would like to keep the "reachability" among vertices in $\widetilde{V}_{\mathsf{p}}^{\mathrm{out}}$ in $\widetilde{G}_{\mathsf{p}}$ while bounding $|\widetilde{V}_{\mathsf{p}}^{\mathrm{in}}| = O(n)$.

We explain how to characterize the reachability on a gadget graph. Consider any gadget graph $\widetilde{G} = (\widetilde{V}, \widetilde{E}, \widetilde{K}, \widetilde{L})$, and let x and y be any two vertices of $\widetilde{V}$. Intuitively, the reachability from x to y is characterized by a directed path on which we can send a token from x to y.

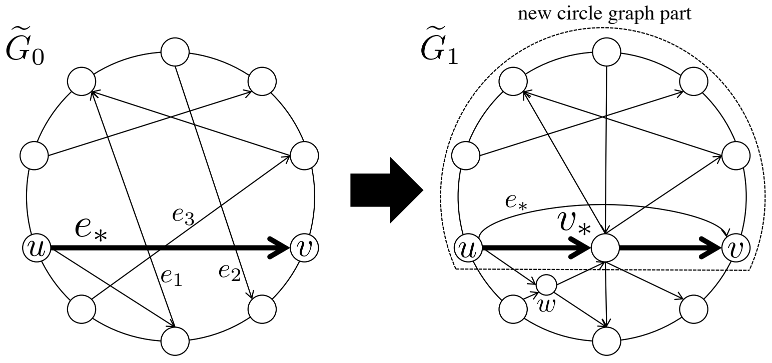

Figure 4 An initial transformation step from $\widetilde{G}_0$ to $\widetilde{G}_1$.

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 ∞ when the token is at vertex x. (For a general discussion, we use a parameter ℓ_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 ℓ, it can "go through" e_j to reach its head vertex $h(e_j)$ if $\widetilde{L}(e_j)$ has an *available label* $i_j \to o_j$ such that $i_j \le \ell$ holds for its in-level i_j. If the token uses a label $i_j \to o_j$, 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 $\widetilde{K}$ so that some edge can specify the next edge. We consider only a path $p = (e_1, \ldots, e_m)$ as "valid" such that $e_{i+1} = \widetilde{K}(e_i)$ for all e_i such that $\widetilde{K}(e_i) \neq \bot$. We characterize the reachability from x to y on gadget graph $\widetilde{G}$ by using a valid path realizing a token tour from x to y.

▶ **Definition 6.** For any gadget graph $\widetilde{G} = (\widetilde{V}, \widetilde{E}, \widetilde{K}, \widetilde{L})$, and for any two vertices x, y of $\widetilde{V}$, there exists a *token tour* from x to y with initial level ℓ_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. $h(e_i) = t(e_{i+1})$ $(1 \le i < m)$,
3. if $\widetilde{K}(e_i)$ is not $\bot$ $(1 \le i < m)$, then $e_{i+1} = \widetilde{K}(e_i)$,
4. there exist labels $i_1 \to o_1 \in \widetilde{L}(e_1), \ldots, i_m \to o_m \in \widetilde{L}(e_m)$ such that $\ell_s \ge i_1$ and $o_t \ge i_{t+1}$ for all $1 \le t < m$.

At the beginning of our algorithm, we obtain a gadget graph $\widetilde{G}_0 = (\widetilde{V}_0, \widetilde{E}_0, \widetilde{K}_0, \widetilde{L}_0)$ whose base graph is equal to $G_0^{\rm cir}$, and $\widetilde{K}_0(e) = \bot$, $\widetilde{L}_0(e) = \{0 \to \infty\}$ for every $e \in \widetilde{E}_0$. It is obvious that $G_0^{\rm cir}$ and $\widetilde{G}_0$ have the same reachability. Namely, there exists a token tour from x to y for $x, y \in \widetilde{V}_0$ in $\widetilde{G}_0$ if and only if there exists an edge $(x, y) \in \widetilde{E}_0$.

We explain first the outline of our transformation from $\widetilde{G}_0$ to $\widetilde{G}_{\sf p}$. We begin by finding a chord $e_* = (u, v)$ with gap ≥ 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 $\widetilde{G}_0$ crossing this chord e_* (e_1, e_2 and e_3 in Figure 4). Create a new inner vertex v_* of $\widetilde{G}_{\sf p}$ 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

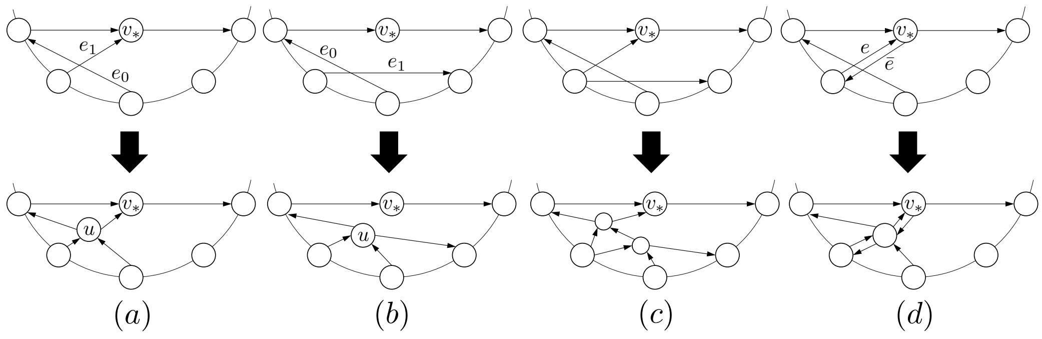

Figure 5 Examples of vertices made by MakePlanar.

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 $\widetilde{G}_0$ by replacing the arc $C[u, v]$ by a tour (u, v_*, v) to create a new circle graph part of $\widetilde{G}_1$. We then repeat this transformation step on the circle graph part of $\widetilde{G}_1$. In the algorithm, U_t is the vertices of the circle graph part of $\widetilde{G}_t$, thus $\widetilde{G}_t[U_t]$ indicates the circle graph part of $\widetilde{G}_t$. Note that e_* is not removed and becomes a gap-1 chord in the next step.

We explain step (ii) for $\widetilde{G}_0$ 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 $\widetilde{K}_1(t(e_i), u) = (u, h(e_i))$. The edges might be divided into more than two segments (see Figure 5(c)). We call the edge of $\widetilde{G}_0$ *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 $\bar{e} = (h(e), t(e))$ (see Figure 5(d)). In this case, e and $\bar{e}$ share a new vertex for resolving crossing points. For $\widetilde{G}_t[U_t]$ $(t > 0)$, we process the lower area in the same way. We refer to this algorithm as MakePlanar, and the new inner vertices created by MakePlanar in step t as V^t_{MP}.

The detailed process of step (iii) is written in Algorithm 2, and Algorithm 1 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 7.** *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^t_* 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^t_*, is also at most n. If a gap-k chord is picked, we make at most $2k - 1$ new inner vertices by MakePlanar, namely $|V^t_{\mathsf{MP}}| \leq 2k - 1$, 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, $|\widetilde{V}^{\text{out}}_{\mathsf{p}} \cup \widetilde{V}^{\text{in}}_{\mathsf{p}}| \leq n + 5n = 6n$. ◀

Algorithm 1

```
Input: A circle graph $G_0^{\rm cir} = (V_0^{\rm cir}, E_0^{\rm cir})$ obtained from a block graph.
Task: Output a plane gadget graph $\widetilde{G}_{\sf p} = (\widetilde{V}_{\sf p}^{\rm out} \cup \widetilde{V}_{\sf p}^{\rm in}, \widetilde{E}_{\sf p}, \widetilde{K}_{\sf p}, \widetilde{L}_{\sf p})$ which satisfies $\widetilde{V}_{\sf p}^{\rm out} = V_0^{\rm cir}$
    and the reachability among vertices in $\widetilde{V}_{\sf p}^{\rm out}$ in $\widetilde{G}_{\sf p}$ is the same as $G_0^{\rm cir}$.
1:  initialize $t = 0$ // loop counter
2:  $\widetilde{G}_0 = (\widetilde{V}^{\rm out} \cup \widetilde{V}_0, \widetilde{E}_0, \widetilde{K}_0, \widetilde{L}_0)$ where $\widetilde{V}^{\rm out} \leftarrow V_0^{\rm cir}, \widetilde{V}_0 \leftarrow \emptyset, \widetilde{E}_0 \leftarrow E_0^{\rm cir}, \widetilde{K}_0(e) \leftarrow \perp, \widetilde{L}_0(e) \leftarrow$
    $\{0 \to \infty\}$ for each $e \in E_0^{\rm cir}$, and $U_0 \leftarrow \widetilde{V}^{\rm out}$
3:  for every $v \in \widetilde{V}^{\rm out}$, $\ell_{in}^0(v) \leftarrow 0, \ell_{out}^0(v) \leftarrow \infty, p^0(v) \leftarrow v$.
4:  while $\widetilde{G}_t[U_t]$ has a lowest gap-$2^+$ chord do
5:      pick a lowest gap-$2^+$ chord $e_*^t$
6:      make a new vertex $v_*^t$
7:      $\widetilde{V}_{t+1} \leftarrow \widetilde{V}_t \cup \{v_*^t\}$
8:      $\widetilde{E}_{t+1} \leftarrow (\widetilde{E}_t \cup \{(t(e), v_*^t), (v_*^t, h(e)) \mid e \text{ crosses } e_* \text{ or } e = e_*\}) \setminus \{e \mid e \text{ crosses } e_*\}$
9:      $U_{t+1} \leftarrow (U_t \cup \{v_*^t\}) \setminus C_{\widetilde{G}_t[U_t]}(t(e_*^t), h(e_*^t))$
10:     use MakePlanar to make the lower area of $e_*^t$ planar and update $\widetilde{V}_{t+1}$, $\widetilde{E}_{t+1}$ and $\widetilde{K}_{t+1}$.
11:     change the labels by using Algorithm 2 for keeping reachability
12:     output $\widetilde{G}_{t+1}[C_{\widetilde{G}_t[U_t]}[t(e_*^t), h(e_*^t)] \cup \{v_*^t\} \cup V_{\sf MP}^t]$, which is the lower area of $e_*^t$.
13:     $t \leftarrow t + 1$
14: end while
15: use MakePlanar to make $\widetilde{G}_t[U_t]$ planar and assign labels by line 17-24 of Algorithm 2.
16: output $\widetilde{G}_t[U_t \cup V_{\sf MP}^t]$
```

Now we explain Algorithm 2 describing how to assign labels to $\widetilde{G}_{t+1}$ constructed in Algorithm 1. For each outer vertex $v \in \widetilde{V}^{\rm out}$, we keep three attributes $p^t(v)$, $\ell_{in}^t(v)$ and $\ell_{out}^t(v)$, 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 $\ell_{in}^t(v)$. On the other hand, for the token tour from $p^t(v)$, it is enough to have $\ell_{out}^t(v)$ as an initial level to reach v. We will show these facts implicitly in the proof of Lemma 8.

At the beginning of each iteration of Algorithm 1, we choose a lowest gap-2^+ chord e_*^t. We collect vertices in U_t which are endpoints for some edges crossing with e_*^t, and we refer to the vertices among them which are in the lower area of e_*^t as S^ℓ and the vertices in the upper area of e_*^t as S^u (see Figure 6(a) and line 2). Next we collect vertices whose parents are in S^ℓ (resp., S^u), and we denote them by T^ℓ (resp., T^u) (line 3). Let x' and y' be vertices whose parents are $t(e_*^t)$ and $h(e_*^t)$ respectively. We assign indices to the vertices in T^u and T^ℓ such that the nearer to x' a vertex is located, the larger index the vertex has (see Figure 6(b)). We regard T^ℓ as a sequence $(t_1^\ell, t_2^\ell, \ldots, t_{|T^\ell|}^\ell)$, and T^u as a sequence $(t_1^u, t_2^u, \ldots, t_{|T^u|}^u)$. For each vertex t_i^ℓ in T^ℓ, we calculate $\ell_{in}^{t+1}(t_i^\ell)$ and $\ell_{out}^{t+1}(t_i^\ell)$ in Algorithm 3. From line 1 to 4, we decide temporary values of $\ell_{in}^{t+1}(t_i^\ell)$ and $\ell_{out}^{t+1}(t_i^\ell)$ according to reachability among vertices in T^ℓ and T^u in $G_0^{\rm cir}$. When t_j^u has the maximum index among vertices that t_i^ℓ can reach in T^u, we let $\ell_{in}^{t+1}(t_i^\ell) = j + i/n$. When t_j^u has the minimum index among vertices which can reach t_i^ℓ in T^u, we let $\ell_{out}^{t+1}(t_i^\ell) = j + i/n$. The term i/n is for breaking ties. In the next for-loop, we change the in- and out-levels so that the in-level of the larger indexed vertex is larger than the out-level of the smaller indexed vertex. If there exists a vertex t_j^ℓ such that $i > j$ and $\ell_{out}^{t+1}(t_j^\ell) > \ell_{in}^{t+1}(t_i^\ell)$, then we let $\Delta = (\ell_{out}^{t+1}(t_j^\ell) - j/n) - (\ell_{in}^{t+1}(t_i^\ell) - i/n)$ and add Δ to $\ell_{in}^{t+1}(t_i^\ell)$ and $\ell_{out}^{t+1}(t_i^\ell)$. For preserving the magnitude relationship between in- and out-levels

Algorithm 2

```
Task: Set $\widetilde{L}_{t+1}$ so that $\widetilde{G}_{t+1}$ has the same reachability as $\widetilde{G}_t$
1: For every edge $e$ appearing in both $\widetilde{G}_t$ and $\widetilde{G}_{t+1}$, let $\widetilde{L}_{t+1}(e) = \widetilde{L}_t(e)$.
2: $S^\ell$ (resp., $S^u$) $\leftarrow \{v \in U \mid \exists e \in \widetilde{E}_t$ s.t. $e$ crosses $e^t_*$, $t(e) = v$ or $h(e) = v$, and $v$ is at the lower (resp., upper) area of $e^t_*\}$
3: $T^\ell$ (resp., $T^u$) $\leftarrow \{v \in V^{\text{cir}}_0 \mid p^t(v) \in S^\ell$ (resp. $S^u$)$\}$
4: Fix any vertices $x', y' \in V^{\text{cir}}_0$ such that $p^t(x') = t(e^t_*), p^t(y') = h(e^t_*)$.
5: Set an order to $T^\ell$ according to the order appearing in $C_{G^{\text{cir}}_0}[y', x']$. We regard $T^\ell$ as a sequence $(t^\ell_1, t^\ell_2, \dots, t^\ell_{|T^\ell|})$ (see Figure 6(b)).
6: Set an order to $T^u$ in the same way as $T^\ell$ but according to the tour along the other arc. We also regard $T^u$ as a sequence $(t^u_1, t^u_2, \dots, t^u_{|T^u|})$ (see Figure 6(b)).
7: Use Algorithm 3 for calculating $\ell^{t+1}_{in}(v)$ and $\ell^{t+1}_{out}(v)$ for all $v \in T^\ell$.
8: for $u \in S^\ell$ do
9:     $\widetilde{L}_{t+1}(u, v^t_*) \leftarrow \{\ell^t_{in}(v) \to \ell^{t+1}_{in}(v) \mid p^t(v) = u\}$
10:    $\widetilde{L}_{t+1}(v^t_*, u) \leftarrow \{\ell^{t+1}_{out}(v) \to \ell^t_{out}(v) \mid p^t(v) = u\}$
11: end for
12: for $u \in S^u$ do
13:    $\widetilde{L}_{t+1}(u, v^t_*) \leftarrow \{\ell^t_{in}(t^u_i) \to \max_{t^\ell \in T^\ell}\{\ell^{t+1}_{out}(t^\ell) | (t^u_i, t^\ell) \in E^{\text{cir}}_0\} | t^u_i \in T^u$ and $p^t(t^u_i) = u\}$
14:    $\widetilde{L}_{t+1}(v^t_*, u) \leftarrow \{\min_{t^\ell \in T^\ell}\{\ell^{t+1}_{in}(t^\ell) \mid (t^\ell, t^u_i) \in E^{\text{cir}}_0\} \to \ell^t_{out}(t^u_i) | t^u_i \in T^u$ and $p^t(t^u_i) = u\}$
15: end for
16: $\widetilde{L}_{t+1}(t(e^t_*), v^t_*) \leftarrow \{\infty \to 0\}$, $\widetilde{L}_{t+1}(v^t_*, h(e^t_*)) \leftarrow \{\infty \to 0\}$
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:        $\widetilde{L}_{t+1}(e) \leftarrow \{a \to b \mid a \to b \in \widetilde{L}_t(e')\}$
21:    else
22:        $\widetilde{L}_{t+1}(e) \leftarrow \{b \to b \mid a \to b \in \widetilde{L}_t(e')\}$
23:    end if
24: end for
25: for $v \in \{w \in U_t \mid w$ is at the lower area of $e^t_*\}$ do
26:    $p^{t+1}(v) = v^t_*$
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 $v \in T^\ell$.
1: for $i \in [1, |T^\ell|]$ do
2:     $\ell^{t+1}_{in}(t^\ell_i) \leftarrow \max\{j \mid (t^\ell_i, t^u_j) \in E^{\text{cir}}_0,\ t^u_j \in T^u\} + i/n$
3:     $\ell^{t+1}_{out}(t^\ell_i) \leftarrow \min\{j \mid (t^u_j, t^\ell_i) \in E^{\text{cir}}_0,\ t^u_j \in T^u\} + i/n$
4: end for
5: for $i = 1$ to $|T^\ell|$ do
6:     $\Delta \leftarrow \max(0, \max\{\ell^{t+1}_{out}(t^\ell_j) - j/n \mid 1 \le j < i\} - (\ell^{t+1}_{in}(t^\ell_i) - i/n))$
7:     for $k \in [i, |T^\ell|]$ do
8:         $\ell^{t+1}_{in}(t^\ell_k) \leftarrow \ell^{t+1}_{in}(t^\ell_k) + \Delta$
9:         $\ell^{t+1}_{out}(t^\ell_k) \leftarrow \ell^{t+1}_{out}(t^\ell_k) + \Delta$
10:    end for
11: end for
```

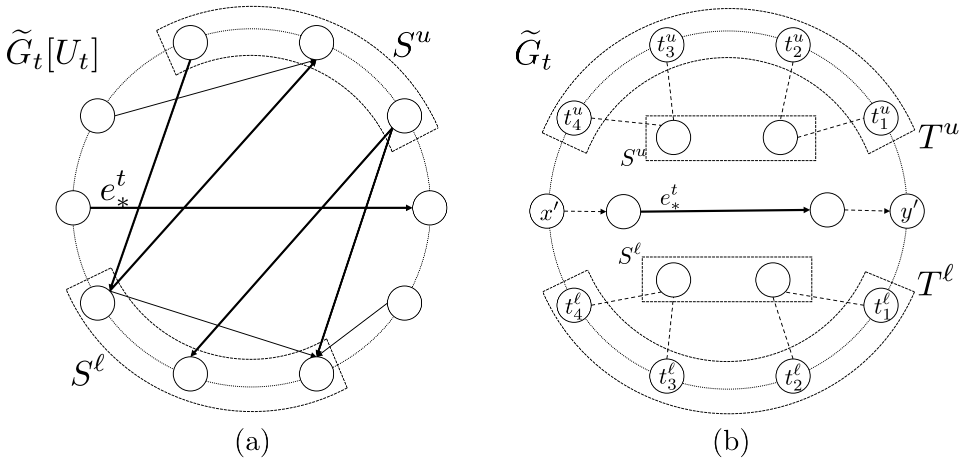

Figure 6 (a) An example of S^ℓ and S^u, (b) An example of T^ℓ and T^u.

of t_i^ℓ and those of t_k^ℓ $(k > i)$, we also add Δ to $\ell_{in}^{t+1}(t_k^\ell)$ and $\ell_{out}^{t+1}(t_k^\ell)$.

Back to Algorithm 2. From line 8 to 16, we assign labels to edges newly appearing in $\widetilde{G}_{t+1}$. Let v be any vertex in T^ℓ. For edges in the lower area of e_*^t, the edge $(p^t(v), v_*^t)$ has a label $\ell_{in}^t(v) \rightarrow \ell_{in}^{t+1}(v)$ (line 9), and the edge $(v_*^t, p^t(v))$ has a label $\ell_{out}^{t+1}(v) \rightarrow \ell_{out}^t(v)$ (line 10). Consider edges in the upper area of e_*^t. Let v be any vertex in T^u. The edge $(p^t(v), v_*^t)$ has a label $\ell_{in}^t(v) \rightarrow \ell_{max}$ where ℓ_{max} is the maximum in-level of vertices in T^ℓ that can reach v (line 13). The edge $(v_*^t, p^t(v))$ has a label $\ell_{min} \rightarrow \ell_{out}^t(v)$ where ℓ_{min} is the minimum out-level of vertices in T^ℓ that v can reach (line 14). The edges $(t(e_*^t), v_*^t)$ 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 $(u, w_1), (w_1, w_2), \ldots (w_k, v)$ 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 $b \rightarrow b$.

From line 25 to 27, we update the parents of the vertices in the lower area of e_*^t. For each vertex v in U_t and in the lower area of e_*^t, we let $p^{t+1}(v) = v_*^t$.

The following lemma shows that paths in $\widetilde{G}_0$ remain in $\widetilde{G}_t$ for every t.

▶ **Lemma 8.** *For any t in Algorithm 1, if there exists an edge from x toward y in $\widetilde{G}_0$, then there exists a token tour from x to y in $\widetilde{G}_t$ whose length is at most $2t + 1$.*

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 4, it is enough to prove the following Lemma.

▶ **Lemma 9.** *For any t and $x, y \in V_0^{\mathrm{cir}}$, if there exists a token tour from x to y in $\widetilde{G}_t$, then there exists a traversable edge sequence $(e_1, \ldots, e_k)$ in G_0^{cir} such that $t(e_1) = x$ and $h(e_k) = y$.*

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 $G_0^{\mathrm{cir}} = (V_0^{\mathrm{cir}}, E_0^{\mathrm{cir}})$ as the input. For every $v \in V_0^{\mathrm{cir}}$, we keep three attributes $\ell_{in}^t(v)$, $\ell_{out}^t(v)$ 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 $\widetilde{O}(n)$ space for preserving them. In step

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

Now we consider Algorithm 2. The number of edges in $\widetilde{G}_t[U_t]$ is at most $2|U_t|^2 = O(n^2)$. Thus, for line 2, we can find S^ℓ and S^u by using $\widetilde{O}(1)$ space, and we use $\widetilde{O}(n)$ space for keeping them. For line 3 to 6, since $|T^\ell|, |T^u| = O(n)$, we also use $\widetilde{O}(n)$ for keeping T^ℓ and T^u. In addition, we use $\widetilde{O}(n)$ space for calculating $\ell_{in}^{t+1}(v)$ and $\ell_{out}^{t+1}(v)$ for all $v \in T^\ell$. In Algorithm 3, we use $\widetilde{O}(1)$ space for each operation and the length of for-loops is $O(n)$. Thus we use $\widetilde{O}(1)$ 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 $\widetilde{O}(1)$ space. For line 17 to 24, we check whether an edge in the lower area was divided by MakePlanar and we use additional $\widetilde{O}(1)$ space. For line 25 to 27, we can find all vertices in the lower area of e_*^t by using $\widetilde{O}(n)$ space, and we use additional $\widetilde{O}(1)$ space for updating $p^{t+1}(\cdot)$.

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_*^t. Here we have to calculate the labels on the gap-1 chords (other information is preserved now). Let the gap-1 chord be (v_*^p, v_*^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 $v \in V_0^{\mathrm{cir}}$ such that $p^t(v) = v_*^p$, we calculate $\ell_{out} = \max_{t^\ell \in V_0^{\mathrm{cir}}, p^t(t^\ell) = v_*^q} \{\ell_{out}^t(t^\ell) \mid (v, t^\ell) \in E_0^{\mathrm{cir}}\}$, and $\ell_{in}^t(v) \to \ell_{out}$ becomes one of the labels on the edge (if the vertex v is not in T^u, ℓ_{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 $v \in V_0^{\mathrm{cir}}$ such that $p^t(v) = v_*^q$, we calculate $\ell_{in} = \min_{t^\ell \in V_0^{\mathrm{cir}}, p^t(t^\ell) = v_*^p} \{\ell_{in}^t(t^\ell) \mid (t^\ell, v) \in E_0^{\mathrm{cir}}\}$, and $\ell_{in} \to \ell_{out}^t(v)$ becomes one of the labels on the edge (if the vertex v is not in T^u, ℓ_{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 7, we proved that the while-loop at line 4 stops after at most n steps. Since the sizes of U_t, S^ℓ, S^u, T^ℓ and T^u are all $O(n)$, every operation in the Algorithms takes poly(n) time. Thus this algorithm runs in polynomial time.

▶ **Lemma 10.** *Algorithm 1 runs in polynomial time with using $\widetilde{O}(n)$ space.*

From Lemma 8, 9 and 10, we can obtain $\widetilde{G}_{\mathsf{p}}$ with $\widetilde{O}(n) = \widetilde{O}(N^{1/3})$ space and polynomial time. By applying PlanarReach to the plane gadget graph with $O(N^{2/3})$ vertices, we can prove Theorem 2.

4 Conclusion

We presented an $\widetilde{O}(n^{1/3})$ 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 [10]. 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. $\widetilde{O}(\sqrt{n})$-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 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.

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

9 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.

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

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