Space-Efficient Algorithms for Reachability in Directed Geometric Graphs

Sujoy Bhore ✉ ⬤
Indian Institute of Science Education and Research, Bhopal, India

Rahul Jain ✉ ⬤
Fernuniversität in Hagen, Germany

---- **Abstract** ----

The problem of graph REACHABILITY is to decide whether there is a path from one vertex to another in a given graph. In this paper, we study the REACHABILITY problem on three distinct graph families - intersection graphs of Jordan regions, unit contact disk graphs (penny graphs), and chordal graphs. For each of these graph families, we present space-efficient algorithms for the REACHABILITY problem.

For intersection graphs of Jordan regions, we show how to obtain a "good" *vertex separator* in a space-efficient manner and use it to solve the REACHABILITY in polynomial time and $O(m^{1/2} \log n)$ space, where n is the number of Jordan regions, and m is the total number of crossings among the regions. We use a similar approach for chordal graphs and obtain a polynomial time and $O(m^{1/2} \log n)$ space algorithm, where n and m are the number of vertices and edges, respectively. However, for unit contact disk graphs (penny graphs), we use a more involved technique and obtain a better algorithm. We show that for every $\epsilon > 0$, there exists a polynomial time algorithm that can solve REACHABILITY in an n vertex directed penny graph, using $O(n^{1/4+\epsilon})$ space. We note that the method used to solve penny graphs does not extend naturally to the class of geometric intersection graphs that include arbitrary size cliques.

1 Introduction

Given a directed graph $G = (V, E)$ and two of its vertices s and t, the problem of REACHABILITY is to decide if there exists a path from s to t in G. REACHABILITY is one of the fundamental problems in theoretical computer science, and for directed and undirected graphs the problem is known to be complete for the classes NL and L, respectively (see [26, 28]). The famous open question $L \overset{?}{=} NL$ essentially asks if there exists a deterministic log-space algorithm for REACHABILITY or not. Note that REACHABILITY can be solved in $\Theta(n \log n)$ space and optimal time by using standard graph traversal algorithms such as DFS and BFS. Furthermore, it is known that this problem can be solved in $\Theta(\log^2 n)$-space and $n^{\Theta(\log n)}$ time [32].

In the realm of space-efficient algorithms, the primary objective is to optimize the space-complexity of an algorithm while maintaining a polynomial-time bound. Wigderson asked in his survey of REACHABILITY problems [33], whether there is an algorithm for REACHABILITY that runs in $O(n^{1-\varepsilon})$ space (for some $\varepsilon > 0$) and polynomial time. Barnes et al. [6] partially answered this question and showed that REACHABILITY on general graphs can be solved in polynomial time and $O(n/2^{\Theta(\sqrt{\log n})})$ space. This result is followed by numerous works on various restricted graph families.

Asano and Doerr [3] presented an algorithm for grid graphs that uses $O(n^{1/2+\varepsilon})$-space, for any small $\varepsilon > 0$. Imai et al. [23] achieved the similar space bound for planar graphs. Later, Asano et al. [4] improved the space bound to $\tilde{O}(n^{1/2})$ for planar graphs. Recently, this bound has been improved to $O(n^{1/4+\varepsilon})$ for grid graphs [24]. Besides, Chakraborty et al. [10] studied REACHABILITY for graphs with higher genus and gave an $\tilde{O}(n^{2/3}g^{1/3})$-space bound algorithm, and an $\tilde{O}(n^{2/3})$ space algorithm for H minor-free graphs.

For layered planar graphs, Chakraborty and Tewari [11] showed that there is an $O(n^{\varepsilon})$-space and polynomial algorithm. Gupta et al. [20] showed that given a pair $\{s, t\}$ and an embedding of an $O(\log n)$ genus graphs, REACHABILITY from s to t in G can be decided *unambiguously* in logspace (i.e. REACHABILITY is in the class UL).

Tree-decomposition and associated *treewidth* is an important notion of the graphs. Many problems which are computationally hard on general graphs are efficiently solvable on graphs of bounded Treewidth [1]. Graphs of small Treewidth also have small *vertex-separators*, which is a small set of vertices of the graph, removal of which divides the graph into pieces whose size are at most a fraction of the original graph. Recently, Jain and Tewari [25] showed that given an n vertex directed graph of treewidth w along with its tree decomposition, there exists an algorithm for RECHABILITY problem that runs polynomial time and $O(w \log n)$ space. They achieved this result by using the vertex separator of small Treewidth graphs. In fact, they formalized the connection between *Vertex Separator* and REACHABILITY problems, which has several consequences. For the sake of completeness, we state their result below.

▶ **Theorem 1** ([25]). *Let $\mathcal{G}$ be a class of directed graphs and $w : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ be a function. If there exist an $O(w(n, m) \log n)$ space and polynomial time algorithm, that given a graph $G \in \mathcal{G}$ of n vertices and m edges, and a set U of $V(G)$, outputs a separator of U in underlying undirected graph of G^1 of size $O(w(n, m))$, then there exists an algorithm to decide REACHABILITY in G that uses $O(w(n, m) \log n)$ space and polynomial time.*

In this paper, we study the REACHABILITY problem on directed geometric graph families. Many important graph families can be described as intersection graphs of more restricted types of set families. More often than not, the geometric intersection graph families provide additional geometric structures that help to generate efficient algorithms. Many problems that are NP-complete on general graphs are tractable on geometric intersection graphs [13, 22]. Indeed, such advantages have been exploited for space-efficient algorithms as well. We refer to the survey on geometric algorithms with limited work-space [5].

In this work, we make progress towards answering the question raised by Wigderson in his survey [33]. We primarily design and use the *Vertex separator* of some geometric graphs space-efficiently. Previously, vertex separators in the context of geometric graphs have been studied by Fox and Pach [16], who established several geometric extensions of the famous Lipton-Tarjan separator theorem [27]. Recently, Carmi et al. [8], and Hoffmann et al. [21] established improved bounds on the size of separators for some restricted classes of geometric graphs.

1.1   Our Contribution

We study the REACHABILITY problem on three different graph families – intersection Graphs of Jordan Regions, Unit Contact Disk Graphs (Penny Graphs) and Chordal Graphs.

---

[1] For a directed graph G, its *underlying undirected graph* is the graph formed by removing the orientation of all the edges.

First, in Section 2, we show that given a collection of Jordan regions, there exists a polynomial time algorithm that computes a separator of size $O(m^{1/2})$ using $O(m^{1/2} \log n)$ space, where n is the number of Jordan regions and m is the total number of crossings among those regions. Then, by combining this with Theorem 1, we note that REACHABILITY on directed intersection graphs of Jordan regions can be solved in polynomial time and $O(m^{1/2} \log n)$ space.

In Section 3, we present a space-efficient algorithm for REACHABILITY on penny graphs that uses $O(n^{1/4+\varepsilon})$ space and polynomial time. Since penny graphs are a subclass of planar graphs, REACHABILITY can be solved for penny graphs in $O(n^{1/2} \log n)$ space. However, to reduce the space complexity, we use an involved technique. First, we use the axis-parallel separator of Carmi et al. [8] repeatedly to form rectangular subdivision such that each cell of a subdivision contains a bounded size subgraph of the input graph. Then using these subdivisions, we construct a smaller auxiliary graph that preserves REACHABILITY information. Finally, using a notion of *pseduo-separator*, we solve REACHABILITY in this auxiliary graph in a space-efficient manner. Note that, there exists a a $O(n^{1/4+\varepsilon})$ space and polynomial time algorithm for reachability in grid graphs [24]. Since grid graphs are a subclass of penny graphs, our result is a generalization and improvement of previously known results.

Finally, in Section 4, we adopt the algorithm of [19] for REACHABILITY on chordal graphs to provide a space-efficient and polynomial time algorithm.

Even though chordal graphs seem to be an exception in the context of geometric graph families, we note that there exists work representing chordal graphs as a subfamily of string graphs [12]. Moreover, it is known that chordal graphs can be characterized as intersection graphs of sub-trees of a tree [18], and interval graphs are a subfamily of chordal graphs.

1.2 Preliminaries

Throughout the text, we denote the set $\{1, 2, \ldots, n\}$ as $[n]$. For a graph $G = (V, E)$ and a subset $U \subseteq V$, $G[U]$ denotes the subgraph induced on U. Given a graph $G = (V, E)$ with vertex set V and edge set E, a weight function $w : V \to R_{\geq 0}$ is a non-negative function on V such that the sum of the weights is 1. For any subset $S \subset V$, the weight $w(S)$ is defined to be $\sum_{v \in S} w(v)$. A separator in a graph G with respect to a weight function w is a subset $S \subset V$ of vertices such that there is a partition $V = S \cup V_1 \cup V_2$ such that $w(V_1), w(V_2) \leq 2/3$ and there are no edges between V_1 and V_2. If the weight function is not specified, it is assumed that $w(v) = \frac{1}{|V|}$, for every vertex v. We refer to Arora and Barak [2] for a basic understanding of the model and terminologies for space-efficient algorithms.

2 Intersection Graphs of Jordan Regions

In this section, we study the REACHABILITY problem on the intersection graphs of Jordan regions. Let $\mathcal{C}$ be a set such that each element C of $\mathcal{C}$ is a simply connected compact region in a plane bounded by a closed Jordan curve. Let $G(\mathcal{C})$ be an intersection graph on $\mathcal{C}$, where two distinct elements $C_1, C_2 \in \mathcal{C}$ are adjacent if and only if their intersection is not empty. Additionally, each edge of $G(\mathcal{C})$ is a directed edge. A Jordan region A *contains* another Jordan region B if $A \subseteq int(B)$, where $int(B)$ is the interior of B. We assume that no point is a boundary point of three elements of $\mathcal{C}$, and each element of $\mathcal{C}$ intersects at least one other element of $\mathcal{C}$. Further, we may assume that the number of such intersection points is finite. In [16], Fox and Pach showed the existence of a $O(m^{1/2})$-size separator on a collection of Jordan regions, where m is the total number of crossings points in the boundary of Jordan regions.
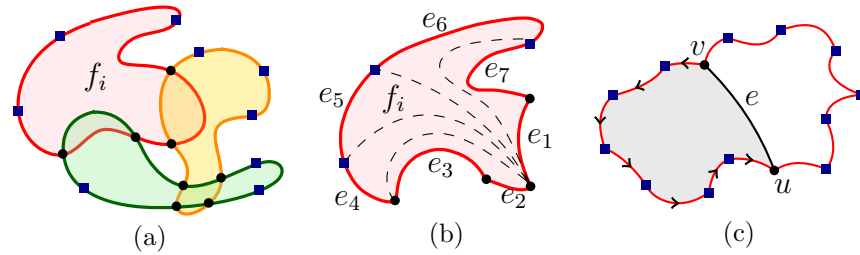
Figure 1 (a) An illustration of the planar embedding of the curves in $\mathcal{C}$. The black disk points are the intersection points from set $A(\mathcal{C})$, and the blue square points the three extra points from set $B(\mathcal{C})$. (b) An illustration of a face in the embedding and its corresponding planar triangulation. (c) Traversal of the left-face of the edge $e = (u, v)$.

For the purpose of computation, we work with those classes of Jordan regions that can be represented compactly, i.e., the set of n Jordan regions can be represented by $\text{poly}(n)$ bits. Furthermore, we assume that basic operations, such as determining the intersection point of two Jordan regions and outputting a constant number of points of given Jordan regions, can be performed in log-space. We prove the following theorem about the separator. Note that any *vertex separator* of an intersection graph does not rely on the direction of the edges. Hence for ease of explanation, we drop the directions from the input graph $G(\mathcal{C})$.

▶ **Theorem 2.** *Let $\mathcal{C}$ be a collection of Jordan regions, and let $G(\mathcal{C})$ be an intersection graph on $\mathcal{C}$. Let w be a weight function on $\mathcal{C}$. There exists a polynomial time algorithm that takes as an input the set $\mathcal{C}$ and outputs a separator of $G(\mathcal{C})$ with respect to w, of size $O(m^{1/2})$ using $O(m^{1/2} \log n)$ space.*

We assume that the sum of the weights of the given Jordan regions is one. Let $H(\mathcal{C})$ be the set of *heavy* regions in $\mathcal{C}$ whose weight is more than $1/m^{1/2}$, and let $L(\mathcal{C})$ be set of regions in $\mathcal{C}$ that are involved[2] in at least $\frac{1}{3}m^{1/2}$ containments with other elements of $\mathcal{C}$. Let $I(\mathcal{C}) = \mathcal{C} \setminus (H(\mathcal{C}) \cup L(\mathcal{C}))$. We define a planar graph $G_P(\mathcal{C})$. The vertex set of this graph is the union of two subsets, i.e., $A(\mathcal{C}) \cup B(\mathcal{C})$, where $A(\mathcal{C})$ is the set of all intersection points that lie on the boundary of at least one element of $I(\mathcal{C})$ and $B(\mathcal{C})$ is a collection of points not in $A(\mathcal{C})$ such that the boundary of each $C \in I(\mathcal{C})$ contains precisely three points in $B(\mathcal{C})$. There exists an edge between two vertices of $G_P(\mathcal{C})$ if and only if there are consecutive points along the boundary of an element of $I(\mathcal{C})$; see Figure 1(a).

To see that the defined graph is planar, we see that we have added a vertex at every point of intersection of the boundary of the given Jordan curves. We can now draw the edges along the boundary. We proceed with the following lemmas.

▶ **Lemma 3.** *There exists a log-space algorithm that takes as an input a set of Jordan regions $\mathcal{C}$ and outputs a graph $G_P(\mathcal{C})$.*

Proof. First of all, note that there exists a log-space subroutine that on an input Jordan region C of $\mathcal{C}$ determines if C is in $L(\mathcal{C})$. To do this, the subroutine iterates over all Jordan regions in $\mathcal{C}$ and calculates the total number of containment with which C is involved. The weight of each Jordan region comes with the input. Hence it is enough for us to determine if a Jordan region C of $\mathcal{C}$ belongs to the set $I(\mathcal{C})$. In order to construct the planar graph, we essentially need to know $I(\mathcal{C})$. Thereby concluding the proof of the Lemma.   ◀

---

[2] We say that a region C_1 is involved in a *containment* with a region C_2 if either C_1 contains C_2 or C_2 contains C_1.

Next, we triangulate $G_P(\mathcal{C})$, and denote the triangulated planar graph by $G_T(\mathcal{C})$. This triangulation can be obtained in log-space using the following lemma

▶ **Lemma 4.** *There exists an algorithm that takes a planar graph in the input and returns a triangulated planar graph in log-space.*

Proof. Consider a planar graph G_P. The embedding of a planar graph can be computed in log-space [15]. Note that an edge $e_1 = (u_1, v_1)$ in G_P is part of two faces in the planar embedding of the graph. We call them the left and the right face, respectively. We explain how to traverse the left face of e_1 in log-space. In order to do this, we start by traversing e_1 to one of its endpoint (say v_1) and take the edge clockwise next to e_1, that is incident on v_1. Let e_2 be such an edge; see Figure 1. We continue the traversal along the edge e_3, which is the edge clockwise next to e_2 from the endpoint v_2 of e_2, and so on. In general, if we reach at the vertex v_i using the edge e_i, we continue along the edge clockwise next to e_i from v_i. Clearly, by following this procedure, we can traverse the boundary of the face. In order to prove this lemma, it is sufficient to show that given two vertices u_l and v_l of the input planar graph G_P, whether an edge e_l can be added between them in log-space as part of the triangulation. We assume that the vertices of the input graph G_P are indexed by an integer from 1 to k, for some $k \in [n]$. For each edge e that is incident with u, we first traverse the left face of e and see if (i) v is present in that face, (ii) either u or v is the lowest-indexed vertex of that face. There is a triangulated edge between u and v if and only if both these conditions are true for any such edge e; see Figure 1(b). This concludes the proof of the lemma. ◀

Let $d(C)$ be the number of points on the boundary of C that belong to the vertex set of $G_T(\mathcal{C})$. For a vertex v in $G_T(\mathcal{C})$, we define a new weight function $\mathsf{weight}(v)$ as follows:
Case 1: v is part of two boundaries C_1 and C_2 of $\mathcal{C}$, $\mathsf{weight}(v) = \frac{w(C_1)}{d(C_1)} + \frac{w(C_2)}{d(C_2)}$.
Case 2: v is in the boundary of only one element C_1 of $\mathcal{C}$, $\mathsf{weight}(v) = \frac{w(C_1)}{d(C_1)}$.

Fox and Pach [16] used the idea to find a cycle-separator in this triangulated graph $G_T(\mathcal{C})$ and used it to construct a separator of the original geometric intersection graph. Instead, we will use the result of Imai et al. [23] to obtain such a separator in a space-efficient manner.

▶ **Lemma 5** ([23]). *Let G be a triangulated planar graph. There exists a polynomial time algorithm that uses $O(\sqrt{n}\log n)$ space to output a separator of G of size $O(\sqrt{n})$.*

Now, consider the set of regions of $\mathcal{C}$ whose boundary contains at least one of the points of the separator returned by the algorithm of Imai et al. on $G_T(\mathcal{C})$. We denote these regions by $\mathsf{sep}(\mathcal{C})$. In the following, we show that this set is indeed the required separator. Moreover, this set can be calculated within the required space-time bounds.

▶ **Lemma 6.** *The set $\mathsf{sep}(\mathcal{C})$ is a separator of the intersection graph of $\mathcal{C}$*

Proof. First of all, note that $G_T(\mathcal{C})$ is a triangulated planar graph. Let S be the separator of this graph returned by the algorithm of Imai et al. [23]. For a triangulated graph, we observe that the separator S is a cycle. Let V_0 be the set of all elements of $\mathcal{C}$ that are either not in $I(\mathcal{C})$, or whose boundary contains a vertex of S. Let K_1 and K_2 be the set of vertices that are *inside* and *outside* the cycle S, respectively. Let V_i (for $i \in \{1, 2\}$) be the set of elements of $I(\mathcal{C})$, such that all vertex of $V(G_T(\mathcal{C}))$ which are on its boundary belong to the component K_i. It is easy to see that V_0, V_1, and V_2 are pairwise disjoint sets, and their union

is $\mathcal{C}$. If V_0 has a weight of at least $1/3$, then we see that it act as a trivial separator. Hence, for the rest of this proof, we assume that the weight of V_0 is less than $1/3$. Thus, we can also see that the weight of V_i is at most $2/3$. It only remains to show that there is no edge between a vertex of V_1 and a vertex of V_2.

Let us assume, w.l.o.g., that the weight of V_2 is greater than the weight of V_1 and hence greater than $1/3$. In order to show that there is no edge between a vertex of V_1 and a vertex of V_2, let us assume to the contrary that there is one such edge. Since V_1 and V_2 are on two different sides of a closed Jordan curve, there exists an element v of V_1 that contains in its interior all the elements of V_2. The number of elements in V_2 is at least $\frac{1}{3}m^{1/2}$ contradicting the fact that v belongs to $I(\mathcal{C})$. ◀

Next, in the final lemma, we argue about the space-complexity of the above mentioned procedure.

▶ **Lemma 7.** *There exists a polynomial time algorithm that takes as an input $\mathcal{C}$ and outputs the set $\mathsf{sep}(\mathcal{C})$ in $O(m^{1/2} \log n)$ space.*

Proof. We first see that on input $\mathcal{C}$, the graph $G_T(\mathcal{C})$ contains $O(m)$ vertices. We can output this graph in log-space (see Lemma 4). Then, using the planar separator algorithm of Imai et al. [23], we can get a set of $O(m^{1/2})$ vertices which acts as the separator of this graph. This process can be done in $O(m^{1/2} \log n)$ space and polynomial time. Once we obtain this set, it is possible to construct $\mathsf{sep}(\mathcal{C})$ by using this set. ◀

This completes the proof of Theorem 2. Then, by combining Theorem 2 and Theorem 1, we conclude the following.

▶ **Corollary 8.** *There exists an algorithm that solves the REACHABILITY on the directed intersection graph of Jordan regions in polynomial time and $O(m^{1/2} \log n)$ space.*

3   Unit Contact Disk Graphs (Penny Graphs)

We now study the REACHABILITY problem on directed unit contact disk graphs (penny graphs). Penny graphs are also known as unit coin graphs [9]. Let $G = (D, E)$ be a directed penny graph, where $D = \{d_1, \ldots, d_n\}$ is a set of unit disks, and there is an edge $e \in E$ between two disks d_i and d_j (for some $i, j \in [n]$) if d_i and d_j touch each other. Moreover, each edge in $e \in E$ is a directed edge.

We prove the following theorem.

▶ **Theorem 9.** *For every $\varepsilon > 0$, there exists a polynomial time algorithm that can solve REACHABILITY in an n vertex directed penny graph, using $O(n^{1/4+\varepsilon})$ space.*

We know that given a set of n unit disks with m intersections, there exists an axis-parallel line intersecting $O(\sqrt{m+n})$ disks such that each half-plane separated by that line contains at most $4n/5$ disks [8]. They call such a separator a *balanced separator*. We first describe that how such a balanced separator can be obtained in a space-efficient manner.

Let R be a rectangular bounding box that contains the disks in D. Our algorithm to solve the REACHABILITY problem on penny graphs consists of four steps: (1) we find the balanced separators and use them to obtain a rectangular subdivision of the plane, (2) create an auxiliary graph, (3) obtain a pseudo-separator, (4) solve the REACHABILITY.

3.1 Obtaining Rectangular Subdivion

In this step, the direction of the edges of G is not relevant, so we simply consider the input as a set D of unit non-overlapping unit disks. The main idea is to divide the rectangle R into smaller rectangles such that each smaller rectangle contains at most $n^{1-\varepsilon}$ unit disks, and the number of unit disks intersecting the boundary of any rectangle is at most $n^{1/2+\varepsilon/2}$.

In what follows, we describe a sweeping procedure. We discuss primarily in terms of vertical sweeping, and the the horizontal sweeping is done similarly.

We consider the disks in sorted order based on the x-coordinates of their corresponding centers, which can be easily done in log-space. Consider a vertical sweep line ℓ. We start with the leftmost disk from D and sweep until we find a balanced separator that is intersecting $O(\sqrt{m+n})$ many disks such that each half-plane separated by that line contains at most $4n/5$ disks. If we do not find any such vertical separator, then we apply the same procedure with a horizontal sweep line (say ℓ') sweeping from top to bottom. From Theorem 2 in [8], we know that there exists an axis-parallel balanced separator. Therefore, we shall obtain a balanced separator by doing this procedure.

When the sweep line ℓ encounters or leaves a disk, we call it an event. There are precisely $2n$ number of *events*. Note that it is possible to test by using $\log n$ space whether a disk d_i intersects the line ℓ. We maintain a counter c_ℓ corresponding to the sweep line ℓ. At each event k (for some $k \in [n]$), we determine the number of disks that intersect ℓ, by checking each disk that whether it is intersecting with ℓ or not in log-space. Then, when the next event happens, we increase the value of the counter by 1 if it intersects a new disk, or decrease it by 1, otherwise. By using this procedure, we can find a separator line ℓ, that is, a balanced separator. It is also clear that we can determine such a separator by using $\log n$ space. Once we find such a separator, we only store the x-coordinate (resp. y-coordinate) of the vertical line ℓ (resp. horizontal line ℓ'). Note that it is possible to find the actual set of disks that form such a separator in log-space when needed.

We subdivide the rectangles repeatedly until each of the rectangles has smaller than $n^{1-\varepsilon}$ disks. Initially, we have the rectangle R containing all the disks. Let $\mathcal{R}_0 = \{R\}$ be the initial set of rectangles. After step i, we have the rectangles $\mathcal{R}_i$ be the set of rectangles, and We pick the rectangle with more than $n^{1-\varepsilon}$ disks and subdivide it further using the above process to get $\mathcal{R}_{i+1}$. See Figure 2 for an illustration.
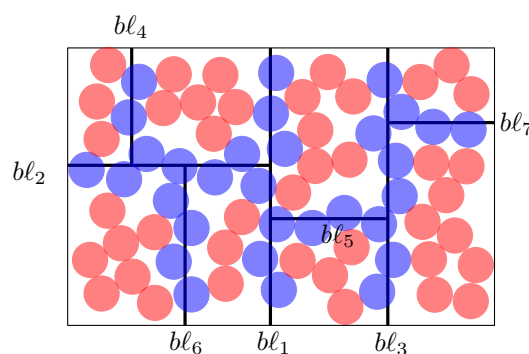
Figure 2 An illustration of the rectangular subdivision by using the balanced separators. The blue disks are the ones intersected by the balanced separator lines, and the red disks are contained inside the rectangles.

We calculate the number of separators required to reach this termination point. From [8], we know that one $\sqrt{m+n}$ size separator guarantees that on each side there are at most $\frac{4n}{5}$ disks. Since the class of penny graphs is a subclass of planar graphs, the total number of

edges is at most $3n - 6$. Now at each step, we have obtained a *balanced separator* whenever it has satisfied the criteria. We need $O(n^\varepsilon)$ many separators to have at most $O(n^{1-\varepsilon})$ disks in each cell. To store these line separators, we need to use $O(n^\varepsilon \log n)$ space.

The initial graph G is divided into $n^\varepsilon \times n^\varepsilon$ rectangles obtained from the above procedure. Let $\mathcal{Z}$ be the set of all rectangles. The idea is to reduce the size of the graph G by dropping the disks that are completely contained inside some rectangle and are not touched or intersected by its boundary line. However, while reducing the size of the number of disks, we need to ensure that the REACHABILITY information is fully preserved between any pair of disks in G. For that, we proceed to the next step and build an auxiliary graph.

3.2 Building Auxiliary Graph

For a rectangle R, let G_R be the graph defined as follows. The vertex set of G_R is the set of all disks which intersect at least one of the boundaries of R. We add an edge from a vertex u to a vertex v in G_R if there is a directed path from u to v, which contains only the disks present inside the rectangle R. Let v_1 be an arbitrary disk, and let $\{v_1, \ldots, v_k\}$ be the sets of disks intersecting the boundary of R in the anti-clockwise order. We place the disk centers on the boundary while preserving (1) the order of them on the boundary, (2) each vertex v_i is on the side of the boundary that it intersects. However, if it is intersected by more than one side (one vertical and one horizontal) then we create an additional dummy vertex (say v_i') and assign v_i and v_i' to the horizontal and vertical side, respectively. Moreover, we add a bidirectional edge between v_i and v_i'. see Figure 3(b)) for an illustration.

The edges of G_R are drawn in the following manner. If there is an edge between two vertices that are on different sides, then we give a directed straight line edge. Otherwise, we join them by a directed circular arc. Moreover, we ensure that there is a crossing between two edges (v_i, v_k) and (v_l, v_j) in the drawing if and only if there is an ordering on the boundary which is one of the followings - 1. $\{v_i \prec v_l \prec v_k \prec v_j\}$, 2. $\{v_j \prec v_l \prec v_k \prec v_i\}$, 3. $\{v_l \prec v_i \prec v_j \prec v_k\}$, 4. $\{v_l \prec v_i \prec v_j \prec v_k\}$. Clearly, there exists such a drawing as their arc edges can be drawn arbitrarily close to the boundary lines.
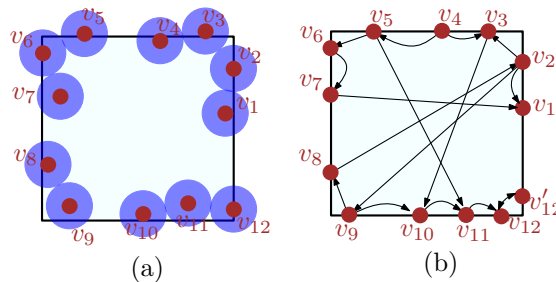

Figure 3 An illustration of the representation of the graph G_R, (a) the blue disks are the vertices of G_R intersecting the boundary of the rectangle (b) drawing of G_R.

Now, by combining the graphs defined for each rectangle, we define the auxiliary graph $Aux_\varepsilon(G)$, for $0 < \varepsilon < 1$. The vertex set of $Aux_\varepsilon(G)$ is $\bigcup_{R \in \mathcal{Z}} V(G_R)$ and the edge set is $\bigcup_{R \in \mathcal{Z}} E(G_R)$. Notice that $Aux_\varepsilon(G)$ might have parallel edges, since there exists paths between vertices in two adjacent rectangles, and in that case we keep both of these edges in their respective rectangles. The total number of vertices in each cell is $O(n^{1-\varepsilon})$. Hence the total number of vertices in $Aux_\varepsilon(G)$ is $O(n^{1/2+\varepsilon/2})$. We point out that we do not store $Aux_\varepsilon(G)$ explicitly because that requires too much space. Instead, we deal with each cell recursively when the subroutine queries for an edge in that cell of $Aux_\varepsilon(G)$. Now, we prove the following property about the auxiliary graph.

▶ **Lemma 10.** *Let G be a penny graph and $e = (v_i, v_j)$ and $e' = (v_k, v_l)$ be two edges in $Aux_\varepsilon(G)$. If e and e' cross each other, then $Aux_\varepsilon(G)$ also contains the edges (v_i, v_l) and (v_k, v_j).*

Proof. Consider two edges $e = (v_i, v_j)$ and $e' = (v_k, v_l)$ be two edges in $Aux_\varepsilon(G)$. From the definition, these edges corresponding to the directed paths in the input graph G. From the construction, we know that the ordering of the end vertices are one of the following - $\{v_i \prec v_l \prec v_k \prec v_j\}$, $\{v_j \prec v_l \prec v_k \prec v_i\}$, $\{v_l \prec v_i \prec v_j \prec v_k\}$, $\{v_l \prec v_i \prec v_j \prec v_k\}$. Now, we know that the corresponding directed paths are fully embedded inside the grid cell. If two directed path intersects in the Auxiliary graph (based on our definition) of an input penny graph, that is embedded inside a rectangle, then they must have one common vertex; see Figure 4. Otherwise, it will not admit a planar embedding. Hence, there must be directed path from v_i to v_l, and v_k to v_j. Thereby proving the lemma. ◀
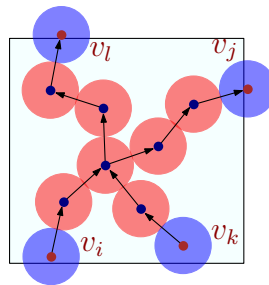
Figure 4 An illustration of two directed paths inside a grid cell.

3.3 Constructing Pseudo-Separator

An essential property of a vertex separator is that, for any two vertices u and v, a path between them must contain a vertex from the separator if u and v lie in two different components with respect to the separator. We use a separator construction for the auxiliary graph $Aux_\varepsilon(G)$. However, note that $Aux_\varepsilon(G)$ is not a planar graph anymore. Therefore, we need a special kind of separator, which we call a *pseudo-separator*.

The notion of the pseudo-separator was introduced by Jain and Tewari [24] in the context of grid graphs. However, since the class of Penny graphs is a superclass of grid graphs, it is not possible to use their idea directly.

Let G be a penny graph and $H = (V_1, E_1)$ be a vertex induced subgraph of $Aux_\varepsilon(G)$ with h vertices. Let $f : \mathbb{N} \to \mathbb{N}$ be a function. A subgraph $C = (V_2, E_2)$ of H is said to be an $f(h)$-PseudoSeparator of H, if the size of every connected component in $H \cdot C$ is at most $f(h)$, where the graph $H \cdot C = (V_3, E_3)$ is defined as $V_3 = V_1 \setminus V_2$ and $E_3 = E_1 \setminus \{e \in E_1 \mid \exists e' \in E_2, e \text{ crosses } e'\}$. The size of C is the total number of vertices and edges of C summed together.

The general idea of our approach is the following. Consider a vertex induced subgraph H of $Aux_\varepsilon(G)$. We choose a maximal subset of edges such that H is a planar graph, thus admits a planar embedding. Next, we triangulate this chosen sub-graph. We show that until this point, each operation can be performed in log-space. Then, we use the algorithm of Imai et al. [23] to obtain a separator of the triangulated graph. In what follows, we describe these procedures in detail.

We start with a maximal planar graph $H(Aux_\varepsilon(G))$ of $Aux_\varepsilon(G)$. The vertex set of $H(Aux_\varepsilon(G))$ is same as the vertex set of $Aux_\varepsilon(G)$. For each rectangle, we index the vertices in ascending order while traversing them in anti-clockwise direction. For each rectangle R, an

edge $e_l = (u_l, v_l)$ of G_R is added to $H(Aux_\varepsilon(G))$ if there is no edge $e_k = (u_k, v_k)$ such that the ordering of the vertices is one of the following - $\{u_k \prec u_l \prec v_k \prec v_l\}$, $\{v_k \prec u_l \prec u_k \prec v_l\}$, $\{u_k \prec v_l \prec v_k \prec u_l\}$, $\{u_k \prec v_l \prec v_k \prec u_l\}$. We prove that, $H(Aux_\varepsilon(G))$ is indeed a maximal planar graph of $Aux_\varepsilon(G)$.

▶ **Lemma 11.** $H(Aux_\varepsilon(G))$ *is a maximal planar graph of* $Aux_\varepsilon(G)$.

Proof of Lemma 11 can be found in the full version [7].

Next, we triangulate $H(Aux_\varepsilon(G))$ by adding the boundary edges. Then for each rectangle, we consider a face of $H(Aux_\varepsilon(G))$ and add the edges to complete the triangulation by a similar procedure described in Lemma 4 from Section 2. Moreover, the direction of the edges are arbitrary. Let $\hat{H}(Aux_\varepsilon(G))$ be the triangulated graph. Note that this procedure can be done in log-space. Next, we use a Lemma from [23] that is stated below.

▶ **Lemma 12** ([23]). *For each $\beta > 0$, there exists a polynomial time algorithm and $\tilde{O}(h^{1/2+\beta/2})$ space algorithm that takes a h-vertex planar graph P as input and and outputs a set of vertices S, such that $|S|$ is $O(h^{1/2+\beta/2})$ and removal of S disconnects the graph into components of size $O(h^{1-\beta})$.*

Next, we construct the pseudo-separator $S_H(Aux_\varepsilon(G))$ by using the following steps. First, by Lemma 12 we find a set S in $\hat{H}(Aux_\varepsilon(G))$ that divides it into components of size $O(h^{1-\beta})$, where h is the number vertices in $\hat{H}(Aux_\varepsilon(G))$. We add the vertices and edges of S to the vertex and edge set of $S_H(Aux_\varepsilon(G))$, respectively. However, there is a small caveat to use Lemma 12 on $\hat{H}(Aux_\varepsilon(G))$. In order to triangulate the graph, we have added edges that were originally not part of the auxiliary graph. Therefore, for each edge $e_k = (u_k, v_k)$ of the triangulation that is present in some rectangle $\mathcal{R}$, we consider a set of at most four edges of $Aux_\varepsilon(G)$ that form a so-called *shield* around the edge e_k. Two of these edges start from u_k ending at two vertices v_p, v_p', where v_p and v_p' are the closest point to the left and closest point to the right of u_k, respectively, in the total ordering of the boundary vertices. The other two edges start from v_k and end at two vertices u_q, u_q', where u_q and u_q' are the closest points to the right and the left of v_k respectively, in the total ordering of boundary vertices. See Figure 5(a) for an illustration. Later, we argue that if these edges do not exist, and the cycle separator intersects the corresponding triangulation edge, then there must be other edges chosen in the maximal planar graph intersecting that triangulation edge, and hence this gives a contradiction.

In order to prove that $S_H(Aux_\varepsilon(G))$ is indeed a pseudo-separator, we need a property of triangulated graphs from [25].

▶ **Lemma 13** ([25]). *Let G be a triangulated planar graph and S be a subset of its vertices. For every pair of vertex u, v which belong to different components of $G \setminus S$, there exists a cycle in $G[S]$, such that u and v belong to different sides of this cycle.*

Next, we prove the Lemma about pseudo-separator.

▶ **Lemma 14** (Pseudo-Separator Lemma). *Let G be a penny graph and $H(Aux_\varepsilon(G))$ be a vertex induced subgraph of $Aux_\varepsilon(G)$. The graph $S_H(Aux_\varepsilon(G))$ is a $h^{1-\beta}$-pseudo-separator of $H(Aux_\varepsilon(G))$.*

Proof. Let S be a set of vertices chosen from $\hat{H}(Aux_\varepsilon(G))$ by using Lemma 12. The claim is that if two vertices u and v belong to two different components in $\hat{H}(Aux_\varepsilon(G)) \setminus S$, then a path between u and v in G either takes a vertex of $S_H(Aux_\varepsilon(G))$ or crosses an edge of
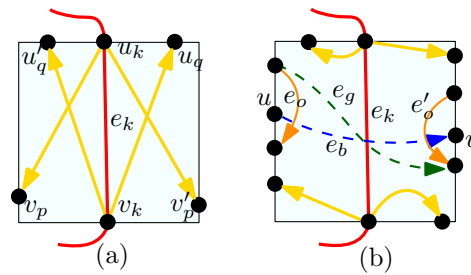
Figure 5 An illustration of the pseudo-separator. The red edge is an edge of the triangulation and part of the red cycle separator. The yellow edges are chosen to form the shield (a) when the proper shield exists (b) otherwise.

$S_H(Aux_\varepsilon(G))$. Note that due to Lemma 13, we know that u and v are on two different sides of a cycle of S. Thus, any edge drawn in the plane connecting u and v (not necessarily a straight line edge) crosses the cycle of S. If the path crosses non-triangulation edge or an edge which is shielded properly (see Figure 5(a)), we are done as the path will intersect an edge of $S_H(Aux_\varepsilon(G))$. The other situation is when such a path intersects a triangulation edge. We prove by contradiction that this can not happen. Assume that it does happen (see Figure 5(b)). Let e_r be the triangulation edge intersected by the edge e_b of the path from u to v. Now, since e_b is not chosen in the maximal planar graph, this means there exist other edges chosen in the $H(Aux_\varepsilon(G))$. Let e_o and e'_o be two such edges with maximum width on either side of the triangulated edge (see the orange edges in Figure 5(b)). By Lemma 10, there exists an edge e_g from the tail of e_o to the head of e'_o (the green path in Figure 5(b)). Since this edge e_g also crosses the triangulation edge e_r, it is not present in the maximal planar graph. Any edge that is present in maximal planar graph and crosses e_g contradicts the fact that e_o and e'_o are of maximum width. There are other cases when either e_o or e'_o is not present, but they can be handled similarly. Thereby concluding the proof of the Lemma. ◀

3.4   The Algorithm

Let H be a vertex induced subgraph of an auxiliary graph. We first explain how to solve REACHABILITY in H. Initially, H is the whole auxiliary graph, and we wish to find the REACHABILITY between given two vertices s and t of H.

By using Lemma 14, we find a pseudo-separator S of H. W.l.o.g., assume that s and t are both in S. The pseudo-separator divides the graph into components $C_1, C_2, \ldots, C_k$, for $k \in [h]$. We maintain a set of vertices M that we call *marked* vertex set. We use an array of size at most $|S|$ to mark a set of vertices M_1 of the pseudo-separator. Additionally, for each edge of the pseudo-separator we have at most one associated vertex, and they form the set M_2. Then, $M = M_1 \cup M_2$ is called a *marked* vertex set.

Throughout the algorithm, we maintain that if a vertex is marked, then there is a path from s to that vertex in the auxiliary graph. Initially, only the vertex s is marked. We then perform h iterations. In each iteration, we update the set of marked vertices as follows:

Step 1.   For every vertex of S, we mark it if there is a path from an already marked vertex to it such that the internal vertices of that path all belong to only one component C_i, for $i \in [h]$. We check this by recursively running our algorithm on the subgraph of H induced by the vertex set C_i.

Step 2. For each edge e of S, the algorithm sets the associated marked vertex to u if the following three conditions are satisfied: (a) There exists an edge $e' = (u, v)$ which crosses e, (b) there exists a path from a marked vertex to u such that the internal vertices of that path all belong to only one component C_i (we check this by recursively running our algorithm on the vertex induced subgraph of C_i), (c) e' is the closest such edge to u.

Finally, we output 'YES' if and only if t is marked at the end of these iterations.

3.5   Correctness

Let P be a path from s to t in H. Suppose P passes through the components $C_{\sigma_1}, C_{\sigma_2}, \ldots, C_{\sigma_l}$ (each σ_i is a value in $[k]$) in this order. By the definition of a pseudo-separator, the path can go from one component, C_{σ_i} to the next component $C_{\sigma_{i+1}}$ in the following ways only:

Case 1: The path exits C_{σ_i} and enters $C_{\sigma_{i+1}}$ through a vertex w of the pseudo-separator.

Case 2: The path exits C_{σ_i} and enters $C_{\sigma_{i+1}}$ through an edge $e' = (u, v)$ whose tail is in C_{σ_i} and head is in $C_{\sigma_{i+1}}$. This edge will cross an edge $e = (x, y)$ of the pseudo-separator.

We see that after the i-th iteration, our algorithm will traverse the fragment of the path in the component C_{σ_i} and either mark in **Case 1** its endpoint or in **Case 2** a vertex u' such that the edge (u', v) exist. Thus, t will be marked after l iterations if and only if there is a path from s to t in H. Since l can be at most h, these many iterations would suffice.

From the above discussion, we know how to solve REACHABILITY on the auxiliary graph with the desired space-bound. To solve the problem on the input graph, we find the rectangular subdivision of the input graph. Then, we form the auxiliary graph by solving each rectangle recursively. Next, we solve the REACHABILITY problem on the auxiliary graph by following the above procedure. We have shown earlier that the reachability information of the input graph is fully preserved in the auxiliary graph. Hence, it is possible to report the solution of the input directed penny graph from the solution on the auxiliary graph.

3.6   Space-Complexity

The space complexity of our algorithm is dominated by the space required to store the *marked* vertices. Since there can be only $h^{1/2+\varepsilon/2}$ such vertices, we need $O(h^{1/2+\varepsilon/2} \log n)$ space. It is easy to see that for every $\varepsilon' > 0$, there exists an $\varepsilon > 0$ such that $O(h^{1/2+\varepsilon/2} \log n) = O(h^{1/2+\varepsilon'})$ Note that, for the input auxiliary graph H, our algorithm recurses on vertex induced subgraphs whose size is $O(h^{1-\varepsilon})$. Hence, the depth of recursion is bounded by a constant. This increases the space required by at most a constant factor. Since the number of vertices in the initial auxiliary graph was itself $O(n^{1/2+\varepsilon/2})$, we get the desired space bound.

From the correctness and the space-complexity analysis of the algorithm, we conclude the proof of Theorem 9

4   Chordal Graphs

In this section, we study the REACHABILITY problem on directed chordal graphs and design a space-efficient algorithm. A graph is said to be chordal if every cycle of length at least four has a *chord*, which is an edge joining two vertices that are not adjacent on the cycle. A *directed* chordal graph is a graph whose underlying undirected graph is chordal. See [14, 17, 31] for the basic theory on chordal graphs. We adopt the algorithm of Gilbert et al. [19] for *Vertex Separator* and analyze to obtain the desired space-bound. In [25], it was noted that REACHABILITY on chordal graphs could be solved in a space-efficient manner, however, without any formal explanation. Here we provide a detailed analysis of this claim.

Let $G = (V, E)$ be a directed chordal graph. For the sake of completeness, we will state some of the known results that are relevant to us. Chordal graphs are also known as *triangulated graphs, monotone transitive graphs, rigid circuit graphs, perfect elimination graphs* in the literature.

Finding Separator

Gilbert et al. [19] presented $O(mn)$-time algorithm for finding *Vertex Separator* on chordal graphs. Furthermore, a better algorithm of time complexity $O(m)$ is also shown. We design a space-efficient algorithm for Vertex separator that uses $O(m^{1/2} \log n)$ space and polynomial time. We adopt the $O(mn)$-time algorithm from [19] and analyze it to provide the desired space-bound. In order to do that, we allow time complexity that is much larger than $O(mn)$ but remains polynomial in n. We proceed with the following lemma: A similar property is somewhat implicitly used by [19].

▶ **Lemma 15.** *Let G be a chordal graph and let C be a clique. Let A be the largest component in $G \setminus C$. Then, either of the following two statements is true:*
- *There exists a vertex v in C which is not adjacent to any vertex in A.*
- *There exists a vertex u in A which is adjacent to every vertex in C.*

We need several definitions and structural properties of the Chordal graphs in order to prove this Lemma. However, some of the properties are well-known results for chordal graphs. We proceed with the following definitions.

▶ **Definition 16.** *Let G be a graph and v be a vertex of G. The deficiency of v, denoted by $D(v)$ is defined as follows: $D(v) = \{\{u_1, u_2\} \mid \{u_1, v\} \in E(G), \{u_2, v\} \in E(G)$ and $\{u_1, u_2\} \notin E(G)\}$*

▶ **Definition 17.** *Let G be a graph and v be a vertex of G. We define the graph G_v as follows: $G_v = (V(G) \setminus \{v\}, E(G[V(G) \setminus \{v\}]) \cup D(v))$. We say that the graph G_v is formed by eliminating v from G.*

▶ **Definition 18.** *When a sequence of vertices is eliminated from a graph, the edges in the deficiencies that are added are called fill-in edges. A simplicial vertex of a graph is a vertex that has a deficiency of 0.*

We state the following known facts about the chordal graphs. The following Lemmas are due to [14, 17, 30, 29, 31]. For the sake of completion, we state them here in the form that we will be using and provide their proofs.

▶ **Lemma 19.** *Let G be a chordal graph and a and b be two vertices in $V(G)$. Let S be a set of vertices of G such that: 1) a and b are in different components of the graph $G \setminus S$; 2) there exists no proper subset $S' \subseteq S$ such that vertices a, b are in different components of the graph $G \setminus S'$. If these conditions hold, then the set S forms a clique in G.*

Proof. Let C_a and C_b be the components in $G \setminus S$ containing a and b, respectively. Note that, each vertex $s \in S$ is adjacent to some vertices in C_a, and some vertices in C_b. Consider two vertices $x, y \in S$. Let $\mathcal{P}_a$ be the shortest path between x and y in $G[C_a \cup \{x, y\}]$. Similarly, let $\mathcal{P}_b$ be the shortest path between x and y in $G[C_b \cup \{x, y\}]$. The paths $\mathcal{P}_a$ and $\mathcal{P}_b$ together forms a cycle. Hence, there must be an edge between x and y, since it is the only *chord* that is possible. This argument holds for any pair of vertices in S. This completes the proof. ◀

▶ **Lemma 20.** *Let G be a chordal graph and let C be any clique of G. Then, either G is a complete graph, or there is a vertex $v \in G \setminus C$ that is simplicial.*

Proof. We prove this by induction on $|G|$. The base case $|G| = 1$ is trivial. Let us assume that the Lemma holds for all G such that $|G| \leq k$, for some $k \in [n]$. Now, G be a graph such that $|G| = k + 1$. If G is not a complete graph then, let a, b be two vertices of G, that are not adjacent. Due to Lemma 19, there exists a set S that separates a and b. Let C_a, C_b, be the corresponding components G containing a and b, respectively. Note that, the vertices in $C \setminus S$ should be in one component. W.l.o.g., assume that they belong to C_a. Consider the graph $G_b = G[S \cup C_b]$. We have $|G_b| \leq k$. Hence, by induction, either G_b is a clique or there is a vertex $u \notin S$, that is *simplical* in G_b. In either of the cases, there exists a vertex $u \notin S$ that is simplicial in G_b since G_b must contain at least one vertex, not in S. Note, u is a simplicial vertex in G since u is not adjacent to a vertex in any component other than C_b. This completes the proof.   ◀

▶ **Corollary 21.** *Let G be a chordal graph and C be a clique. Let A be the largest component in $G \setminus C$. If B is a non-empty subset of A, then B contains a vertex whose neighbours in $G[B \cup C]$ forms a clique.*

▶ **Lemma 22** ([19]). *Let $a_0, a_1, \ldots, a_k$ be an elimination ordering for a graph G. Let v and w be nonadjacent vertices of G. Then $\{v, w\}$ is a fill-in edge if and only if there is a path from v to w consisting of vertices that are eliminated earlier than both v and w.*

Now, we have the ingredients to prove the main Lemma 15.

Proof. Let G be a chordal graph, and let C be a clique. Let A be the largest component in $G \setminus C$. Let us assume that each vertex $v \in C$ is adjacent to at least one vertex in A. We will show that under this assumption, there exists a vertex u in A that is adjacent to each vertex in C, thereby proving the Lemma.

It is known that a vertex induced subgraph of a chordal graph is also a chordal graph. Hence, the subgraph $G_0 := G[A \cup C]$ is a chordal graph. Due to Lemma 20, we know that there is a vertex $u_0 \in G_0 \setminus C$, that is simplicial, whose neighbors form a clique. Let $G_1 := G[(A \cup C) \setminus \{u_0\}]$. Similarly, let u_1 be a vertex in $G_1 \setminus C$ that is simplicial. In general, let u_i be a vertex in $G_i \setminus C$ whose neigbhours forms a clique, where G_i is defined as $G[(A \cup C) \setminus \{u_0, u_1, \ldots, u_{i-1}\}]$. Let k be an integer such that $G_{k+1} = G[C]$. We claim that u_k is adjacent to every vertex of C.

Consider a vertex x in C. Since A is connected and by our assumption, x is adjacent to a vertex of A, there is a path from x to u_k in $G[A \cup C]$ that uses only vertices of A. Lemma 22 says that if $\{x, u_k\}$ is not an edge of $G[A \cup C]$, then it is a fill-in edge. But a perfect elimination ordering has no fill-in, so x is adjacent to u_k in $G[A \cup C]$. Thus u_k is adjacent to every vertex of C. This completes the proof.   ◀

Given the above lemma, the algorithm for finding a separator in a chordal graph is rather straightforward, as also observed by [19]. The algorithm is described in the following Procedure 1.

The correctness of the algorithm directly comes from Lemma 15. We show that this algorithm uses $O(m^{1/2} \log n)$ space.

Space-Complexity

To see that the above algorithm can be implemented in $O(m^{1/2} \log n)$-space and polynomial time, we first recall that the separator S always forms a clique in the graph. Since the size of a clique is upper-bounded by $m^{1/2}$, it will require at most $O(m^{1/2} \log n)$ space to store the clique.

▬ **Algorithm 1** Vertex Separator in Chordal Graphs.

Input: A chordal graph G
Output: A separator S of G of size $\sqrt{m}$

1 $S \leftarrow \emptyset$;
2 while *there exists a component A of $G \setminus S$ that has weight more than $n/2$* **do**
3   **while** *there exists a vertex x of S that is adjacent to no vertex of A* **do**
4 $S \leftarrow S \setminus \{x\}$;
5   **end**
6 $v \leftarrow$ a vertex of A adjacent to every vertex of S;
7 $S \leftarrow S \cup \{v\}$;
8 end
9 Return S;

Now, to implement the algorithm, we need to calculate the weights of each of the connected components of $G \setminus S$. However, we cannot afford to store these components explicitly, since the number of vertices present in these components could be large. Instead, we identify a component with the lowest-index vertex present in it. We call the lowest-index vertex a *marker* of the component. In order to check that whether a vertex v is a *marker*, we run Reingold's undirected REACHABILITY algorithm to see if it is not connected with any vertex u in $G \setminus S$ such that the index of u is lower than v. Also, to know the size of the connected component for a marker vertex v, we use Reingold's algorithm [28] to count the number of vertices that are connected to it. Therefore it is possible to count the weight of any component of $G \setminus S$ in the desired space-bound. We conclude the following theorem.

▶ **Theorem 23.** *Given a chordal graph $G = (V, E)$, there exists an algorithm that computes a $\sqrt{m}$ separator in polynomial time by using $O(m^{1/2} \log n)$-space.*

From Theorem 23 and Theorem 1, we have the following corollary.

▶ **Corollary 24.** *There exists an algorithm that solves the REACHABILITY problem for chordal graphs by using $O(m^{1/2} \log n)$-space and polynomial time.*

5   Conclusion

We studied REACHABILITY problem on three important graph families and obtained space-efficient algorithms for each of these classes. An interesting open problem is whether one can obtain a space-efficient algorithm for intersection graphs of Jordan regions when the embedding of the graph is not provided in the input. Another important open problem is to study the REACHABILITY for unit disk intersection graphs and obtain a space-efficient algorithm. Our method that we used for unit contact disk graphs does not generalize to the case of unit disk intersection graphs, as in the latter case, there can be arbitrarily large directed cliques, and it is not possible to obtain auxiliary graphs while preserving reachability information between every pair of vertices. However, we believe that our method can be used to solve to REACHABILITY for other classes of geometric contact graphs.

───── **References** ─────

1   Stefan Arnborg and Andrzej Proskurowski. Linear time algorithms for np-hard problems restricted to partial k-trees. *Discrete applied mathematics*, 23(1):11–24, 1989.
2   Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

3   Tetsuo Asano and Benjamin Doerr. Memory-constrained algorithms for shortest path problem. In *CCCG*, 2011.

4   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.

5   Bahareh Banyassady, Matias Korman, and Wolfgang Mulzer. Geometric algorithms with limited workspace: A survey. *CoRR*, abs/1806.05868, 2018. `arXiv:1806.05868`.

6   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.

7   Sujoy Bhore and Rahul Jain. Space-efficient algorithms for reachability in geometric graphs. *CoRR*, abs/2101.05235, 2021. `arXiv:2101.05235`.

8   Paz Carmi, Man-Kwun Chiu, Matthew J. Katz, Matias Korman, Yoshio Okamoto, André van Renssen, Marcel Roeloffzen, Taichi Shiitada, and Shakhar Smorodinsky. Balanced line separators of unit disk graphs. *Comput. Geom.*, 86, 2020.

9   Marcia R Cerioli, Luerbio Faria, Talita O Ferreira, and Fábio Protti. A note on maximum independent sets and minimum clique partitions in unit disk graphs and penny graphs: complexity and approximation. *RAIRO-Theoretical Informatics and Applications*, 45(3):331–346, 2011.

10 Diptarka Chakraborty, Aduri Pavan, Raghunath Tewari, N. V. Vinodchandran, and Lin F. Yang. New time-space upperbounds for directed reachability in high-genus and h-minor-free graphs. In Venkatesh Raman and S. P. Suresh, editors, *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014, December 15-17, 2014, New Delhi, India*, volume 29 of *LIPIcs*, pages 585–595. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2014.

11 Diptarka Chakraborty and Raghunath Tewari. An $o(n^\epsilon)$ space and polynomial time algorithm for reachability in directed layered planar graphs. *ACM Trans. Comput. Theory*, 9(4):19:1–19:11, 2018.

12 Steven Chaplick, Vít Jelínek, Jan Kratochvíl, and Tomáš Vyskočil. Bend-bounded path intersection graphs: Sausages, noodles, and waffles on a grill. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 274–285. Springer, 2012.

13 Brent N Clark, Charles J Colbourn, and David S Johnson. Unit disk graphs. In *Annals of Discrete Mathematics*, volume 48, pages 165–177. Elsevier, 1991.

14 Gabriel Andrew Dirac. On rigid circuit graphs. In *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, volume 25, pages 71–76. Springer, 1961.

15 Michael Elberfeld and Ken-ichi Kawarabayashi. Embedding and canonizing graphs of bounded genus in logspace. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 383–392, 2014.

16 Jacob Fox and János Pach. Separator theorems and turán-type results for planar intersection graphs. *Advances in Mathematics*, 219(3):1070–1080, 2008.

17 Delbert Fulkerson and Oliver Gross. Incidence matrices and interval graphs. *Pacific journal of mathematics*, 15(3):835–855, 1965.

18 Fănică Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *Journal of Combinatorial Theory, Series B*, 16(1):47–56, 1974.

19 John R Gilbert, Donald J Rose, and Anders Edenbrandt. A separator theorem for chordal graphs. *SIAM Journal on Algebraic Discrete Methods*, 5(3):306–313, 1984.

20 Chetan Gupta, Vimal Raj Sharma, and Raghunath Tewari. Reachability in o (log n) genus graphs is in unambiguous logspace. In *36th International Symposium on Theoretical Aspects of Computer Science (STACS 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

21 Michael Hoffmann, Vincent Kusters, and Tillmann Miltzow. Halving balls in deterministic linear time. In *European Symposium on Algorithms*, pages 566–578. Springer, 2014.

22   Hiroshi Imai and Takao Asano. Finding the connected components and a maximum clique of an intersection graph of rectangles in the plane. *Journal of Algorithms*, 4(4):310–323, 1983.

23   Tatsuya Imai, Kotaro Nakagawa, Aduri Pavan, N. V. Vinodchandran, and Osamu Watanabe. An $o(n^{1/2} + \sum)$-space and polynomial-time algorithm for directed planar reachability. In *Proceedings of the 28th Conference on Computational Complexity, CCC 2013, K.lo Alto, California, USA, 5-7 June, 2013*, pages 277–286. IEEE Computer Society, 2013.

24   Rahul Jain and Raghunath Tewari. An $o(n^{(1/4+\epsilon)}$ space and polynomial algorithm for grid graph reachability. In Arkadev Chattopadhyay and Paul Gastin, editors, *39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2019, December 11-13, 2019, Bombay, India*, volume 150 of *LIPIcs*, pages 19:1–19:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019.

25   Rahul Jain and Raghunath Tewari. Reachability in high treewidth graphs. In Pinyan Lu and Guochuan Zhang, editors, *30th International Symposium on Algorithms and Computation, ISAAC 2019, December 8-11, 2019, Shanghai University of Finance and Economics, Shanghai, China*, volume 149 of *LIPIcs*, pages 12:1–12:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019.

26   Harry R Lewis and Christos H Papadimitriou. Symmetric space-bounded computation. *Theoretical Computer Science*, 19(2):161–187, 1982.

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

28   Omer Reingold. Undirected connectivity in log-space. *Journal of the ACM (JACM)*, 55(4):1–24, 2008.

29   Donald J Rose. Triangulated graphs and the elimination process. *Journal of Mathematical Analysis and Applications*, 32(3):597–609, 1970.

30   Donald J Rose. A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. In *Graph theory and computing*, pages 183–217. Elsevier, 1972.

31   Donald J Rose, R Endre Tarjan, and George S Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on computing*, 5(2):266–283, 1976.

32   Walter J Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of computer and system sciences*, 4(2):177–192, 1970.

33   Avi Wigderson. The complexity of graph connectivity. In *International Symposium on Mathematical Foundations of Computer Science*, pages 112–132. Springer, 1992.