An $O(n^\epsilon)$ Space and Polynomial Time Algorithm for Reachability in Directed Layered Planar Graphs

DIPTARKA CHAKRABORTY, Computer Science Institute of Charles University
RAGHUNATH TEWARI, Indian Institute of Technology Kanpur

Given a graph G and two vertices s and t in it, *graph reachability* is the problem of checking whether there exists a path from s to t in G. We show that reachability in directed layered planar graphs can be decided in polynomial time and $O(n^\epsilon)$ space, for any $\epsilon > 0$. The previous best-known space bound for this problem with polynomial time was approximately $O(\sqrt{n})$ space (Imai et al. 2013).

Deciding graph reachability in SC (Steve's class) is an important open question in complexity theory, and in this article, we make progress toward resolving this question.

CCS Concepts: • **Theory of computation → Complexity classes**;

Additional Key Words and Phrases: Directed reachability problem, layered planar graph, layered grid graph, time-space tradeoff

ACM Reference format:
Diptarka Chakraborty and Raghunath Tewari. 2017. An $O(n^\epsilon)$ Space and Polynomial Time Algorithm for Reachability in Directed Layered Planar Graphs. *ACM Trans. Comput. Theory* 9, 4, Article 19 (December 2017), 11 pages.
https://doi.org/10.1145/3154857

1 INTRODUCTION

Given a graph and two vertices s and t in it, the problem of determining whether there is a path from s to t in the graph is known as the graph reachability problem. Graph reachability problem is an important question in complexity theory. Particularly in the domain of space bounded computations, the reachability problem in various classes of graphs characterize the complexity of different complexity classes. The reachability problem in directed and undirected graphs is complete for the classes non-deterministic log-space (NL) and deterministic log-space (L), respectively [20, 22]. The latter follows due to a famous result by Reingold who showed that undirected reachability is in L [22]. Various other restrictions of reachability have been studied in the context of understanding the complexity of other space bounded classes (see [14, 19, 23]). Wigderson gave a fairly comprehensive survey that discusses the complexity of reachability in various computational models [26].

A preliminary version of this article [13] was presented in the 26th International Symposium on Algorithms and Computation (ISAAC) 2015, Nagoya, Japan. The work was done while the first author was a PhD student at the Department of Computer Science and Engineering, Indian Institute of Technology Kanpur, Kanpur, India.
Authors' addresses: D. Chakraborty, Room 324, Computer Science Institute of Charles University, Malostranské námesti 25, 118 00 Praha 1, Czech Republic; email: diptarka@iuuk.mff.cuni.cz; R. Tewari, Room 514, Rajeev Motwani Building, Department of Computer Science and Engineering, Indian Institute of Technology Kanpur, Kanpur - 208 016, Uttar Pradesh, India; email: rtewari@cse.iitk.ac.in.

https://doi.org/10.1145/3154857

The time complexity of directed reachability is fairly well understood. Standard graph traversal algorithms such as DFS (depth-first search) and BFS (breadth-first search) solve this problem in linear time. We also have a $O(\log^2 n)$ space algorithm due to Savitch [24], however, it requires $O(n^{\log n})$ time. The question whether there exists a single algorithm that decides reachability in polynomial time and polylogarithmic space is unresolved. In his survey, Wigderson asked whether it is possible to design a polynomial time algorithm that uses only $O(n^\epsilon)$ space for some constant $\epsilon < 1$ [26]. This question is also still open. In 1992, Barnes, Buss, Ruzzo, and Schieber made some progress on this problem and gave an algorithm for directed reachability that requires polynomial time and $O(n/2^{\sqrt{\log n}})$ space [7].

Planar graphs are a natural topological restriction of general graphs consisting of graphs that can be embedded on the surface of a plane such that no two edges cross. *Grid graphs* are a subclass of planar graphs, where the vertices are placed at the lattice points of a two dimensional grid and edges occur between a vertex and its immediate adjacent horizontal or vertical neighbor.

Asano and Doerr provided a polynomial time algorithm to compute the *shortest path* (hence, can decide reachability) in grid graphs, which uses $O(n^{1/2+\epsilon})$ space for any small constant $\epsilon > 0$ [5]. Imai et al. extended this to give a similar bound for reachability in planar graphs [17]. Their approach was to use a space efficient method to design a separator for the planar graph and use the divide and conquer strategy. Note that although it is known that reachability in grid graphs reduces to planar reachability in log-space, since this class (polynomial time and $O(n^{1/2+\epsilon})$ space) is not closed under log-space reductions, planar reachability does not follow from grid graph reachability. Subsequently, the result of Imai et al. was extended to the class of *high-genus* and *H-minor-free* graphs [12]. Recently, Asano et al. gave a $\tilde{O}(\sqrt{n})$ space and polynomial time algorithm for reachability in planar graphs, thus improving upon the previous space bound [6]. More details on known results can be found in a recent survey article [25].

In another line of work, Kannan et al. gave a $O(n^\epsilon)$ space and polynomial time algorithm for solving the reachability problem in *unique path graphs* [18]. Unique path graphs are a generalization of *strongly unambiguous* graphs, and a reachability problem in strongly unambiguous graphs is known to be in SC (Steve's class) (polynomial time and polylogarithmic space) [11, 15]. Reachability in strongly unambiguous graphs can also be decided by an $O(\log^2 n/\log\log n)$ space algorithm; however, this algorithm requires super polynomial time [3]. SC also contains the class *randomized log-space* or RL [21]. We refer the readers to a recent survey by Allender [1] to further understand the results on the complexity of the reachability problem in UL (unambiguous log-space) and on certain special subclasses of directed graphs.

Our Contribution

We show that reachability in directed layered planar graphs can be decided in polynomial time and $O(n^\epsilon)$ space for any constant $\epsilon > 0$. A layered planar graph is a planar graph where the vertex set is partitioned into layers (say L_0 to L_m) and every edge occurs between layers L_i and L_{i+1} only. Our result significantly improves upon the previous space bound due to References [17] and [6] for layered planar graphs.

Theorem 1.1. *For every $\epsilon > 0$, there is a polynomial time and $O(n^\epsilon)$ space algorithm that decides reachability in directed layered planar graphs.*

Reachability in layered grid graphs (denoted as LGGR) is in UL, which is a subclass of NL [2]. Subsequently, this result was extended to the class of all planar graphs [10]. Allender et al. also gave some hardness results for the reachability problem in certain subclasses of layered grid graphs. Specifically, they showed that 1LGGR is hard for [NC (Nick's class)][1] and 11LGGR is hard for TC^0 [2]. Both these problems are, however, known to be contained in L.

As a consequence of our result, it is easy to achieve the same time-space upper-bound for the reachability problem in *upward planar graphs*. We say that a graph is upward planar if it admits an upward planar drawing, i.e., a planar drawing where the curve representing each edge should have the property that every horizontal line intersects it in at most one point. In the domain of graph drawing, it is an important topic to study the upward planar drawing of planar DAGs (directed acyclic graphs) [8, 9]. It is NP-complete to determine whether a planar DAG with multiple sources and sinks has an upward planar drawing [16]. However, given an upward planar drawing of a planar DAG, the reachability problem can easily be reduced to reachability in a layered planar graph using only logarithmic amount of space and, thus, admits the same time-space upper bound as of layered planar graphs.

Firstly, we argue that it is enough to consider layered grid graphs (a subclass of general grid graphs). Now, on a layered grid graph, if we directly apply DFS (depth-first search), then we need linear space, where the space requirement is because of storing the current path and marking already visited vertices while constructing the DFS tree. Now, to avoid linear space usage, we use the divide and conquer strategy. We divide a given layered grid graph into a coarser grid structure along k horizontal and k vertical lines (see Figure 1). We then design a modified DFS strategy that makes queries to the smaller graphs defined by these gridlines (we assume a solution in the smaller graphs by recursion) and visits every reachable vertex from a given start vertex. The modified DFS stores the highest visited vertex in each vertical line and the left-most visited vertex in each horizontal line. We use this information to avoid visiting a vertex a multiple number of times in our algorithm. The assumption that the given graph is a layered grid graph guarantees that we will not miss out on any path while constructing the DFS tree using only this information instead of marking all the previously visited vertices. The choice of the value of k helps us to bound the length of any path and thus the space required to store any current path. We choose the number of horizontal and vertical lines to divide the graph appropriately to ensure that the algorithm runs in the required time and space bound.

The rest of the article is organized as follows. In Section 2, we give some basic definitions and notations that we use in this article. We also state certain earlier results that we use in this article. In Section 3, we give a proof of Theorem 1.1.

2 PRELIMINARIES

We will use the standard notations of graphs without defining them explicitly and follow the standard model of computation to discuss the complexity measures of the stated algorithms. In particular, we consider the computational model in which an input appears on a read-only tape and the output is produced on a write-only tape and we only consider an internal read-write tape in the measure of space complexity. Throughout this article, by log, we mean logarithm to the base 2. We denote the set $\{1, 2, \ldots, n\}$ by $[n]$. Given a graph G, let $V(G)$ and $E(G)$ denote the set of vertices and the set of edges of G, respectively.

Definition 2.1 (Layered Planar Graph). A planar graph $G = (V, E)$ is referred to as *layered planar* if it is possible to represent V as a union of disjoint partitions, $V = V_1 \cup V_2 \cup \cdots \cup V_k$, for some $k > 0$, and for any two consecutive partitions V_i and V_{i+1}, there is a planar embedding of edges from the vertices of V_i to that of V_{i+1} and there is no edge between two vertices of non-consecutive partitions.

Now, let us define the notion of the layered grid graph and also note that grid graphs are by definition planar.

Definition 2.2 (Layered Grid Graph). A directed graph G is said to be an $n \times n$ *grid graph* if it can be drawn on a square grid of size $n \times n$ and two vertices are neighbors if their L_1-distance is one.

In a grid graph, an edge can have four possible directions, i.e., north, south, east, and west, but if we are allowed to have only two directions north and east, then we call it a *layered grid graph.*

We also use the following result of Allender et al. to simplify our proof [2].

Proposition 2.3 ([2]). *The reachability problem in directed layered planar graphs is log-space reducible to the reachability problem in layered grid graphs.*

2.1 Class nSC and Its Properties

$TISP(t(n), s(n))$ denotes the class of languages decided by a deterministic Turing machine that runs in time $O(t(n))$ and uses $O(s(n))$ space. Then, $\mathsf{SC} = TISP(n^{O(1)}, (\log n)^{O(1)})$. Expanding the class SC, we define the complexity class nSC (short for near-SC) in the following definition.

Definition 2.4 (Complexity Class near-SC *or nSC).* For a fixed $\epsilon > 0$, we define

$$nSC_\epsilon := TISP(n^{O(1)}, n^\epsilon).$$

The complexity class nSC is defined as

$$nSC := \bigcap_{\epsilon > 0} nSC_\epsilon.$$

We next show that nSC is closed under log-space reductions, denoted by $\leq_l$. For the definition of log-space reduction, we refer the reader to any standard textbook on computational complexity (e.g., Definition 4.16 of Reference [4]). This is an important property of the class nSC and will be used to prove Theorem 1.1. Although the proof is quite standard, for the sake of completeness, we provide it here.

Theorem 2.5. *If $A \leq_l B$ and $B \in nSC$, then $A \in nSC$.*

Proof. Let us consider that a log-space computable function f be the reduction from A to B. It is clear that for any $x \in A$ such that $|x| = n$, $|f(x)| \leq n^c$ for some constant $c > 0$. We can think that after applying the reduction, $f(x)$ appears in a separate write-once output tape and then we can solve $f(x)$, which is an instance of the language B, and now the input length is at most n^c. Now, take any $\epsilon > 0$ and consider $\epsilon' = \frac{\epsilon}{c} > 0$. $B \in nSC$ implies that $B \in nSC_{\epsilon'}$, and as a consequence, $A \in nSC_\epsilon$. This completes the proof. □

Let us now consider deterministic auxiliary pushdown machines instead of deterministic Turing machines and study the power of the corresponding complexity class. However, note that the result that we are going to discuss now is not required to prove our main theorem, i.e., Theorem 1.1, and thus is of independent interest. First, we define the complexity class P-nSC (short for Pushdown near-SC) as follows.

Definition 2.6 (Complexity Class Pushdown near-SC *or P-nSC).* For a fixed $\epsilon > 0$, we define $P\text{-}nSC_\epsilon$ to be the class of languages decided by a deterministic auxiliary pushdown machine that runs in time $n^{O(1)}$ and uses n^ϵ space. The complexity class P-nSC is defined as

$$P\text{-}nSC := \bigcap_{\epsilon > 0} P\text{-}nSC_\epsilon.$$

Next, we show that in the scenario we are concerned about, a deterministic auxiliary pushdown machine does not provide any extra power over a deterministic Turing machine.

Theorem 2.7. *P-nSC=nSC.*

The above theorem comes as a corollary of an old result by Cook [15], and here, we first restate that result.

Theorem 2.8 ([15]). *If a language is decided by a deterministic auxiliary pushdown machine that runs in time $t(n)$ and uses $s(n) \geq \log n$ space, then that language is in $TISP((t(n))^6, (s(n) + \log t(n)) \log t(n))$.*

Now, it is easy to see that Theorem 2.7 follows from the above theorem.

Proof of Theorem 2.7. From the definition of deterministic auxiliary pushdown machine, it is trivial to see that for any $\epsilon > 0$, $nSC_\epsilon \subseteq P\text{-}nSC_\epsilon$ and thus $nSC \subseteq P\text{-}nSC$.

Now, for the converse direction, let us consider a language $L \in P\text{-}nSC$, which implies $L \in P\text{-}nSC_\epsilon$ for any $\epsilon > 0$. Now, by Theorem 2.8, $L \in nSC_{2\epsilon}$. As a consequence, $L \in nSC$ and this completes the proof. □

3 REACHABILITY IN LAYERED PLANAR GRAPHS

In this section, we prove Theorem 1.1. We first apply Proposition 2.3 and Theorem 2.5 to claim that to prove Theorem 1.1, it is sufficient to show that the reachability problem in layered grid graphs (denoted as LGGR) is in nSC. Now, we devote the rest of the article to prove that it is indeed the case that the problem LGGR is in nSC.

Theorem 3.1. *$LGGR \in nSC$.*

To establish Theorem 3.1, we define an auxiliary graph in Section 3.1 and give the required algorithm in Section 3.2.

3.1 The Auxiliary Graph H

Let G be an $n \times n$ layered grid graph. We denote the vertices in G as (i, j), where $0 \leq i, j \leq n$. Without loss of generality, we can assume that $s = (0, 0)$ and $t = (n, n)$; otherwise, we preprocess G to construct another layered grid graph G' in the following way: consider the subgraph of G such that s be the leftmost and the bottommost vertex and t be the rightmost and the topmost vertex of that subgraph, and then form G' by adding a dummy path from $(0, 0)$ to s and t to (n, n). It is easy to see that the above preprocessing can be done in log-space. Let k be a parameter that determines the number of pieces in which we divide G. We will fix the value of k later to optimize the time and space bounds. Assume without loss of generality that k divides n. Given G, we construct an auxiliary graph H as described below.

Divide G into k^2 many *blocks* (will be defined shortly) of dimension $n/k \times n/k$. More formally, the vertex set of H is

$$V(H) := \{(i, j) \mid i \text{ or } j \text{ is a non-negative multiple of } n/k\}.$$

Note that $V(H) \subseteq V(G)$ and $|V(H)| = 2(k + 1)n - (k + 1)^2$. We consider k^2 many blocks G_1, $G_2, \ldots, G_{k^2}$, where a vertex $(i, j) \in V(G_l)$ if and only if $i' \frac{n}{k} \leq i \leq (i' + 1) \frac{n}{k}$ and $j' \frac{n}{k} \leq j \leq (j' + 1) \frac{n}{k}$, for some integer $i' \geq 0$ and $j' \geq 0$ and the vertices for which any of the four inequalities becomes equality, will be referred to as *boundary vertices*. Moreover, we have $l = i' \cdot k + j' + 1$. $E(G_l)$ is the set of edges in G induced by the vertex set $V(G_l)$.

For every $i \in [k + 1]$, let $L_h(i)$ and $L_v(i)$ denote the set of vertices, $L_h(i) := \{(i', j') | j' = (i - 1) \frac{n}{k}\}$ and $L_v(i) := \{(i', j') | i' = (i - 1) \frac{n}{k}\}$. When it is clear from the context, we will also use $L_h(i)$ and $L_v(i)$ to refer to the corresponding gridline in H. Observe that H has $k + 1$ vertical gridlines and $k + 1$ horizontal gridlines.

For every pair of vertices $u, v \in V(G_l) \cap V(H)$ for some l, add the edge (u, v) to $E(H)$ if and only if there is a path from u to v in G_l, unless $u, v \in L_v(i)$ or $u, v \in L_h(i)$ for some i. Also, for every pair of vertices $u, v \in V(G_l)$ for some l, such that $u = (i_1, j_1)$ and $v = (i_2, j_2)$, where $i_1 = i_2 = i' \frac{n}{k}$ for some i' and $j_1 = j' \frac{n}{k}$, $j_2 = (j' + 1) \frac{n}{k}$ for some j', or $j_1 = j_2 = j' \frac{n}{k}$ for some j' and

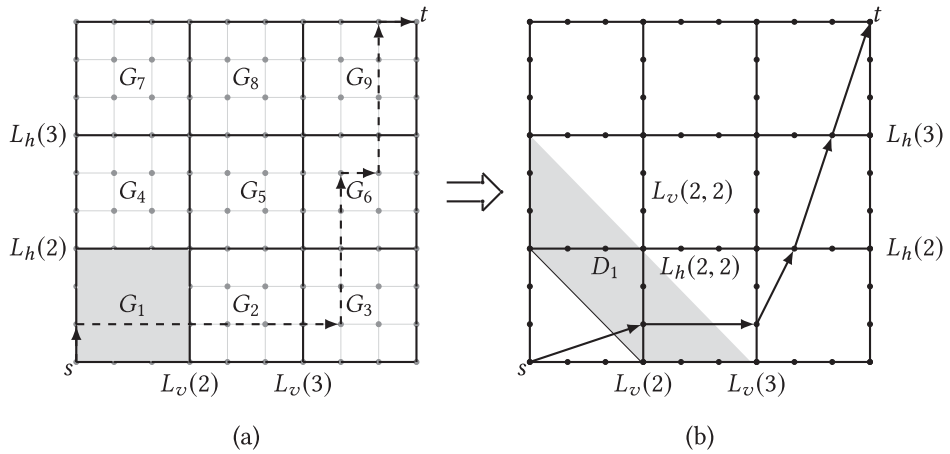

Fig. 1. (a) An example of layered grid graph G and its decomposition into blocks. (b) Corresponding auxiliary graph H.

$i_1 = i'\frac{n}{k}$, $i_2 = (i'+1)\frac{n}{k}$ for some i', we add an edge between u and v in the set $E(H)$ if and only if there is a path from u to v in G_l and we call such vertices *corner vertices*.

Before proceeding further, let us introduce some notation that will be used later. For $j \in [k]$, let $L_h(i,j)$ denote the set of vertices in $L_h(i)$ in between $L_v(j)$ and $L_v(j+1)$. Similarly, we also define $L_v(i,j)$ (see Figure 1). For two vertices $x, y \in L_v(i)$, we say $x \prec y$ if x is *below* y in $L_v(i)$. For two vertices $x, y \in L_h(i)$, we say $x \prec y$ if x is *right of* y in $L_h(i)$. Our algorithm (Algorithm 1) will ensure that for any $x, y \in V(H)$ reachable from s in H, if $x \prec y$, then x will be traversed before y.

Lemma 3.2. *There is a path from s to t in G if and only if there is path from s to t in the auxiliary graph H.*

Proof. As every edge (a,b) in H corresponds to a path from a to b in G, if-part is trivial to see. Now for the only-if-part, consider a path P from s to t in G. P can be decomposed as $P_1P_2\cdots P_r$, such that P_i is a path from x_i to x_{i+1}, where x_i is the first vertex on P that belongs to $V(G_l)$ and x_{i+1} be the last vertex on P that also belongs to $V(G_l)$, for some l. In a layered grid graph, for such x_i and x_{i+1}, we have only following two possibilities:

(1) x_i and x_{i+1} belong to different horizontal or vertical gridlines; or
(2) x_i and x_{i+1} are two corner vertices.

Now, by the construction H, for every i, there must be an edge (x_i, x_{i+1}) in H for both the above cases and, hence, there is a path from s to t in H as well. □

Now, we consider the case when two vertices $x, y \in V(H)$ belong to the same vertical or horizontal gridlines.

Claim 3.3. *Let x and y be two vertices contained in either $L_v(i)$ or $L_h(i)$ for some i. Then, deciding reachability between x and y in G can be done in log-space.*

Proof. Let us consider that $x, y \in L_v(i)$, for some i. As the graph G under consideration is a layered grid graph, if there is a path between x and y, then it must pass through all the vertices in $L_v(i)$ that lies in between x and y. Hence, just by exploring the path starting from x through $L_v(i)$,

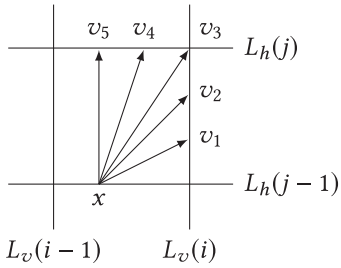

Fig. 2. The neighbors of x are traversed in the following order: v_1, v_2, v_3, v_4, v_5.

we can check the reachability and it is easy to see that this can be done in log-space, because the only thing we need to remember is the current vertex in the path. The same argument will also work when $x, y \in L_h(i)$, for some i, and this completes the proof. □

Now we argue on the upper bound of the length of any path in the auxiliary graph H. The idea is to partition the set $V(H)$ into $2k+1$ partitions in such a way that any two consecutive vertices on a path in H lie on two different partitions.

Lemma 3.4. *Any path between s and t in H is of length $2k-1$.*

Proof. Let us first define the sets $D_0, D_1, \ldots, D_{2k-1}$ (e.g., shaded region in Figure 1(b) denotes D_1), where

$$D_l := \left\{(i,j) | (i'-1)\frac{n}{k} \leq i < i'\frac{n}{k},\ (j'-1)\frac{n}{k} \leq j < j'\frac{n}{k} \text{ and } i'+j' = l+1\right\}.$$

Now consider $D'_l := D_l \cap V(H)$ for $0 \leq l \leq 2k-1$. Clearly, $D'_0, D'_1, \ldots, D'_{2k-1}$ induce a partition on $V(H)$. Now let us take any path $s = x_1 x_2 \cdots x_r = t$, from s to t in H, denoted as P. Observe that by the construction of H, for any two consecutive vertices x_i and x_{i+1} for some i, if $x_i \in D'_l$ for some l, then $x_{i+1} \in D'_{l+1}$ and $s \in D'_0$, $t \in D'_{2k-1}$. As a consequence, $r = 2k$ and, hence, the length of the path P is $2k-1$. □

3.2 Description of the Algorithm

We next give a modified version of DFS that, starting at a given vertex, visits the set of vertices reachable from that vertex in the graph H. At every vertex, the traversal visits the set of outgoing edges from that vertex in counter-clockwise order.

In our algorithm, we maintain two arrays of size $k+1$ each, say A_v and A_h, one for vertical and the other for horizontal gridlines, respectively. For every $i \in [k+1]$, $A_v(i)$ is the *topmost* visited vertex in $L_v(i)$ and, analogously, $A_h(i)$ is the *leftmost* visited vertex in $L_h(i)$. This choice is guided by the choice of traversal of our algorithm. More precisely, we cycle through the outgoing edges of a vertex in counter-clockwise order (see Figure 2).

We perform a standard DFS-like procedure using the tape space to simulate a stack, say S. S keeps track of the path taken to the current vertex from the starting vertex. By Lemma 3.4, the maximum length of a path in H is at most $2k-1$. Whenever we visit a vertex in a vertical gridline (say $L_v(i)$), we check whether the vertex is lower than the i-th entry of A_v. If so, we return to the parent vertex and continue with its next child. Otherwise, we update the i-th entry of A_v to be the current vertex and proceed forward. Similarly, when we visit a horizontal gridline (say $L_h(i)$), we check whether the current vertex is to the right of the i-th entry of A_h. If so, we return to the parent vertex and continue with its next child. Otherwise, we update the i-th entry of A_h to be the

current vertex and proceed. The reason for doing this is to avoid revisiting the subtree rooted at the node of an already visited vertex. The algorithm is formally defined in Algorithm 1.

ALGORITHM 1: AlgoLGGR: Algorithm for Reachability in the Auxiliary Graph H

Input: The auxiliary graph H, two vertices $s, t \in V(H)$
Output: YES if there is a path from s to t; otherwise NO
Initialize two arrays A_v and A_h and a stack S;
Initialize three variables *curr*, *prev* and *next* to NULL;
Push s onto S;
while *S is not empty* **do**
  curr ← top element of S;
  if *prev* ≠ NULL **then**
    next ← neighbor of *curr* following *prev* in counter-clockwise order;
  else
    next ← any neighbor of *curr*;
  end
  while *next* ≠ NULL **do**
    /* cycles through neighbors of curr in counter-clockwise order */
    if *next* = t **then**
      return YES;
    end
    if *next* $\in L_v(i)$ *for some* i *and* ($A_v[i] \prec$ *next or* $A_v[i]$ = NULL) **then**
      $A_v[i]$ ← *next*;
      break;
    end
    if *next* $\in L_h(i)$ *for some* i *and* ($A_h[i] \prec$ *next or* $A_h[i]$ = NULL) **then**
      $A_h[i]$ ← *next*;
      break;
    end
    prev ← *next*;
    next ← neighbor of *curr* following *prev* in counter-clockwise order;
    /* NULL if no more neighbors are present */
  end
  if *next* = NULL **then**
    remove *curr* from S;
    prev ← *curr*;
  else
    add *next* to S;
    prev ← NULL;
  end
end
return NO;

Lemma 3.5. *Let G_l be some block and let x and y be two vertices on the boundary of G_l such that there is a path from x to y in G. Let x' and y' be two other boundary vertices in G_l such that (i) there is a path from x' to y' in G and (ii) x' lies on one segment of the boundary of G_l between vertices x and y and y' lies on the other segment of the boundary. Then, there is a path in G from x to y' and from x' to y. Hence, if (x, y) and (x', y') are present in $E(H)$, then so are (x, y') and (x', y).*

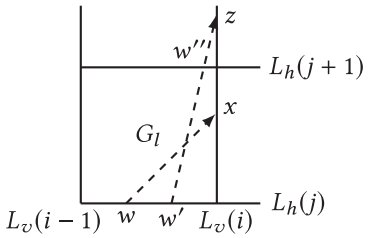

Fig. 3. Crossing between two paths.

Proof. Since G is a layered grid graph, then the paths x to y and x' to y' must lie inside G_l. Also, because of planarity, the paths must intersect at some vertex in G_l. Now using this point of intersection, we can easily show the existence of paths from x to y' and from x' to y. □

The following lemma will help us to prove the correctness of Algorithm 1.

Lemma 3.6. *Let u and v be two vertices in H. Then, starting at u, Algorithm 1 visits v if and only if v is reachable from u in H.*

Proof. It is easy to see that every vertex visited by the algorithm is reachable from u since the algorithm proceeds along the edges of H.

By induction on the shortest path length to a vertex, we will show that if a vertex is reachable from u, then the algorithm visits that vertex. Let $B_d(u)$ be the set of vertices reachable from u that are at a distance d from u. Assume that the algorithm visits every vertex in $B_{d-1}(u)$. Let x be a vertex in $B_d(u)$. Without loss of generality, assume that x is in $L_v(i,j)$ for some i and j. A similar argument can be given if x belongs to a horizontal gridline. Further, let x lie on the right boundary of a block G_l. Let $W_x = \{w \in B_{d-1}(u) | (w,x) \in E(H)\}$. Note that by the definition of H, all vertices in W_x lie on the bottom boundary or on the left boundary of G_l.

Suppose the algorithm does not visit x. Since x is reachable from u via a path of length d, therefore, W_x is non-empty. Let w be the first vertex added to W_x by the algorithm. Then, w is either in $L_h(j)$ or in $L_v(i-1)$. Without loss of generality, assume w is in $L_h(j)$. Let z be the value in $A_v(i)$ at this stage of the algorithm (that is when w is the current vertex). Since x is not visited, $x \prec z$. Also, this implies that z was visited by the algorithm at an earlier stage of the algorithm. Let w' be the ancestor of z in the DFS tree such that w' is in $L_h(j)$. There must exist such a vertex because z is above the j-th horizontal gridline, that is $L_h(j)$.

Suppose if w' lies to the left of w, then by the description of the algorithm, w is visited before w'. Hence, x is visited before z. On the other hand, suppose w' lies to the right of w. Clearly, w' cannot lie to the right of vertical gridline $L_v(i)$ since z is reachable from w' and z is in $L_v(i)$. Let w'' be the vertex in $L_h(j+1)$ such that w'' lies in the tree path between w' and z (see Figure 3). Observe that all four vertices lie on the boundary of G_l. Now, by applying Lemma 3.5 to the four vertices w, x, w', and w'', we conclude that there exists a path from w' to x as well. Since $x \prec z$, x must have been visited before z from the vertex w'. In both cases, we see that z cannot be $A_v(i)$ when w is the current vertex. Since z was an arbitrary vertex such that $x \prec z$, the lemma follows. □

Our next lemma will help us to achieve a polynomial bound on the running time of Algorithm 1.

Lemma 3.7. *Every vertex in the graph H is added to the set S at most once in Algorithm 1.*

Proof. Observe that a vertex u in $L_v(i)$ is added to S only if $A_v(i) \prec u$, and once u is added, $A_v(i)$ is set to u. Also, during subsequent stages of the algorithm, if $A_v(i)$ is set to v, then $u \prec v$. Hence, $u \prec A_v(i)$. Therefore, u cannot be added to S again.

We give a similar argument if u is in $L_h(i)$. Suppose if u is in $L_v(i)$ for some i and $L_h(j)$ for some j, then we add u only once to S. This check is done in Line 16 of Algorithm 1. However, we update both $A_v(i)$ and $A_h(j)$. □

One can observe that Algorithm 1 does not need to explicitly compute or store the graph H. Whenever it is queried for an edge (x, y) in H, it recursively runs a reachability query in the corresponding sub-grid graph of G such that x is in the bottom left corner and y is in the top right corner of that sub grid graph and produces an answer. The base case is when a query is made to a grid graph of size $k \times k$. For the base case, we run a standard DFS procedure on the $k \times k$ size graph.

In every iteration of the *outer while* loop (Lines 4–29) of Algorithm 1, either an element is added or an element is removed from S. Since $|V(H)| < 2(k+1)n$, by Lemma 3.7, the outer while loop iterates at most $5nk$ times. The *inner while* loop (Lines 7–21) cycles through all the neighbors of a vertex, which is bounded by $2n/k$ and, hence, iterates for at most $2n/k$ times. Each iteration of the inner while loop makes a constant number of calls to check the presence of an edge in an $n/k \times n/k$ sized grid. Let $\mathcal{T}(n)$ and $\mathcal{S}(n)$ be the time and space required to decide reachability in a layered grid graph of size $n \times n$, respectively. Then,

$$\mathcal{T}(n) = \begin{cases} 10n^2(\mathcal{T}(n/k) + O(1)) & \text{if } n > k \\ O(k^2) & \text{otherwise.} \end{cases}$$

Hence, $\mathcal{T}(n) = O(n^{3\frac{\log n}{\log k}})$.

Since we do not store any query made to the smaller grids, the space required to check the presence of an edge in H can be reused. A_v and A_h are arrays of size $k+1$ each. By Lemma 3.4, the number of elements in S at any stage of the algorithm is bounded by $2k-1$. Thus, to store A_v, A_h, and S, our algorithm needs total $(|A_v| + |A_h| + |S|) \log n = O(k \log n)$ space. Therefore,

$$\mathcal{S}(n) = \begin{cases} \mathcal{S}(n/k) + O(k \log n) & \text{if } n > k \\ O(k^2) & \text{otherwise.} \end{cases}$$

Hence, $\mathcal{S}(n) = O(\frac{k}{\log k} \log^2 n + k^2)$.

Now, given any constant $\epsilon > 0$, if we set $k = n^{\epsilon/2}$, then we get $\mathcal{T}(n) = O(n^{6/\epsilon})$ and $\mathcal{S}(n) = O(n^{\epsilon})$. This proves Theorem 3.1.

ACKNOWLEDGMENTS

We thank N. V. Vinodchandran for his helpful suggestions and comments. The first author would like to acknowledge the support of Research-I Foundation.

REFERENCES

[1] Eric Allender. 2007. Reachability problems: An update. In *Proceedings of Computation and Logic in the Real World, 3rd Conference on Computability in Europe (CiE'07)*. 25–27. DOI: http://dx.doi.org/10.1007/978-3-540-73001-9_3

[2] Eric Allender, David A. Mix Barrington, Tanmoy Chakraborty, Samir Datta, and Sambuddha Roy. 2009. Planar and grid graph reachability problems. *Theory Comput. Syst.* 45, 4 (2009), 675–723. DOI: http://dx.doi.org/10.1007/s00224-009-9172-z

[3] Eric Allender and Klaus-Jörn Lange. 1998. RUSPACE(log n) ⊆ DSPACE ($\log^2 n$ / log log n). *Theory Comput. Syst.* 31, 5 (1998), 539–550. DOI: http://dx.doi.org/10.1007/s002240000102

[4] Sanjeev Arora and Boaz Barak. 2009. *Computational Complexity - A Modern Approach*. Cambridge University Press.

[5] Tetsuo Asano and Benjamin Doerr. 2011. Memory-constrained algorithms for shortest path problem. In *Proceedings of the 23rd Annual Canadian Conference on Computational Geometry*.

[6] Tetsuo Asano, David G. Kirkpatrick, Kotaro Nakagawa, and Osamu Watanabe. 2014. $\tilde{O}(\sqrt{n})$-Space and polynomial-time algorithm for planar directed graph reachability. In *Proceedings of the 39th International Symposium on Mathematical Foundations of Computer Science 2014, Part II, MFCS 2014, Budapest, Hungary, August 25-29, 2014*. 45–56.

[7] Greg Barnes, Jonathan F. Buss, Walter L. Ruzzo, and Baruch Schieber. 1992. A sublinear space, polynomial time algorithm for directed s-t connectivity. In *Proceedings of the 7th Annual Structure in Complexity Theory Conference*. 27–33. DOI: http://dx.doi.org/10.1109/SCT.1992.215378

[8] Giuseppe Di Battista, Wei-Ping Liu, and Ivan Rival. 1990. Bipartite graphs, upward drawings, and planarity. *Inf. Process. Lett.* 36, 6 (1990), 317–322. DOI: http://dx.doi.org/10.1016/0020-0190(90)90045-Y

[9] Giuseppe Di Battista and Roberto Tamassia. 1987. Upward drawings of acyclic digraphs. In *Proceedings of the Graph-Theoretic Concepts in Computer Science, International Workshop, WG'87, Kloster Banz/Staffelstein, Germany, June 29 - July 1, 1987*. 121–133. DOI: http://dx.doi.org/10.1007/3-540-19422-3_10

[10] Chris Bourke, Raghunath Tewari, and N. V. Vinodchandran. 2009. Directed planar reachability is in unambiguous log-space. *ACM Trans. Comput. Theory* 1, 1 (2009), 1–17. DOI: http://dx.doi.org/10.1145/1490270.1490274

[11] Gerhard Buntrock, Birgit Jenner, Klaus-Jrn Lange, and Peter Rossmanith. 1991. Unambiguity and fewness for logarithmic space. In *Fundamentals of Computation Theory*, L. Budach (Ed.). Lecture Notes in Computer Science, Vol. 529. Springer Berlin, 168–179. DOI: http://dx.doi.org/10.1007/3-540-54458-5_61

[12] Diptarka Chakraborty, Aduri Pavan, Raghunath Tewari, N. V. Vinodchandran, and Lin F. Yang. 2014. New time-space upperbounds for directed reachability in high-genus and h-minor-free graphs. In *Proceedings of the 34th International Conference on Foundation of Software Technology and Theoretical Computer Science (FSTTCS'14)*. 585–595. DOI: http://dx.doi.org/10.4230/LIPIcs.FSTTCS.2014.585

[13] Diptarka Chakraborty and Raghunath Tewari. 2015. An $O(n^{\epsilon})$ space and polynomial time algorithm for reachability in directed layered planar graphs. In *Proceedings of the 26th International Symposium on Algorithms and Computation, ISAAC 2015, Nagoya, Japan, December 9-11, 2015*. 614–624. DOI: http://dx.doi.org/10.1007/978-3-662-48971-0_52

[14] Kai-Min Chung, Omer Reingold, and Salil Vadhan. 2011. S-T connectivity on digraphs with a known stationary distribution. *ACM Trans. Algorithms* 7, 3, Article 30 (July 2011), 21 pages. DOI: http://dx.doi.org/10.1145/1978782.1978785

[15] S. A. Cook. 1979. Deterministic CFL's are accepted simultaneously in polynomial time and log squared space. In *Proceedings of the 11th Annual ACM Symposium on Theory of Computing*. ACM, 338–345.

[16] Ashim Garg and Roberto Tamassia. 1995. Upward planarity testing. In *SIAM Journal on Computing*. 436–441.

[17] T. Imai, K. Nakagawa, A. Pavan, N. V. Vinodchandran, and O. Watanabe. 2013. An $O(n^{1/2+\epsilon})$-space and polynomial-time algorithm for directed planar reachability. In *Proceedings of the 2013 IEEE Conference on Computational Complexity (CCC)*. 277–286. DOI: http://dx.doi.org/10.1109/CCC.2013.35

[18] Sampath Kannan, Sanjeev Khanna, and Sudeepa Roy. 2008. STCON in directed unique-path graphs. In *Proceedings of the IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (Leibniz International Proceedings in Informatics (LIPIcs))*, Ramesh Hariharan, Madhavan Mukund, and V Vinay (Eds.), Vol. 2. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 256–267. DOI: http://dx.doi.org/10.4230/LIPIcs.FSTTCS.2008.1758

[19] Klaus-Jörn Lange. 1997. An unambiguous class possessing a complete set. In *Proceedings of the 14th Annual Symposium on Theoretical Aspects of Computer Science (STACS'97)*. 339–350. DOI: http://dx.doi.org/10.1007/BFb0023471

[20] Harry R. Lewis and Christos H. Papadimitriou. 1982. Symmetric space-bounded computation. *Theor. Comput. Sci.* 19 (1982), 161–187. DOI: http://dx.doi.org/10.1016/0304-3975(82)90058-5

[21] Noam Nisan. 1995. RL $\subseteq$ SC. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*. 619–623.

[22] Omer Reingold. 2008. Undirected connectivity in log-space. *J. ACM* 55, 4 (2008), 17:1–17:24. DOI: http://dx.doi.org/10.1145/1391289.1391291

[23] Omer Reingold, Luca Trevisan, and Salil Vadhan. 2006. Pseudorandom walks on regular digraphs and the RL vs. L problem. In *STOC'06: Proceedings of the 38th Annual ACM Symposium on Theory of Computing*. ACM, New York, NY, 457–466. DOI: http://dx.doi.org/10.1145/1132516.1132583

[24] Walter J. Savitch. 1970. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.* 4 (1970), 177–192.

[25] N. V. Vinodchandran. 2014. *Space Complexity of the Directed Reachability Problem over Surface-Embedded Graphs*. Technical Report TR14-008. I.

[26] Avi Wigderson. 1992. The complexity of graph connectivity. In *Proceedings of the 17th International Symposium on Mathematical Foundations of Computer Science 1992, MFCS'92, Prague, Czechoslovakia, August 24-28, 1992*. 112–132. DOI: http://dx.doi.org/10.1007/3-540-55808-X_10

Received April 2016; revised August 2017; accepted October 2017