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

Diptarka Chakraborty*and Raghunath Tewari [†]

Department of Computer Science and Engineering,
Indian Institute of Technology Kanpur,
Kanpur, India

April 27, 2015

Abstract

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 [INP+13].

Deciding graph reachability in SC is an important open question in complexity theory and in this paper we make progress towards resolving this question.

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 [LP82, Rei08]. The latter follows due to a famous result by Reingold who showed that undirected reachability is in L [Rei08]. Various other restrictions of reachability has been studied in the context of understanding the complexity of other space bounded classes (see [RTV06, CRV11, Lan97]). Wigderson gave a fairly comprehensive survey that discusses the complexity of reachability in various computational models [Wig92].

The time complexity of directed reachability is fairly well understood. Standard graph traversal algorithms such as DFS and BFS solve this problem in linear time. We also have a $O(\log^2 n)$ space algorithm due to Savitch [Sav70], 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

---

*diptarka@cse.iitk.ac.in

[†]rtewari@cse.iitk.ac.in

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 $\epsilon < 1$ [Wig92]. 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 [BBRS92].

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 $\epsilon > 0$ [AD11]. Imai et al extended this to give a similar bound for reachability in planar graphs [INP$^+$13]. Their approach was to use a space efficient method to design a separator for the planar graph and use divide and conquer strategy. Note that although it is known that reachability in grid graphs reduces to planar reachability in log space, however 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 [CPT$^+$14]. 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 [AKNW14]. More details on known results can be found in a recent survey article [Vin14].

In another line of work, Kannan et al gave a $O(n^\epsilon)$ space and polynomial time algorithm for solving reachability problem in *unique path graphs* [KKR08]. Unique path graphs are a generalization of *strongly unambiguous* graphs and reachability problem in strongly unambiguous graphs is known to be in SC (polynomial time and polylogarithmic space) [BJLR91, Coo79]. Reachability in strongly unambiguous graphs can also be decided by a $O(\log^2 n / \log\log n)$ space algorithm, however this algorithm requires super polynomial time [AL98]. SC also contains the class *randomized log space* or RL [Nis95]. We refer the readers to a recent survey by Allender [All07] to further understand the results on the complexity of reachability problem in UL 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 [INP$^+$13] and [AKNW14] for layered planar graphs.

Theorem 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 is in UL which is a subclass of NL [ABC$^+$09]. Subsequently this result was extended to the class of all planar graphs [BTV09]. Allender et al also gave some hardness results the reachability problem in certain subclasses of layered grid graphs. Specifically they showed that, 1LGGR is hard for NC^1 and 11LGGR is hard for TC^0 [ABC$^+$09]. Both these problems are however known to be contained in L though.

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 [BT87, BLR90]. It is NP-complete to determine whether a planar DAG with multiple sources and sinks has an upward planar drawing [GT95]. 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 its enough to consider layered grid graphs (a subclass of general grid graphs). We divide a given layered grid graph into a courser 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 multiple number of times in our algorithm. 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 paper is organized as follows. In Section 2, we give some basic definitions and notations that we use in this paper. We also state certain earlier results that we use in this paper. In Section 3, we give a proof of Theorem 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 paper, by log we mean logarithm to the base 2. We denote the set $\{1, 2, \cdots, n\}$ 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 1 (Layered Planar Graph). *A planar graph $G = (V, E)$ is referred 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 there is a planar embedding of edges between the vertices of any two consecutive partitions V_i and V_{i+1} and there is no edge between two vertices of non-consecutive partitions.*

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

Definition 2 (Layered Grid Graph). *A directed graph G is said to be a $n \times n$ 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 a 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 [ABC+09].

Proposition 2 ([ABC+09]). *Reachability problem in directed layered planar graphs is log-space reducible to the reachability problem in layered grid graphs.*

3

2.1 Class nSC and its properties

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

Definition 3 (Complexity Class `near-SC` or nSC). *For a fixed $\epsilon > 0$, we define $\mathsf{nSC}_\epsilon := \mathsf{TISP}(n^{O(1)}, n^\epsilon)$. The complexity class nSC is defined as*

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

We next show that nSC is closed under log-space reductions. This is an important property of the class nSC and will be used to prove Theorem 1. Although the proof is quite standard, but for the sake of completeness we provide it here.

Theorem 3. *If $A \leq_l B$ and $B \in \mathsf{nSC}$, then $A \in \mathsf{nSC}$.*

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 $\epsilon' = \frac{\epsilon}{c} > 0$. $B \in \mathsf{nSC}$ implies that $B \in \mathsf{nSC}_{\epsilon'}$ and as a consequence, $A \in \mathsf{nSC}_\epsilon$. This completes the proof. □

3 Reachability in Layered Planar Graphs

In this section we prove Theorem 1. We show that the reachability problem in layered grid graphs, (denoted as LGGR) is in nSC (Theorem 4). Then by applying Proposition 2 and Theorem 3 we have the proof of Theorem 1.

Theorem 4. $\mathsf{LGGR} \in \mathsf{nSC}$.

To establish Theorem 4 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 a $n \times n$ layered grid graph. We denote the vertices in G as (i, j), where $0 \leq i, j \leq n$ and without loss of generality, we can assume that $s = (0, 0)$ and $t = (n, n)$. 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 $V(H) \subseteq V(G)$. We consider k^2 many blocks $G_1, G_2, \cdots, G_{k^2}$, 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
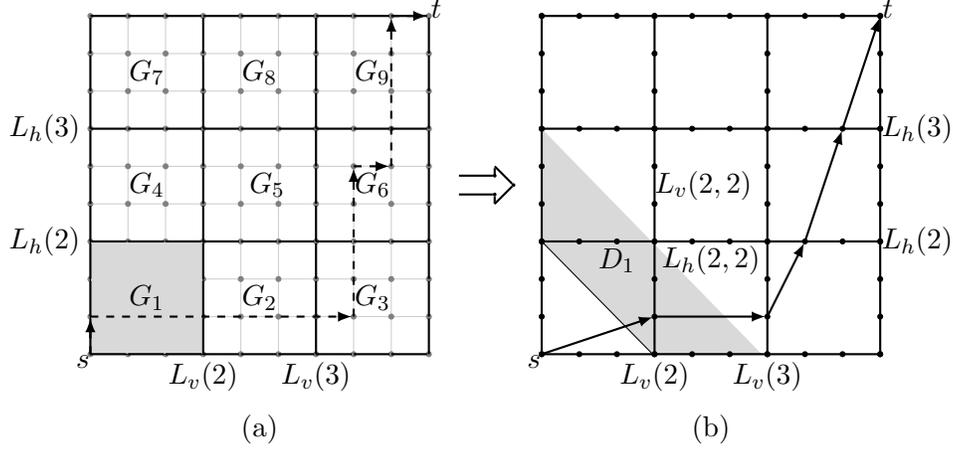
Figure 1: (a) An example of layered grid graph G and its decomposition into blocks (b) Corresponding auxiliary graph H

$j' \geq 0$ and the vertices for which any of the four inequalities becomes equality, will be referred 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 $i_1 = i'\frac{n}{k}$, $i_2 = (i'+1)\frac{n}{k}$ for some i', we add 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 as *corner vertices*.

Before proceeding further, let us introduce a few more notations 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 $L_h(i)$. Note that we consider these two type of orderings to ensure that for any $x, y \in V(H)$ reachable from s in H, if $x \prec y$, then x will be traversed by our algorithm before y.

Lemma 5. *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, so 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 $P_1 P_2 \cdots P_r$, 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 and in a layered grid graph, for such x_i and x_{i+1}, we have only following two possibilities:

5

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 6. *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 $L_v(i)$, 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. 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 7. *Any path between s and t in H is of length $2k$.*

Proof. Let us first define the sets $D_0, D_1, \cdots, D_{2k}$ (e.g., shaded region in Figure 1(b) denotes D_1), where
$$D_l := \{(i,j) | (i'-1)\frac{n}{k} \le i < i'\frac{n}{k}, \ (j'-1)\frac{n}{k} \le j < j'\frac{n}{k} \text{ and } i'+j'=l+1\}.$$
Now consider $D'_l := D_l \cap V(H)$ for $0 \le l \le 2k$. Clearly, $D'_0, D'_1, \cdots, D'_{2k}$ 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 $s \in D'_0$, $t \in D'_{2k}$. As a consequence, $r = 2k+1$ and hence length of the path P is $2k$. □

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 an 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 an counter-clockwise order.

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 7, the maximum length of a path in H is at most $2k$. 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 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.

Lemma 8. *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).*

Proof. Since G is a layered grid graph hence 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. $\square$ $\square$

Lemma 9 will prove the correctness of Algorithm 1.

Lemma 9. *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 hence $x \prec z$. 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 if 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 2). Observe that all four vertices lie on the boundary of G_l. Now by applying Lemma 8 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

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

1 Initialize two arrays A_v and A_h and a stack S;
2 Initialize three variables $curr$, $prev$ and $next$ to NULL;
3 Push s onto S;
4 **while** S *is not empty* **do**
5 $curr \leftarrow$ top element of S;
6 $next \leftarrow$ neighbor of $curr$ next to $prev$ in counter-clockwise order;
7   **while** $next \neq$ NULL **do**
       /* cycles through neighbors of curr   */
8   **if** $next = t$ **then**
9   **return** YES;
10   **end**
11   **if** $next \in L_v(i)$ *for some* i *and* $A_v[i] \prec next$ **then**
12 $A_v[i] \leftarrow next$;
13   **break**;
14   **end**
15   **if** $next \in L_h(i)$ *for some* i *and* $A_h[i] \prec next$ **then**
16 $A_h[i] \leftarrow next$;
17   **break**;
18   **end**
19 $prev \leftarrow next$;
20 $next \leftarrow$ neighbor of $curr$ next to $prev$ in counter-clockwise order;
       /* NULL if no more neighbors are present   */
21   **end**
22   **if** $next =$ NULL **then**
23   remove $curr$ from S;
24 $prev \leftarrow curr$;
25   **else**
26   add $next$ to S;
27 $prev \leftarrow$ NULL;
28   **end**
29 **end**
30 **return** NO;

Algorithm 1: AlgoLGGR: Algorithm for Reachability in the Auxiliary Graph H
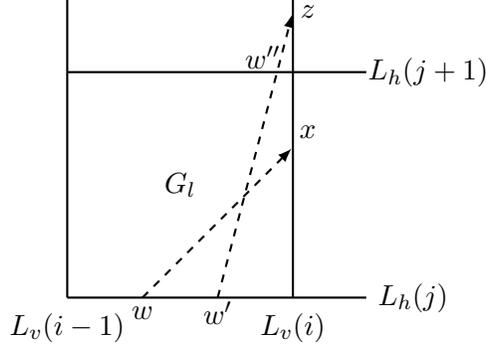
Figure 2: Crossing between two paths

be $A_v(i)$ when w is the current vertex. Since z was an arbitrary vertex such that $x \prec z$, the lemma follows. □ □

We next show Lemma 10 which will help us to achieve a polynomial bound on the running time of Algorithm 1.

Lemma 10. *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 $A_h(j)$. □ □

Algorithm 1 does not explicitly compute and 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. Hence by Lemma 10 the loop iterates at most $4nk$ times. The *inner while* loop (Lines 7 – 21), cycles through all the neighbors of a vertex 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 a $n/k \times n/k$ 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,

$$\mathcal{T}(n) = \begin{cases} 8n^2(\mathcal{T}(n/k) + O(1)) & \text{if } n > k \\ O(k^2) & \text{otherwise.} \end{cases}$$

Hence,

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

9

Since we do not store any query made to the smaller grids, therefore 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 7, the number of elements in S at any stage of the algorithm is bounded by $2k$. 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,

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

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 4.

Acknowledgement

References

[ABC$^+$09] Eric Allender, David A. Mix Barrington, Tanmoy Chakraborty, Samir Datta, and Sambuddha Roy. Planar and grid graph reachability problems. *Theory of Computing Systems*, 45(4):675–723, 2009.

[AD11] Tetsuo Asano and Benjamin Doerr. Memory-constrained algorithms for shortest path problem. In *CCCG*, 2011.

[AKNW14] Tetsuo Asano, David G. Kirkpatrick, Kotaro Nakagawa, and Osamu Watanabe. $\tilde{O}(\sqrt{n})$-space and polynomial-time algorithm for planar directed graph reachability. In *Mathematical Foundations of Computer Science 2014 - 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part II*, pages 45–56, 2014.

[AL98] Eric Allender and Klaus-Jörn Lange. Ruspace(log n) $\subseteq$ DSPACE (log^2 n / log log n). *Theory Comput. Syst.*, 31(5):539–550, 1998.

[All07] E. Allender. Reachability problems: An update. *Computation and Logic in the Real World*, pages 25–27, 2007.

[BBRS92] Greg Barnes, Jonathan F. Buss, Walter L. Ruzzo, and Baruch Schieber. A sublinear space, polynomial time algorithm for directed s-t connectivity. In *Structure in Complexity Theory Conference, 1992., Proceedings of the Seventh Annual*, pages 27–33, 1992.

[BJLR91] Gerhard Buntrock, Birgit Jenner, Klaus-Jrn Lange, and Peter Rossmanith. Unambiguity and fewness for logarithmic space. In L. Budach, editor, *Fundamentals of Computation Theory*, volume 529 of *Lecture Notes in Computer Science*, pages 168–179. Springer Berlin Heidelberg, 1991.

[BLR90]   Giuseppe Di Battista, Wei-Ping Liu, and Ivan Rival. Bipartite graphs, upward draw-ings, and planarity. *Inf. Process. Lett.*, 36(6):317–322, 1990.

[BT87]   Giuseppe Di Battista and Roberto Tamassia. Upward drawings of acyclic digraphs. In *Graph-Theoretic Concepts in Computer Science, International Workshop, WG '87, Kloster Banz/Staffelstein, Germany, June 29 - July 1, 1987, Proceedings*, pages 121–133, 1987.

[BTV09]   Chris Bourke, Raghunath Tewari, and N. V. Vinodchandran. Directed planar reachabil-ity is in unambiguous log-space. *ACM Transactions on Computation Theory*, 1(1):1–17, 2009.

[Coo79]   S.A. Cook. Deterministic CFL's are accepted simultaneously in polynomial time and log squared space. In *Proceedings of the eleventh annual ACM Symposium on Theory of Computing*, pages 338–345. ACM, 1979.

[CPT+14] Diptarka Chakraborty, Aduri Pavan, Raghunath Tewari, N. V. Vinodchandran, and Lin Yang. New time-space upperbounds for directed reachability in high-genus and h-minor-free graphs. In *34th International Conference on Foundation of Software Tech-nology and Theoretical Computer Science, FSTTCS 2014, December 15-17, 2014, New Delhi, India*, pages 585–595, 2014.

[CRV11]   Kai-Min Chung, Omer Reingold, and Salil Vadhan. S-t connectivity on digraphs with a known stationary distribution. *ACM Trans. Algorithms*, 7(3):30:1–30:21, July 2011.

[GT95]   Ashim Garg and Roberto Tamassia. Upward planarity testing. In *SIAM Journal on Computing*, pages 436–441, 1995.

[INP+13] T. Imai, K. Nakagawa, A. Pavan, N.V. Vinodchandran, and O. Watanabe. An $O(n^{1/2+\epsilon})$-Space and Polynomial-Time Algorithm for Directed Planar Reachability. In *Computational Complexity (CCC), 2013 IEEE Conference on*, pages 277–286, 2013.

[KKR08]   Sampath Kannan, Sanjeev Khanna, and Sudeepa Roy. STCON in Directed Unique-Path Graphs. In Ramesh Hariharan, Madhavan Mukund, and V Vinay, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 2 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 256–267, Dagstuhl, Germany, 2008. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[Lan97]   Klaus-Jörn Lange. An unambiguous class possessing a complete set. In *STACS '97: Proceedings of the 14th Annual Symposium on Theoretical Aspects of Computer Sci-ence*, pages 339–350, 1997.

[LP82]   Harry R. Lewis and Christos H. Papadimitriou. Symmetric space-bounded computa-tion. *Theor. Comput. Sci.*, 19:161–187, 1982.

[Nis95]   Noam Nisan. RL $\subseteq$ SC. In *In Proceedings of the Twenty Fourth Annual ACM Sympo-sium on Theory of Computing*, pages 619–623, 1995.

[Rei08]   Omer Reingold. Undirected connectivity in log-space. *Journal of the ACM*, 55(4), 2008.

[RTV06]   Omer Reingold, Luca Trevisan, and Salil Vadhan. Pseudorandom walks on regular digraphs and the RL vs. L problem. In *STOC '06: Proceedings of the thirty-eighth annual ACM Symposium on Theory of Computing*, pages 457–466, New York, NY, USA, 2006. ACM.

[Sav70]   Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4:177–192, 1970.

[Vin14]   N. V. Vinodchandran. Space complexity of the directed reachability problem over surface-embedded graphs. Technical Report TR14-008, I, 2014.

[Wig92]   Avi Wigderson. The complexity of graph connectivity. *Mathematical Foundations of Computer Science 1992*, pages 112–132, 1992.