$\widetilde{O}(\sqrt{n})$-Space and Polynomial-Time Algorithm for Planar Directed Graph Reachability

Tetsuo Asano[1], David Kirkpatrick[2], Kotaro Nakagawa[3], and Osamu Watanabe[3]

[1] Japan Advanced Institute of Technology, Japan
[2] Department of Computer Science, University of British Columbia, Canada
[3] Department of Math. and Comput. Sci., Tokyo Institute of Technology, Japan

Abstract. The directed graph reachability problem takes as input an n-vertex directed graph $G = (V, E)$, and two distinguished vertices v_0, and vertex v_*. The problem is to determine whether there exists a path from v_0 to v_* in G. The main result of this paper is to show that the directed graph reachability problem restricted to planar graphs can be solved in polynomial time using only $\widetilde{O}(\sqrt{n})$ space[1].

1 Introduction and Motivation

For a directed graph $G = (V, E)$, its *underlying graph* is the undirected graph ${}^{\text{`}}G = (V, {}^{\text{`}}E)$, where the vertex pair $\{u, v\}$ belongs to ${}^{\text{`}}E$ if and only if at least one of (u, v) or (v, u) belongs to E. The *planar directed graph reachability problem* is a special case of the directed graph reachability problem where we restrict attention to input graphs whose underlying graph is *planar*.

The general directed graph reachability problem is a core problem in computational complexity theory. It is a canonical complete problem for nondeterministic log-space, NL, and the famous open question L = NL is essentially asking whether the problem is solvable deterministically in log-space. The standard breadth first search algorithm and Savitch's algorithm are two of the most fundamental algorithms known for solving the directed graph reachability problem. The former uses space and time linear in the number of edges of the input graph and the latter uses only $O((\log n)^2)$-space but requires $\Theta(n^{\log n})$ time. Hence a natural and significant question is whether we can design an algorithm for directed graph reachability that is efficient in both space and time. In particular, can we design a polynomial-time algorithm for the directed graph reachability problem that uses only $O(n^\epsilon)$-space for some small constant $\epsilon < 1$? This question was asked by Wigderson in his excellent survey paper [13], and it remains unsettled. The best known result in this direction is the two decades old bound due to Barns, Buss, Ruzzo and Schieber [4], who showed a polynomial-time algorithm for the problem that uses $O(n/2^{\sqrt{\log n}})$ space. Note that this space bound is only

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

slightly sublinear, and improving this bound remains a significant open question. In fact, there are indications that it may be difficult to improve this bound because there are matching *lower bounds* known for solving the directed graph reachability problem on a certain model of computation known as NNJAG; see, e.g., [5]. Though NNJAG is a restrictive model, all the known algorithms for the directed reachability can be implemented in NNJAG without significant blow up in time and space.

Some important progress has been made for restricted graph classes. The most remarkable one is the log-space algorithm of Reingold (which we will refer as `UReach`) for the undirected graph reachability [12]. Recently, Asano and Doerr [2] gave a $\widetilde{O}(n^{1/2+\varepsilon})$-space and $n^{O(1/\epsilon)}$-time algorithm for the reachability problem restricted to directed grid graphs. Inspired by this result Imai et al. proposed [8] an $\widetilde{O}(n^{1/2+\varepsilon})$-space and $n^{O(1/\epsilon)}$-time algorithm for the planar graph reachability problem. It has been left open to design an $\widetilde{O}(n^{1/2})$-space and yet polynomial-time algorithm. More recently, Asano and Kirkpatrick [3] introduced a more efficient way to control the recursion, thereby succeeding to obtain an $\widetilde{O}(\sqrt{n})$-space and polynomial time algorithm for the reachability problem restricted to directed grid graphs. The main result of this paper is to show that this technique can be adapted to design an $\widetilde{O}(\sqrt{n})$-space and polynomial-time algorithm for the planar graph reachability problem.

2 Background

The Planar Separator Theorem, shown first by Lipton and Tarjan [10], asserts that, for any n-vertex undirected planar graph G, there is a polynomial-time algorithm for computing an $O(\sqrt{n})$-size "separator" for G, i.e. a set of $O(\sqrt{n})$ vertices whose removal separates the graph into two subgraphs of similar size. The key idea of the $\widetilde{O}(n^{1/2+\varepsilon})$-space algorithm of Imai et al. is to use an space-constrained algorithmic version of the Planar Separator Theorem. For a given input instance (G, v_0, v_*), they first compute a separator S of the underlying planar graph of G that separates G into two smaller subgraphs G^0 and G^1. They then consider a new directed graph H on $S \cup \{v_0, v_*\}$; it has a directed edge (a, b) if and only if there is a path from a to b in either G^0 or G^1. Clearly, reachability (of v_* from v_0) in H is equivalent to the original reachability. On the other hand, since S has $O(\sqrt{n})$ vertices, the standard linear-space and polynomial-time algorithm can be used to solve the reachability problem in H by applying the algorithm recursively to G^0 and G^1 whenever it is necessary to know if an edge (a, b) exists in H. It should be mentioned that the idea of using separators to improve algorithms for the reachability and related problems is natural, and in fact it has been proposed by several researchers; see, e.g., [7]. The main contribution of [8] is to show how to implement this idea by giving a space efficient separator algorithm based on the parallel separator algorithms of Miller [9] and Gazit and Miller [6].

The algorithm of Asano and Kirkpatrick uses a recursive separation of a grid graph; at each level a separator is formed by the set of vertices on one of the grid center lines. In order to get a polynomial time bound, Asano and Kirkpatrick introduce a kind of budgeted recursion, controlled by a "universal sequence", that restricts the time complexity of each recursive execution on smaller grids. Here we use the same idea for the general planar graph reachability. To mimic the grid-based algorithm of Asano and Kirkpatrick we need a simple separator that allows us to express/identify a hierarchy of subgraphs succinctly/simply. The main technical contribution of this paper is to describe a space efficient way to construct such a simple separator together with a succinct way to express the separated subgraphs. (We note that it is not immediately clear whether the sublinear-space algorithm of [8] always yields such a simple separator. Here instead of analyzing the separator algorithm of [8], we show how to modify a given separator to obtain a suitable cycle-separator.)

3 Planar Graph Reachability Algorithm

3.1 Preliminaries

Let $G = (V, E)$ denote an arbitrary directed graph. For any subset U of V we use $G[U]$ to denote the subgraph of G induced by U. For two graphs G_1 and G_2, by $G_1 \cup G_2$ we mean the graph $(V(G_1) \cup V(G_2), E(G_1) \cup E(G_2))$. For any graph G, consider any subgraph H of G and any vertex v of G that is not in H; then by $H \sqcup_G v$ we denote an induced subgraph of G obtained from H by adding vertex v; that is, $H \sqcup_G v = G[V(H) \cup \{v\}]$. Similarly, for an arbitrary set $A \subset V$, we use $H \sqcup_G A$ to denote $G[V(H) \cup A]$.

Recall that a graph is *planar* if it can be drawn on a plane so that the edges intersect only at end vertices. Such a drawing is called a *planar embedding.* Here we use the standard way to specify a *planar embedding* of G; that is, a sequence of vertices adjacent to v in a clockwise order around v under the planar embedding, for all $v \in V$. We use $\mathrm{N}(v)$ to denote this sequence for v, which is often regarded as a set. For a planar graph and its planar embedding, its *triangulation* (w.r.t. this planar embedding) means to add edges to the planar graph until all its faces under the planar embedding (including the outer one) are bounded by three edges. We note that, in assuming that the input of our algorithm is a planar graph, we assume only that a planar embedding exists, not that it is given as part of the input.

3.2 The Algorithm

We now describe our algorithm for planar graph reachability. To illustrate the idea of the algorithm we consider first the case in which the input graph G is a subgraph of a bi-directed grid graph like Figure 1(a). Here we assume that the original bi-directed grid graph is a $(2^h - 1) \times (2^h - 1)$ square grid. Let ‘G denote the undirected version of this original grid. Note that both G and ‘G have $n = (2^h - 1) \times (2^h - 1)$ vertices.

Although reachability is determined by the edges of G, the computation is designed based on the underlying graph ‘G. Consider a set S of vertices that are

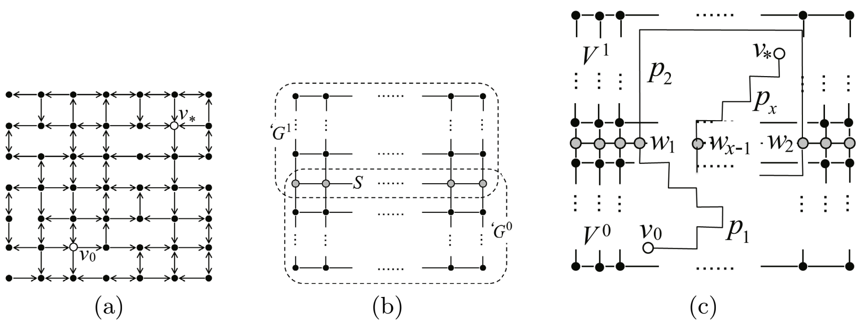


Fig. 1. Example grid graph G and a path from v_0 to v_*

on the horizontal center line. We call S a *grid-separator* because ‘G is separated to two disconnected subgraphs by removing S. Our strategy is to determine, for every vertex $v \in G$, the reachability from v_0 in each subgraph independently, thereby saving the space for the computation. More specifically, we consider a subgraph ‘G^0 (resp., ‘G^1) of ‘G consisting of vertices below (resp., above) S including S (*cf.* Figure 1(b)), and compute the reachability from v_0 on G in the area defined by each subgraph. A crucial point is that we need to keep the reachability information only for vertices in S in order to pass the reachability information to the next computation on the opposite subarea. Also for showing the $\widetilde{O}(\sqrt{n})$-space bound, it is important that S consists of $2^h - 1 = \sqrt{n}$ vertices, and that both subgraphs ‘G^0 and ‘G^1 are almost half of ‘G.

Suppose that v_* is reachable from v_0 in G. We explain concretely our strategy to confirm this by identifying some directed path that witnesses the reachability from v_0 to v_*. Notice here that such a path p is divided into some x subpaths $p_1, \ldots, p_x$ such that the following holds for each $j \in [x-1]$ (*cf.* Figure 1(c)): (i) the end vertex w_j of p_j (that is the start vertex of p_{j+1}) is on S, and (ii) all inner vertices of p_j are in the same V^b for some $b \in \{0,1\}$, where V^0 and V^1 are respectively the set of vertices below and above the separator S. Then it is easy to see that we can find that w_1 is reachable from v_0 by searching vertices in S that are reachable from v_0 in $G[S \cup V^0]$. Next we can find that w_2 is reachable (from v_0) by searching vertices in S that are reachable in $G[S \cup V^1]$ from some vertex in S for which we know already its reachability; in fact, by the reachability from w_1 we can confirm that w_2 is reachable from v_0. Similarly, the reachability of $w_3, \ldots, w_{x-1}$ is confirmed, and then by considering the subgraph $G[S \cup V^1]$ we confirm that v_* is reachable from v_0 because it is reachable from w_{x-1}. This is our basic strategy. Note that the reachability in each subgraph can be checked recursively.

Since a path can cross the separator $\Theta(\sqrt{n})$ times, we cannot avoid making as many recursive calls at each level of recursion as there are vertices in the separator at that level. Consequently, without some modification the algorithm as described cannot hope to terminate in time bounded by some polynomial in n. In order to achieve polynomial-time computability, we introduce the idea of "budgeted recursion": the computation time allocated to individual recursive calls for checking the reachability in each subgraph is restricted in accordance with a predetermined sequence. Since the time required to trace a connecting path within an individual subgrid is not known in advance, we rely on a universality property of the sequence: eventually every subproblem will be allocated a budget sufficiently large to complete the required computation within that subproblem.

Asano and Kirkpatrick [3] describe the construction of a "universal sequence" suitable for this purpose. A similar universal sequence has been used in the context of oblivious algorithms; see [11], for example. Here we consider the following version.

For any $s \geq 0$, the *universal sequence* σ_s *order* s is defined inductively by

$$\sigma_s = \begin{cases} \langle 1 \rangle & \text{if } s = 0\text{, and} \\ \sigma_{i-1} \diamond \langle 2^i \rangle \diamond \sigma_{i-1} & \text{otherwise,} \end{cases}$$

where $\diamond$ signifies concatenation of sequences. By the definition, each element of the sequence is a power of 2. The length of the sequence σ_s is $2^{s+1} - 1$. For example, $\sigma_2 = \langle 1, 2, 1, 4, 1, 2, 1 \rangle$. We will use the following properties of universal sequences in the design and analysis of our algorithm. Their proof is straightforward, and is omitted here due to space constraints.

Lemma 1. *(a) The sequence $\sigma_s = \langle c_1, \ldots, c_{2^{s+1}-1} \rangle$ is* 2^s-universal *in the sense that for any positive integer sequence $\langle d_1, \ldots, d_x \rangle$ such that $\sum_{i \in [x]} d_i \leq 2^s$, there exists a subsequence $\langle c_{i_1}, \ldots, c_{i_x} \rangle$ of σ_s such that $d_j \leq c_{i_j}$ holds for all $j \in [x]$; (b) the sequence σ_s contains exactly 2^{s-i} appearances of the integer 2^i, for all $i \in [s]$, and nothing else; and (c) the sequence σ_s is computable in $O(2^s)$-time and $\widetilde{O}(1)$-space.*

Now we define our reachability algorithm following the strategy explained above. The technical key point here is to define a sequence of separators dividing subgraphs into two parts in a way that we can specify a current target subgraph succinctly. For this we introduce the notion of "cycle-separator." Intuitively, a cycle-separator S of a graph G is a set of cycles $S = \{C_1, \ldots, C_h\}$ that separates G into two subgraphs, those consisting of vertices located left (resp., right) of the cycles (including cycle vertices). In section 4 we outline how such a simple cycle-separator can be efficiently computed from any given separator. Based on this we have the following lemma.

Lemma 2. *There exists an $\widetilde{O}(\sqrt{n})$-space and polynomial-time algorithm (which we refer as* `CycleSep`*) that computes a cycle-separator S for a given undirected graph $G = (V, E)$ with its triangulated planar embedding. The size of the separator is at most $c_{\text{sep}}\sqrt{n}$. Furthermore, there is a way to define subsets V^0 and V^1 of V with the following properties: (a) $V^0 \cup V^1 = V$, $V^0 \cap V^1 = S$, and (b) $|V^b| \leq (2/3)|V| + c_{\text{sep}}\sqrt{n}$ for each $b \in \{0, 1\}$.*

Intuitively we can use cycle-separators like grid-separators to define a sequence of progressively smaller subgraphs of a given planar directed graph G. (Note that its underlying graph $`G$ is used for defining the subgraphs.) From technical reason[2], however, we need to add some edges to the outer faces of $`G[V^b]$ to get it triangulated under the current embedding. We can show that the number of added edges is bounded by $O(|S|)$ and that there is an algorithm `AddTri` that computes these edges and their planar embedding in $\widetilde{O}(|S|)$-space and polynomial-time. We refer to this information as an *additional triangulation edge list* T and consider it with a cycle-separator S and a Boolean label b. By $[`G]^b_{S,T}$ we mean both a graph obtained from $`G[V^b]$ by adding those triangulation edges specified by T and its planar embedding obtained by modifying the original planar embedding by T. In general, for any sequence $\mathbf{S} = \langle (b_1, S_1, T_1), \ldots, (b_t, S_t, T_t) \rangle$ of such triples of a label, a cycle-separator and an edge list, we define $[`G]_{\mathbf{S}}$ by

$$[`G]_{\mathbf{S}} \;=\; \left[ \cdots \left[ [`G]^{b_1}_{S_1,T_1} \right]^{b_2}_{S_2,T_2} \cdots \right]^{b_t}_{S_t,T_t},$$

which we call a *depth t subarea* of G. We should note here that it is easy to identify a depth t subarea by using $\mathbf{S}$; for a given $\mathbf{S}$, we can determine whether $v \in V$ is in the subarea $[`G]_{\mathbf{S}}$ by using only $O(\log n)$-space.

Armed with this method of constructing/specifying subareas we now implement our algorithm idea discussed above as a recursive procedure `ExtendReach` (see Algorithm 1). First we explain what it computes. We use global variables to keep the input graph G, the triangulated planar embedding of its underlying graph $`G$, the start vertex v_0, and the goal vertex v_*. As we will see in the next section, the triangulated planar embedding is $O(\log n)$-space computable. Hence, we can compute it whenever needed; thus, for simplicity we assume here that the embedding is given also as a part of the input. Note that for the space complexity this additional input data is not counted. On the other hand, we define global variables `A` and `R` that are kept in the work space. The variable `A` is for keeping grid-separator vertices that are currently considered; the vertices v_0 and v_* are also kept in `A`. The array `R` captures the reachability information for vertices in `A`; for any $v \in$ `A`, `R`$[v] =$ **true** iff the reachability of v from v_0 has been confirmed. Besides these data in the global variables, the procedure takes arguments $\mathbf{S}$ and ℓ, where $\mathbf{S}$ specifies the current subarea of $`G$ and $\ell$ is a bound on the length of path extensions. Our task is to update the reachability from $v_0$ for all vertices in `A` by using paths of length $\leq \ell$ in the current subarea. More precisely, the procedure `ExtendReach`$(\mathbf{S}, \ell)$ does the following: for each vertex $v \in$ `A`, it sets `R`$[v] =$ `true` if and only if there is a path to $v$ in $G[\mathtt{A} \cup V_{\mathbf{S}}]$ of length $\leq \ell$ from some vertex $u \in$ `A` whose value `R`$[u]$ before the execution equals `true`, where $V_{\mathbf{S}}$ is the set of vertices of the current subarea of $`G$ specified by $\mathbf{S}$. Since any vertex in G that is reachable from v_0 is reachable by a path of length at most $2^{\lceil \log n \rceil}$, the procedure `ExtendReach` can be used to determine

[2] The algorithm `CycleSep` is defined based on the separator algorithm of Lemma 4 that assumes a triangulated graph as input. Thus, in order to apply `CycleSep` to divide $`G[V^b]$ further, we need to get it triangulated.

the reachability of vertex v_* from v_0 as follows, which is our planar graph reachability algorithm: (1) Set $\mathtt{A} \leftarrow \{v_0, v_*\}$, $\mathtt{R}[v_0] \leftarrow \mathbf{true}$, and $\mathtt{R}[v_*] \leftarrow \mathbf{false}$; and (2) Execute $\mathtt{ExtendReach}(\langle\,\rangle, 2^{\lceil \log n \rceil})$, and then output $\mathtt{R}[v_*]$.

Next we give some additional explanation concerning Algorithm 1. Consider any execution of ExtendReach for given arguments $\mathbf{S}$ and ℓ (together with data kept in its global variables). Let $V_{\mathbf{S}}$ be the set of vertices of the subarea $[`G]_{\mathbf{S}}$ specified by $\mathbf{S}$. There are two cases. If $V_{\mathbf{S}}$ has less than $144c_{\mathrm{sep}}^2$ vertices, then the procedure updates the value of R in a straightforward way. As we will see later $G[\mathtt{A} \cup V_{\mathbf{S}}]$ has at most $O(\sqrt{n})$ vertices; hence, we can use any standard linear-space and polynomial-time algorithm (e.g., breadth-first search) to do this task. Otherwise, ExtendReach divides the current subarea $[`G]_{\mathbf{S}}$ into two smaller subareas with new separator vertex set S'_{t+1} that is added to A. It then explore two subareas by using numbers in the universal sequence σ_s to control the length of paths in recursive calls.

The correctness of Algorithm 1 is demonstrated in Lemma 3 below. From this, as summarized in Theorem 1, it is clear that our algorithm correctly determines the reachability of vertex v_* from vertex v_0 in the input graph G.

Lemma 3. *For any input instance G, v_0, and v_* of the planar graph reachability problem, consider any execution of* $\mathtt{ExtendReach}(\mathbf{S}, \ell)$ *for some* $\mathbf{S} = \langle (b_1, S_1, T_1), \ldots, (b_t, S_t, T_t) \rangle$ *and $\ell = 2^s$. Let $V_{\mathbf{S}}$ denote the set of vertices of $[`G]_{\mathbf{S}}$. For each vertex v that is in* A *before the execution,* $\mathtt{R}[v]$ *is set to* **true** *during the execution if and only if there is a path to v in $G[\mathtt{A} \cup V_{\mathbf{S}}]$ of length at most 2^s, from some vertex $u \in \mathtt{A}$ whose value* $\mathtt{R}[u]$ *before the execution equals* true.

Proof. Suppose that there is a path $p = u_0, u_1, \ldots, u_h$ in $G[\mathtt{A} \cup V_{\mathbf{S}}]$, where (i) u_0 and u_h both belong to A, (ii) $h \leq 2^s$, and (iii) $\mathtt{R}[u_0] = \mathtt{true}$ before executing the procedure. For the lemma, it suffices to show that $\mathtt{R}[u_h]$ is set **true** during the execution of the procedure.

We prove our assertion by induction on the size of $V_{\mathbf{S}}$. If $|V_{\mathbf{S}}| \leq 144c_{\mathrm{sep}}^2$, then it is clear from the description of the procedure. Consider the case where $|V_{\mathbf{S}}| > 144c_{\mathrm{sep}}^2$. Then the part from line 7 of the procedure is executed. Let S_{t+1}, T_{t+1}^0, and T_{t+1}^1 be the separator and the edge lists computed there. For any $b \in \{0, 1\}$, let $\mathbf{S}^b$ denote $\langle (b_1, S_1, T_1), \ldots, (b_t, S_t, T_t), (b, S_{t+1}, T_{t+1}^b) \rangle$; also let $`G^b = [`G]_{\mathbf{S}^b}$ $(= [[`G]_{\mathbf{S}}]_{(b, S_{t+1}, T_{t+1}^b)})$ and $V^b = \mathrm{V}(`G^b) \setminus S_{t+1}$.

We observe that p can be decomposed into some number $x \leq |\mathtt{A} \cup S_{t+1}| - 1$ of subpaths $p_1, p_2, \ldots, p_x$, such that (i) both $\mathrm{start}(p_j)$ and $\mathrm{end}(p_j)$ belong to $\mathtt{A} \cup S_{t+1}$ for each $j \in [x]$, (ii) the internal vertices of p_j belong either to V^0 or V^1 for each $j \in [x]$, and (iii) $\mathrm{end}(p_j) = \mathrm{start}(p_{j+1})$ for each $j \in [x-1]$, where by $\mathrm{start}(p_j)$ and $\mathrm{end}(p_j)$ we mean the start and end vertices of p_j respectively. Let h_j denote the number of edges in path p_j. By construction (i) $h_j \geq 1$ for all $j \in [x]$, and (ii) $\sum_{j \in [x]} h_j = h \leq 2^s$. Then by Lemma 1(a), the sequence $\langle h_1, \ldots, h_x \rangle$ is dominated by the universal sequence $\sigma_s = \langle c_1, \ldots, c_{2^{s+1}} \rangle$. That is, there exists a subsequence $\langle c_{k_1}, c_{k_2}, \ldots c_{k_x} \rangle$ of σ_s such that $h_j \leq c_{k_j}$ for all $j \in [x]$. Thus, for any $j \in [x]$, if $\mathtt{R}[\mathrm{start}(p_j)] = \mathbf{true}$ before the execution of $\mathtt{ExtendReach}(\mathbf{S}^0, c_{k_j})$,

Algorithm 1. ExtendReach($\mathbf{S}, \ell$)

```
Given: (as arguments) A sequence $\mathbf{S} = \langle (b_1, S_1, T_1), \ldots, (b_t, S_t, T_t) \rangle$ of triples of a
    binary label, a cycle-separator, and an additional triangulation edge list, and a
    bound $\ell = 2^s$ on the length of path.
    // In this description we use $V_{\mathbf{S}}$ to denote the set of vertices of $[`G]_{\mathbf{S}}$.
    (as global variables) The input graph $G$, its triangulated planar embedding, the
    source vertex $v_0$, the goal vertex $v_*$, a set A of the currently considered vertices,
    and a Boolean array R specifying known reachability from $v_0$, for all $v \in$ A.
Task: For each vertex $v \in$ A, set R$[v]$ = true if there is a path to $v$ in $G[$A $\cup V_{\mathbf{S}}]$ of
    length at most $2^s$ from some vertex $u \in$ A whose value R$[u]$ before the current
    procedure execution equals true.
    // Invariant: A $= \{v_0, v_*\} \cup \bigcup_{i \in [t]} S_i$. R$[v]$ = true $\Rightarrow$ $v$ is reachable from $v_0$ in $G$.
 1: if the number of vertices of $V_t$ is less than $144c_{\mathrm{sep}}^2$ then
 2:     $R_t \leftarrow \{u \in$ A : R$[u]$ = true$\}$;
 3:     for each vertex $v \in$ A do
 4:         R$[v] \leftarrow$ true iff $v$ is reachable from some $u \in R_t$ in $G[$A $\cup V_{\mathbf{S}}]$ by a path of
            length $\leq \ell$;  // Use any linear space and polynomial-time algorithm here.
 5:     end for
 6: else
 7:     Use CycleSep and AddTri to create a new cycle separator $S_{t+1}$ of $[`G]_{\mathbf{S}}$ and its
        additional triangulation edge lists $T_{t+1}^0$ and $T_{t+1}^1$;
 8:     $S'_{t+1} \leftarrow S_{t+1} \setminus$ A;   A $\leftarrow$ A $\cup S'_{t+1}$;
 9:     R$[v] \leftarrow$ false for each vertex $v \in S'_{t+1}$;
10:     for each $c_i = 2^{s_i}$ in the universal sequence $\sigma_s$ (where $i \in [2^{s+1} - 1]$) do
11:         ExtendReach($\langle (b_1, S_1, T_1), \ldots, (b_2, S_t, T_t), (0, S_{t+1}, T_{t+1}^0) \rangle, c_i$);
12:         ExtendReach($\langle (b_1, S_1, T_1), \ldots, (b_2, S_t, T_t), (1, S_{t+1}, T_{t+1}^1) \rangle, c_i$);
13:     end for
14:     A $\leftarrow$ A $\setminus S'_{t+1}$;
15: end if
```

then we have R$[\mathrm{end}(p_j)]$ = **true** after the execution because of the induction hypothesis. Hence, by executing the fragment:

ExtendReach($\mathbf{S}^0, c_{k_1}$); ExtendReach($\mathbf{S}^1, c_{k_1}$); $\cdots$ ExtendReach($\mathbf{S}^0, c_{k_{x-1}}$); ExtendReach($\mathbf{S}^1, c_{k_{x-1}}$); ExtendReach($\mathbf{S}^0, c_{k_x}$); ExtendReach($\mathbf{S}^1, c_{k_x}$);

we have R$[\mathrm{end}(p_x)]$ = true since R$[\mathrm{start}(p_1)]$ = true by our assumption. Therefore, R$[u_h]$ (where $u_h = \mathrm{end}(p_x)$) is set **true** as desired since the above fragment must be executed as a part of the execution of line 10–13 of the procedure. $\square$

By analyzing the time and space complexity of our algorithm, we conclude as follows.

Theorem 1. *For any input instance G, v_0, and v_* of the planar directed graph reachability problem (where n is the number of vertices of G), our planar graph reachability algorithm determines whether there is a path from v_0 to v_* in G in $\widetilde{O}(\sqrt{n})$ space and polynomial-time.*

Proof. The correctness of the algorithm follows immediately from Lemma 3. For the complexity analysis, we consider the essential part, that is, the execution of $\mathtt{ExtendReach}(\langle\,\rangle, 2^{\lceil \log n \rceil})$.

As a key step for estimating the space and time bounds, we show here a bound on the depth of recursion during the execution. Let $t_{\max}$ denote the maximum depth of recursive calls in the execution, for which we would like to show that $t_{\max} \le 2.5 \log n$ holds. Consider any depth t recursive call of ExtendReach; in other words, the execution of $\mathtt{ExtendReach}(\mathbf{S}, \ell)$ with a sequence $\mathbf{S}$ of length t. (Thus, the inital call of ExtendReach is regarded as depth 0 recursive call.) Here some depth t subarea $[`G]_{\mathbf{S}}$ is examined; let $n_t$ be the number of vertices of this subarea. Assume that $n_t \ge 144c_{\rm sep}^2$. Then two smaller subareas of $[`G]_{\mathbf{S}}$ are created and ExtendReach is recursively executed on them. Let n_{t+1} denote the number of vertices of a lager one of these two smaller subareas. Then by Lemma 2 we have $n_{t+1} \le 2n_t/3 + c_{\rm sep}\sqrt{n_t} \le 3n_t/4$ since $n_t \ge 144c_{\rm sep}^2$. Hence, $t_{\max}$ is bounded by $2.5 \log n$ as desired because $n(3/4)^{2.5 \log n} < 144c_{\rm sep}^2$.

We bound the memory space used in the execution $\mathtt{ExtendReach}(\langle\,\rangle, 2^{\lceil \log n \rceil})$. For this, it is enough to bound the number of vertices in $\mathtt{A}$ because the number of words needed to keep in the work memory space during the execution is proportional to $|\mathtt{A}|$. Note further that $\mathtt{A} = \{v_0, v_*\} \cup \bigcup_{i \in [t]} S_i$ at any depth t recursive call of ExtendReach. On the other hand, by using the above notation, it follows from the above and Lemma 2 we have

$$\begin{aligned} |\mathtt{A} \setminus \{v_0, v_*\}| &\le \sum_{i \in [t_{\max}]} |S_i| \;\le\; \sum_{i \in [t_{\max}]} c_{\rm sep}\sqrt{n_i} \\ &\le \sum_{i \in [t_{\max}]} c_{\rm sep}\sqrt{\left(\frac{3}{4}\right)^{i-1} n} \;\le\; \left(c_{\rm sep}\sqrt{n}\right)\left(\sum_{i \ge 0}\left(\frac{3}{4}\right)^{i/2}\right) \;=\; O(\sqrt{n}), \end{aligned}$$

which gives us the desired space bound.

For bounding the time complexity by some polynomial, it suffices to show that the total number of calls of ExtendReach is polynomially bounded. To see this, we estimate $N(t, 2^s)$, the max. number of calls of ExtendReach during any depth t recursive call of $\mathtt{ExtendReach}(\mathbf{S}, 2^s)$ that occurs in the execution of $\mathtt{ExtendReach}(\langle\,\rangle, 2^{\lceil \log n \rceil})$. (Precisely speaking, $N(t, 2^s) = 0$ if no call of type $\mathtt{ExtendReach}(\mathbf{S}, 2^s)$ occurs.) Clearly, $N(t_{\max}, 2^s) = 0$ for any s. Also it is easy to see that $N(t, 2^0) = 2 + 2N(t+1, 2^0)$ for any $t < t_{\max}$; hence, we have $N(t, 2^0) \le 2 \cdot (2^{t_{\max}-t} - 1) \le 2^{t_{\max}-t+1}$. Consider any $t < t_{\max}$ and $s \ge 1$. From the description of ExtendReach and the property of the universal sequence σ_s (Lemma 1(b)), we have

$$N(t, 2^s) \;=\; 2\sum_{i \in [2^{s+1}]} (1 + N(t+1, c_i)) \;=\; 2^{s+2} + \sum_{0 \le j \le s} 2^{s-j} N(t+1, 2^j),$$

from which we can derive $N(t, 2^s) = 2N(t+1, 2^s) + 2N(t, 2^{s-1})$. Then by induction we can show

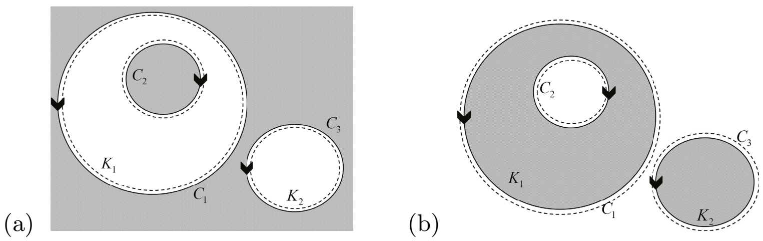
$$N(t, 2^s) \;\le\; 2^{t_{\max}-t+s+1}\binom{t_{\max}-t+s}{s}.$$

Thus, $N(0, 2^{\lceil \log n \rceil})$, the total number of calls of `ExtendReach` is polynomially bounded. □

4 Cycle-Separators

We expand here the notion of a cycle-separator and Lemma 2 used in the previous section. Throughout this section, we consider only undirected graphs. In particular, we fix any sufficiently large planar undirected graph $G = (V, E)$ and discuss a cycle-separator for G; all symbols using G are used to denote some graph related to G.

Roughly speaking, a cycle-separator is a separator S consisting of cycles. In this paper, we assume some orientation for each cycle, and our cycle-separators is required to separate G into two subgraphs by considering a part located left (resp., right) of cycles (*cf.* Figure 2).



S consists of cycles C_1, C_2, C_3, each of which has an orientation specified as in the figure. Dashed lines indicate the cuts corresponding to edges located in the (a) left and (b) right of each cycle. These cuts are used to identify $G[V^0]$ and $G[V^1]$

Fig. 2. An example of a cycle-separator

Recall that we do not assume that our input graph comes equipped with a planar embedding. This is unnecessary for our purposes since Allender and Mahajan [1] showed that the problem of computing a planar embedding can be reduced to the undirected graph reachability problem. Hence, by using the algorithm `UReach` of Reingold, we can compute a planar embedding of G by using $O(\log n)$-space. Also it is also easy to to obtain some triangulation w.r.t. this embedding, and thus we may assume an $O(\log n)$-space algorithm that computes a triangulated planar embedding for a give planar graph. In our reachability algorithm, this $O(\log n)$-space algorithm is used (implicitly) before starting the actual computation and so in the description that follows we proceed as though our input graph G is given with some triangulated planar embedding.

The Planar Separator Theorem guarantees that every planar graph has a separator of size $O(\sqrt{n})$ that disconnects a graph into two subgraphs each of which has at most $2n/3$ vertices, which we call a $2/3$-*separator*. Imai et al. has shown an algorithm that computes a $2/3$-*separator* by using $O(\sqrt{n})$-space and in polynomial-time. Though we use such a separator algorithm as a blackbox, we introduce some modification so that we can specify *two* subgraphs disconnected by a separator in order to use them in the context of sublinear-space computation. A *labeled-separator* of G is a pair of a separator S and a set $\tau = \{v_1, \ldots, v_k\}$ of vertices of G (which we simply denote by $S\tau$) such that no two vertices of τ belong to the same connected component of $G[V \setminus S]$. Graphs $G^0_{S\tau}$ and $G^1_{S\tau}$ are two disconnected subgraphs of $G[V \setminus S]$ defined by $S\tau$; $G^0_{S\tau} = \bigcup_{i=1}^{k} K_i$ where each K_i is the connected component of $G[V \setminus S]$ containing v_i, and $G^1_{S\tau}$ is a subgraph of G consisting of all the other connected components of $G[V \setminus S]$. By the planarity, we can show that $G[V \setminus S]$ has at most $2|S| - 4$ connected components. (Recall that we assumed G is triangulated and hence connected.) Thus, each labeled-separator can be stored in $\widetilde{O}(|S|)$-space. Furthermore, by using `UReach`, we can identify, for each $v_i \in \tau$, the connected component K_i containing v_i in $O(\log n)$-space. Since counting is also possible in $O(\log n)$-space, for a given 2/3-separator, we can in fact collect connected components $K_1, \ldots, K_k$ of $G[V \setminus S]$ (and their representative vertices $v_1, \ldots, v_k$) so that $|\mathrm{V}(G^0_{S\tau})| \le 2|V|/3$ and $|\mathrm{V}(G^1_{S\tau})| \le 2|V|/3$ hold with $\tau = \{v_1, \ldots, v_k\}$. In summary, we have the following separator algorithm that is the basis of our cycle-separator algorithm.

Lemma 4. *There exists an $\widetilde{O}(\sqrt{n})$-space and polynomial-time algorithm that yields a 2/3-labeled-separator of size $\le c_{\mathrm{sep}}\sqrt{n}$ for a given planar graph, where c_{sep} is some constant, which has been used in the previous section.*

Recall that we assume some planar embedding of G; the following notions are defined with respect to this embedding. For any cycle C of G, we use a sequence $\langle u_1, \ldots, u_r \rangle$ of vertices of G in the order of appearing in C under one direction. We call such a sequence as a *cycle representation.* With this orientation, we define the left and the right of the cycle C. Our main technical lemma (see a technical report version [ECCC, TR14-071] for the full proof) is to show a way to compute a set S' of cycles from a given separator S that can be used as a separator in $\widetilde{O}(|S|)$-space and polynomial-time. More specifically, by using cycles in S', we define two subsets V^0 and V^1 of V as sets of vertices respectively located left and right of the cycles. Then they satisfy Lemma 2; that is, $G[V^0]$ and $G[V^1]$ are subgraphs covering G and sharing only vertices in S', which corresponds to 'G^0, 'G^1, and the separator S in the grid case. Furthermore, their size is (approximately) bounded by $2n/3$, and since $S' \subseteq S$ as a set, we have $|S'| \le c_{\mathrm{sep}}\sqrt{n}$. This S' is called a *cycle-separator* in this paper. We also provide a way to identify graphs $G[V^0]$ and $G[V^1]$, which is used as a basis of an algorithm identifying $[\text{'}G]_{\mathbf{S}}$.

5 Conclusion

It should be noted that, though restricted to grid graphs, the problem studied by Asano et al. in [2,3] is the shortest path problem, a natural generalization of the reachability problem. In order to keep the discussion in this paper as simple as possible, and focus on the key ideas, we have restricted our attention here to the graph reachability problem. However, it is not hard to see that our algorithm for reachability can be modified to the shortest path problem (with a modest increase in the polynomial time bound).

Similarly, the focus in Asano et al. in [2,3] is on space efficient and yet practically useful algorithms, including time-space tradeoffs. In this paper, on the other hand, our motivation has been in extending a graph class that is solvable in $\widetilde{O}(\sqrt{n})$-space and polynomial-time, and the specific time complexity of algorithms is not so important so long as it is within some polynomial. In fact, since the algorithm of Reingold for the undirected reachability is used heavily, we need very large polynomial to bound our algorithm's running time.

Since we use Reingold's undirected reachability algorithm, our algorithm (and also the one by Imai et al.) have no natural implementation in the NNJAG model. While the worst-case instances for NNJAG given in [5] are non-planar, it is an interesting question whether we have similar worst-case instances based on some planar directed graphs. A more important and challenging question is to define some model in which our algorithm can be naturally implemented and show some limitation of space efficient computation.

References

1. Allender, E., Mahajan, M.: The complexity of planarity testing. Information and Computation 189(1), 117–134 (2004)
2. Asano, T., Doerr, B.: Memory-constrained algorithms for shortest path problem. In: Proc. of the 23th Canadian Conf. on Comp. Geometry, CCCG 1993 (2011)
3. Asano, T., Kirkpatrick, D.: A $O(\sqrt{n})$-space algorithm for reporting a shortest path on a grid graph (in preparation)
4. Barnes, G., Buss, J.F., Ruzzo, W.L., Schieber, B.: A sublinear space, polynomial time algorithm for directed s-t connectivity. In: Proc. Structure in Complexity Theory Conference, pp. 27–33. IEEE (1992)
5. Edmonds, J., Poon, C.K., Achlioptas, D.: Tight lower bounds for st-connectivity on the NNJAG model. SIAM J. Comput. 28(6), 2257–2284 (1999)
6. Gazit, H., Miller, G.L.: A parallel algorithm for finding a separator in planer graphs. In: Proc. of the 28th Ann. Sympos. on Foundations of Comp. Sci. (FOCS 1987), pp. 238–248 (1987)
7. Henzinger, M.R., Klein, P., Rao, S., Subramanian, S.: Faster shortest-path algorithms for planar graphs. Journal of Comput. Syst. Sci. 55, 3–23 (1997)
8. Imai, T., Nakagawa, K., Pavan, A., Vinodchandran, N.V., Watanabe, O.: An $O(n^{\frac{1}{2}+\epsilon})$-space and polynomial-time algorithm for directed planar reachability. In: Proc. of the 28th Conf. on Comput. Complexity, pp. 277–286 (2013)
9. Miller, G.L.: Finding small simple cycle separators for 2-connected planar graphs. J. Comput. Syst. Sci. 32(3), 265–279 (1986)
10. Lipton, R.J., Tarjan, R.E.: A separator theorem for planar graphs. SIAM Journal on Applied Mathematics 36(2), 177–189 (1979)
11. Pippenger, N., Fischer, M.J.: Relations among complexity measures. J. ACM 26(2), 361–381 (1979)
12. Reingold, O.: Undirected connectivity in log-space. J. ACM 55(4) (2008)
13. Wigderson, A.: The complexity of graph connectivity. In: Havel, I.M., Koubek, V. (eds.) MFCS 1992. LNCS, vol. 629, pp. 112–132. Springer, Heidelberg (1992)