Vol.2013-AL-144 No.25
2013/5/18

An $O(n^{\frac{1}{2}+\epsilon})$-Space and Polynomial-Time Algorithm for Directed Planar Reachability

Imai Tatsuya[1,a)] Nakagawa Kotaro[1,b)] Vinodchandran N.V.[3,c)] Pavan A.[2,d)]
Watanabe Osamu[1,e)]

Abstract: We show that the reachability problem over *directed planar graphs* can be solved simultaneously in polynomial time and approximately $O(\sqrt{n})$ space. In contrast, the best space bound known for the reachability problem on general directed graphs with polynomial running time is $O(n/2^{\sqrt{\log n}})$.

1. Introduction

The *graph reachability problem* is to decide, for a given graph G and its two vertices s and t, whether there is a path from s to t in G. This problem is central to space bounded computations. Different versions of this problem characterize several important space complexity classes. The most standard one for directed graphs, the *directed graph reachability problem*, is the canonical complete problem for non-deterministic log-space (NL). The breakthrough result of Reingold implies that the *undirected graph reachability problem* characterizes the complexity of deterministic log-space (L) [16]. It is also known that certain restricted promise versions of the directed reachability problem characterize randomized log-space computations (RL) [17]. Thus progress in understanding the space complexity of graph reachability problems directly relates to progress in space complexity studies. We refer the readers to a (two decades old, but excellent) survey by Wigderson [19] and a recent update by Allender [3], to further understand the significance of reachability problems in complexity theory. Clearly, designing efficient deterministic algorithms for reachability problems is a major concern of complexity theory.

The standard breadth first search algorithm (denoted as BFS throughout this paper) and Savitch's algorithm are two of the most fundamental algorithms known for solving directed graph reachability. BFS can be implemented in linear time and space. Savitch's algorithm only takes $O(\log^2 n)$ space but requires $\Theta(n^{\log n})$ time. BFS is efficient in time but not in space and Savitch's algorithm is efficient in space but takes super-polynomial time. Hence a natural and significant question that researchers have considered is whether we can design an algorithm for the directed graph reachability problem that is efficient in both time and space. In particular, can we design a polynomial-time algorithm for the directed graph reachability problem that uses only $O(n^{1-\epsilon})$ space for some small constant ϵ? The best known result in this direction is the two decades old bound due to Barnes, Buss, Ruzzo, and Schieber [5]. By cleverly combining BFS and Savitch's algorithm, Barnes et al. designed a polynomial-time algorithm for reachability that uses $O(n/2^{\sqrt{\log n}})$ space. Note that the space bound that Barnes et al. establish is only slightly sublinear. Improving this bound remains a significant open question. There are indications that it may be difficult to improve this bound because there are matching lower bounds known for solving reachability on certain model known as NNJAG [6], [7], [15]. All the known algorithms for the general reachability problem can be implemented in NNJAG without significant blow up in time and space.

However, for certain restricted classes of directed graphs better results are known. For example, in [18] it is shown that for any ϵ, the reachability problem for directed acyclic graphs with $O(n^{1-\epsilon})$ sources and embedded on surfaces with $O(n^{1-\epsilon})$ genus can be solved in polynomial time and $O(n^{1-\epsilon})$ space. For reachability (in fact for the shortest path problem) on *grid graphs* Asano and Doerr [2] gave a polynomial-time algorithm that takes only $O(n^{\frac{1}{2}+\epsilon})$ space. Asano and Doerr left open the question whether for planar graphs such a bound can be achieved. Note that even though directed planar reachability reduces to directed reachability in grid graphs [1], the reduction blows up the size of the graph by a polynomial factor and hence it is not useful in this setting.

In this paper we present a polynomial-time reachability algorithm for *directed planar graphs* that achieves $O(n^{1/2+\epsilon})$-space bound. We prove the following theorem.

Theorem 1. *For any constant $0 < \epsilon < 1/2$, there is an algorithm that, given a directed planar graph G and two vertices s and t, decides whether there is a path from s to t. This algorithm runs in time $n^{O(1/\epsilon)}$ and uses $O(n^{1/2+\epsilon})$ space, where n is the number of*

1 Dept. of Math. and Computer Science, Tokyo Inst. of Technology, Japan
2 Dept. of Computer Science, Iowa State University, U.S.A
3 Dept. of Comp. Sci. & Eng., Univ. of Nebraska-Lincoln, Lincoln, U.S.A
a) imai7@is.titech.ac.jp
b) nakagaw1@is.titech.ac.jp
c) vinod@cse.unl.edu
d) pavan@cs.iastate.edu
e) watanabe@is.titech.ac.jp

vertices of G.

For proving the above theorem we first give a polynomial-time and $\widetilde{O}(\sqrt{n})$-space algorithm for computing a "separator" of $O(\sqrt{n})$ size for an undirected planar graph (By $\widetilde{O}(s(n))$ we mean $O(s(n)(\log n)^{O(1)})$). This result may be of independent interest and we state that here as a theorem.

Theorem 2. *There is an algorithm that takes an undirected planar graph G with n vertices as input and outputs a (8/9)-separator of G. This algorithm runs in polynomial time and uses $\tilde{O}(\sqrt{n})$ space.*

We call the above algorithm `Separator`. Our reachability algorithm uses algorithm `Separator`. It is important to note that though our reachability algorithm works on a directed planar graph, for constructing the separator we work on the underlying undirected graph. For a given directed planar graph G, the reachability algorithm first applies `Separator` to its underlying undirected graph and computes a small size "separator family" S — a set of vertices, removal of which splits G into several disconnected subgraphs of smaller size. The crucial observation is that any path in G from one component to another has to go through S. Based on this, we construct a new directed graph H on S. There is a directed edge (a, b) in H if there is a path from a to b that goes through a single component. This can be decided by solving reachability on a smaller planar graph. This implies that reachability in G reduces to reachability in H. Since H is a smaller graph we can afford to perform `BFS` on H. This leads to a polynomial-time, $O(n^{2/3})$-space algorithm. To obtain the required $n^{1/2+\epsilon}$-space bound we apply this idea recursively. By ensuring that the depth of recursion is a constant, we can ensure that the running time of the algorithm remains polynomial[*1]. The idea of applying separator algorithms for solving path problems in planar graphs is well explored, especially in the context of time complexity (see for example [8], [10]).

The well-known planar separator algorithm due to Lipton and Tarjan [13] runs in linear time, but requires linear space. This is because a crucial step in their algorithm is a BFS tree computation of the input graph. Computing a BFS tree of a graph (directed or undirected) is as difficult as the directed graph reachability problem. Our algorithm to construct a separator is based on ideas from the parallel planar separator algorithm of Gazit and Miller [9]. This algorithm, instead of constructing a single BFS tree, constructs a collection of smaller BFS trees. They introduced this approach for parallel computation, but it is also useful for designing a space efficient algorithm. We design our algorithm by combining this approach with ideas from [11], [12], [13], [14].

The rest of the paper is organized as follows. In the next section we introduce basic definitions and notation that we use. In Section 3, we present the algorithm for reachability. In Section 4 we present our algorithm for computing a separator for a given undirected planar graph.

2. Preliminaries

We will use the standard notions and notation for algorithms, complexity measures, and graphs without defining them. Throughout this paper, for any set X, $|X|$ denotes the number of elements in X. By log we mean the base 2 logarithm.

Although we are given a directed graph for the reachability problem, we often consider its underlying undirected graph as an input to some procedures. Thus, while $G = (V, E)$ is usually used to denote an input graph, it can be either an directed or an undirected graph; their distinction should be clear from the context. When necessary, for a directed graph G, by $\underline{G}$ we mean its underlying undirected graph. We use n to denote the number of vertices of an input graph G; on the other hand, we sometimes use $\widehat{n}$ to denote the number of vertices of a graph $\widehat{G}$ considered in each context. For any $U \subseteq V$, let $G(U)$ (resp., $\underline{G}(U)$) denote the subgraph of G (resp., $\underline{G}$) induced by U.

For discussing the complexity of algorithms, we follow the standard computation model. In particular, for discussing sublinear space complexity, we consider a computation model in which an input is present on some read-only outside memory, an output is produced on some write-only outside memory, and only internal memory space is sublinearly bounded.

Our algorithms heavily depend on the $O(\log n)$ space (and polynomial time) algorithm of Reingold [16] for the undirected graph reachability problem. We denote this algorithm by `UReach`. Also as shown by Allender and Mahajan [4], we have some $O(\log n)$-space algorithm that tests whether a given undirected graph is planar and produces (if it is planar) its combinatorial planar embedding (i.e., the order of edges adjacent to each vertex in some planar embedding).

The notion of separator is central throughout this paper. Here we define this notion formally.

Definition 1. For any undirected graph G and for any constant ρ, $0 < \rho < 1$, a subset of vertices S is called a *ρ-separator* if (i) removal of S disconnects G into two subgraphs A and B, and (ii) the number of vertices of any component is at most ρn. The *size* of a separator is the number of vertices in the separator.

It is well known that every planar graph with n vertices has a (2/3)-separator of size $O(\sqrt{n})$ [13].

3. Reachability algorithm given a separator

In this section we present our reachability algorithm that uses the space efficient separator algorithm (called `Separator`) given in Theorem 2. First we define an algorithm that uses `Separator` iteratively to compute a separator family that splits a given graph into sublinear size components.

Definition 2. For any undirected graph G with n vertices, an *$r(n)$-separator family* of G is a set $\overline{S}$ of vertices of G so that the removal of $\overline{S}$ disconnects G into subgraphs each of which contains at most $r(n)$ vertices.

Although we use the term "separator family", a separator family is just a set of vertices, and hence, the *size* of a separator family is simply the number of vertices in the set.

By using `Separator`, for any $\epsilon > 0$, we can design an algorithm that produces an $n^{1-\epsilon}$-separator family of $O(n^{1/2+\epsilon/2})$ size in polynomial time and $\widetilde{O}(n^{1/2+\epsilon/2})$ space.

Lemma 3. *For any $\epsilon > 0$, there is an algorithm* `SepFamily` *that takes a planar graph as input and outputs an $n^{1-\epsilon}$-separator family of size $O(n^{1/2+\epsilon/2})$ in polynomial time and $\widetilde{O}(n^{1/2+\epsilon/2})$ space.*

[*1] We note that this approach also leads to a space efficient algorithm for computing shortest paths in planar graphs.

Proof. Our key tool is the separator algorithm `Separator` of Theorem 2. For a given undirected planar graph of size n, this algorithm produces a (8/9)-vertex separator of size at most $c\sqrt{n}$ (for some constant $c > 0$); that is, removal of vertices in the separator disconnects the graph into two subgraphs of size $\leq 8n/9$. Intuitively, we repeat this procedure for some sufficient number of times so that each connected component contains at most $n^{1-\epsilon}$ vertices. The union of separators increases monotonically, and, as we will see, its size is bounded by $O(n^{1/2+\epsilon/2})$. This whole procedure can be implemented by using $\widetilde{O}(n^{1/2+\epsilon/2})$ space for keeping the obtained separators.

Let us examine this idea in more detail. Fix any input undirected graph $G = (V, E)$ with n vertices. We first define an algorithm `Separator`$^+$ so that it can be used iteratively. This algorithm takes G, a parameter i indicating the number of iterations, and a separator family $\overline{S}_i$ obtained so far, and it outputs a new separator family $\overline{S}_{i+1} := \overline{S}_i \cup \overline{S}'$. Here we may assume (by induction hypothesis) that each connected component of $G(V \setminus \overline{S}_i)$ is of size $\leq n_i := (8/9)^i n$. The set $\overline{S}'$ added by the algorithm is the union of vertex separators $S'_1, ..., S'_\ell$ of size $c\sqrt{n_i}$. Each S'_j is obtained by applying `Separator` to a connected component of $G(V \setminus \overline{S}_i)$ of size $> (8/9)n_i$. This guarantees the induction hypothesis on the size of connected components. More specifically, for every vertex $v \in V \setminus \overline{S}_i$, the algorithm first checks whether v is the vertex with the smallest index in the component G_v connected to v in $G(V \setminus \overline{S}_i)$, and if so and if the size of G_v is larger than $(8/9)^{i+1}n$, then `Separator` is applied to this component to produce a vertex separator S'_j for G_v.

We apply this algorithm `Separator`$^+$ for k times, where k is defined by

$$k = \left\lceil \frac{\epsilon}{\log(9/8)} \log n \right\rceil$$

so that

$$\frac{1}{2} n^{1-\epsilon} \leq n\left(\frac{8}{9}\right)^k \leq n^{1-\epsilon}$$

holds. Thus, after the k applications of `Separator`$^+$, the size of connected components is at most $n^{1-\epsilon}$ as desired.

Next we bound the size of the obtained separator family. Note that there are at most $(9/8)^{i+1}$ components of size $\geq (8/9)^{i+1}n$; hence, there are at most $(9/8)^{i+1}$ vertex separators added to $\overline{S}_i$ at the ith iteration, and each such vertex separator size is at most $c\sqrt{n_i} = c\sqrt{(8/9)^i n}$. Thus, the number of vertices in the final separator family $\overline{S}_{k+1}$ is bounded by

$$\left(\frac{9}{8}\right) c\sqrt{n} + \left(\frac{9}{8}\right)^2 c\sqrt{\left(\frac{8}{9}\right) n} + \cdots + \left(\frac{9}{8}\right)^{k+1} c\sqrt{\left(\frac{8}{9}\right)^k n}$$
$$= \frac{9c\sqrt{n}}{8} \sum_{i=0}^{k} \left(\frac{9}{8}\right)^{k/2} \leq c'\sqrt{n}\left(\frac{9}{8}\right)^{k/2} \leq 2c' n^{1/2+\epsilon/2}$$

for some constant $c' > 0$. Finally, observe that the space used is dominated by the the space required to store the separator family. It is easy to see that the running time is bounded by a polynomial. □

Now we are ready to state our algorithm for the directed planar graph reachability problem. Consider any constant $\epsilon > 0$. We may assume that $\epsilon < 1/2$, because otherwise our target bounds, e.g., $\widetilde{O}(n^{1/2+\epsilon})$ become trivial. Let $G = (V, E)$, s, and t be the given input; that is, G is a directed graph, and s and t are start and goal vertices in V. As outlined in the introduction, our algorithm first uses `SepFamily` to compute an $n^{1-\epsilon}$-separator family $\overline{S}$ of size $O(n^{1/2+\epsilon/2})$ for the underlying undirected graph $\underline{G}$, and explores a path from s to t through vertices in $\overline{S}$. For any vertices a and b in $\overline{S}$, the reachability from a to b is determined by the reachability in $G(V' \cup \overline{S})$, where V' is the set of vertices of some connected component of $\underline{G}(V \setminus \overline{S})$ that is adjacent to both a and b in $\underline{G}(V' \cup \overline{S})$. (Thus, we can immediately determine that b is not reachable from a if there is no such connected component V' of $\underline{G}(V \setminus \overline{S})$.) For checking this connectivity, we may use the standard algorithm `BFS` for the reachability problem if $G(V' \cup \overline{S})$ is small enough. Although Lemma 3 guarantees that $|V' \cup \overline{S}| = O(n^{1-\epsilon})$, it is still large to execute `BFS`. Instead we use our algorithm recursively on $G(V' \cup \overline{S})$. Note further that $|V'|$ is too large to store; thus, V' (and hence, $G(V' \cup \overline{S})$) must be given implicitly. In the algorithm, we specify V' as a set of vertices connected to some vertex in $\underline{G}(V \setminus \overline{S})$; then we can check whether $v \in V'$ for a given v by using the undirected graph reachability algorithm `UReach` in $O(\log n)$ space and polynomial time. Hence, the algorithm can be modified to handle such implicitly given graphs with the same order of space complexity (whereas some big polynomial-time overhead may be necessary).

1. `PlanarReach`$(\widehat{G}, \widehat{s}, \widehat{t}, n)$
   (let $\widehat{n}$ be the number of vertices of $\widehat{G}$)
2. **If** $\widehat{n} \leq n^{1/2}$
3. **then** `BFS`$(\widehat{G}, \widehat{s}, \widehat{t})$
4. **Else** (let $\widehat{r} = \widehat{n}^{1-\epsilon}$)
5. Run `SepFamily` to compute $\widehat{r}$-separator family $\overline{S}$
6. Run `ImplicitBFS`$((\overline{S} \cup \{\widehat{s}, \widehat{t}\}, \overline{E}), \widehat{s}, \widehat{t})$
   // `ImplicitBFS` executes in the same way as `BFS`
   // except for the case "$(a, b) \in \overline{E}$?" is queried,
   // i.e., it is asked whether $G(V' \cup \overline{S})$ has an
   // edge (a, b)? In this case the query is answered
   // by the following process 6.1 ~ 6.5.
6.1. **For** every $x \in V$
   // V_x = the set of vertices of $\underline{G}(V \setminus \overline{S})$'s
   // connected component containing x.
6.2. **If** `PlanarReach`$(G(V_x \cup S), a, b, n)$ is True
6.3. **then** Return True for the query
6.4. **End_For**
6.5. Return False for the query

Fig. 1 Algorithm for the Directed Planar Reachability

This is the outline of our algorithm, and in Figure 1 we describe it in a pseudo code. For solving an instance (G, s, t) of the planar graph reachability problem, it suffices to call `PlanarReach`(G, s, t, n).

We analyze the space and time complexity of this algorithm; let $\mathcal{S}$ and $\mathcal{T}$ denote its space and time complexity functions. First note that, since $(1 - \epsilon)^k \leq 1/2$ for $k = O(1/\epsilon)$, the depth of recursion is $O(1/\epsilon)$, which is a constant.

We begin with the space complexity. Our goal is to show that $\mathcal{S}(n) = \widetilde{O}(n^{1/2+\epsilon})$. Consider $\mathcal{S}(\widehat{n})$ for any $\widehat{n} > n^{1/2}$. Line 5 uses space $\widetilde{O}(\widehat{n}^{1/2+\epsilon/2})$. For Line 6, we run `ImplicitBFS` on[*2]

[*2] Though we use $\overline{E}$ to denote symbolically the set of edges of $G(V' \cup \overline{S})$, this set is not used in the algorithm explicitly.

$(\overline{S} \cup \{\widehat{s},\widehat{t}\}, \overline{E})$, and its main computation can be done by using $\widetilde{O}(|\overline{S}|)$ space like the standard BFS. On the other hand, for each query $(a, b) \in \overline{E}$ asked in the computation, we need to run Lines 6.1 ∼ 6.5, and the space needed for this computation is essentially $\mathcal{S}(|V_x| + |S|) \leq \mathcal{S}(2\widehat{n}^{1-\epsilon})$ for Line 6.2. Hence we get the following recurrence.

$$\mathcal{S}(\widehat{n}) = \begin{cases} \widetilde{O}\left(\widehat{n}^{1/2+\epsilon/2}\right) + \mathcal{S}\left(2\widehat{n}^{1-\epsilon}\right) & \text{if } \widehat{n} > n^{1/2}, \\ \widetilde{O}\left(n^{1/2}\right) & \text{otherwise.} \end{cases}$$

Since the recursion depth is bounded by $O(1/\epsilon)$, it is easy to see that $\mathcal{S}(n) = O(1/\epsilon)\widetilde{O}(n^{1/2+\epsilon/2}) = \widetilde{O}(n^{1/2+\epsilon/2})$, which is sufficient for our goal.

Next consider the time complexity. We need to be precise only up to a polynomial factor. Then by analysis similar to the above, we have the following recurrence.

$$\mathcal{T}(\widehat{n}) = \begin{cases} q(n)\left(p_1(\widehat{n})\mathcal{T}\left(2\widehat{n}^{1-\epsilon}\right) + p_2(\widehat{n})\right) & \text{if } \widehat{n} > n^{1/2}, \\ q(n)\widetilde{O}\left(n^{1/2}\right) & \text{otherwise.} \end{cases}$$

Here $p_1(\widehat{n})$ is the number of times we make the recursive calls of Line 6.2, and $p_2(\widehat{n})$ is the time needed by SepFamily at Line 5. On the other hand, a polynomial $q(n)$ is for the overhead when $\widehat{G}$ is given implicitly. Again from the $O(1/\epsilon)$ bound for the recursion depth, it is easy to see the bound $\mathcal{T}(n) = p(n)^{O(1/\epsilon)}$ holds for some polynomial $p(n)$.

Finally we argue the correctness of the algorithm. Consider the execution of PlanarReach(G, s, t, n), and let H denote the graph $(\overline{S} \cup \{s, t\}, \overline{E})$ given at Line 6 in this execution, where $\overline{E}$ denotes the set of pairs (a, b) of $\overline{S} \cup \{s, t\}$ such that the process of Line 6.1 ∼ 6.5 returns true. We inductively assume that this process returns true if and only if there is a directed path from a to b in the induced graph $G(V_x \cup \overline{S})$ for some set of vertices V_x connected to x in $\underline{G}(V \setminus \overline{S})$. We claim that there is a directed path from s to t in G if and only if there is a directed path from s to t in H. The if-part is trivial because every directed edge of H corresponds to some directed path in G. Thus, we consider the only-if-part. Let p be a path from s to t in G. We can decompose p into $p_s p_1 p_2 \cdots p_l p_t$. Path p_s is the part of p that starts at s and enters $\overline{S}$ at the very first time. Let y_1 be the end vertex of p_s. In general p_i is the part of p that starts at $y_i \in \overline{S}$ and ends in y_{i+1} where y_{i+1} is the first vertex where the path p reenters $\overline{S}$ (after leaving $\overline{S}$). p_t is the part of the path that starts at y_{l+1} and ends in t. Notice that since $\overline{S}$ is a separator family, p_i completely lies inside the subgraph $G(V_x \cup \overline{S})$ for some x. Because of this observation, H has an edge (y_i, y_{i+1}) for every i, $1 \leq i \leq l$, and also edges (s, y_1) and (y_{l+1}, t). Hence $s y_1 y_2 \cdots y_{l+1} t$ is a path in H from s to t, as desired.

4. Space and time efficient separator algorithm

This section is devoted to the proof of the following theorem.

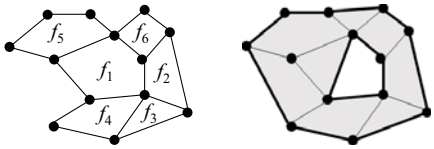
Theorem 2. *There is an algorithm that takes an undirected planar graph G with n vertices as input and outputs a (8/9)-separator of G. This algorithm runs in polynomial time and uses $\tilde{O}(\sqrt{n})$ space.*

We first make some assumptions that will make the presentation of the algorithm easier. We assume that a given input undirected graph for our separator algorithm is connected and triangulated. In fact, in our application of this algorithm, only a connected graph is given as an input to the algorithm. Also triangulation is easy by adding "imaginary edges" so that every face becomes a triangle. For example, we may use the following rule which can be implemented in $O(\log n)$ space (and hence polynomial time): For each face, consider the smallest indexed vertex on it. For every other vertex on that face, add an edge to this vertex, thereby triangulating the face. We may assume that this triangulation algorithm is applied before the execution of the separator algorithm. Also, as mentioned in the preliminaries, we assume without loss of generality the combinatorial planar embedding is given as a part of an input.

4.1 Preliminaries for the Separator Theorem

As mentioned in the introduction, our separator algorithm draws heavily from the work of Gazit and Miller [9]. Here we recall some key notions and notation which can be found in the literature [9], [12], [13], [14].

Let G be a planar graph (not necessarily triangulated) and let f be a *face*. The *face-size of* f is the number of edges (hence, that of vertices) of f. Two faces of G are *edge-connected* if they share an edge. A set of faces R is *edge-connected* if for every pair of faces f and g in R, there exist faces $f_1, \cdots, f_i$ in R such that f and f_1 are edge-connected, f_i and g are edge-connected, and f_j and f_{j+1} are edge-connected for all j, $1 \leq j \leq i-1$. A *region* of a planar graph is a set of edge-connected faces. The boundary of a region is the set of edges such that each edge lies on exactly one face of the region. Given a region R, we denote the boundary with $\mathcal{B}(R)$. It is known that the boundary of any region can be decomposed into a set of disjoint simple cycles [14].

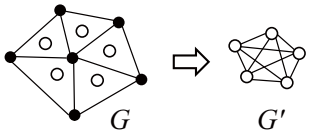


faces f_1, f_2, ... a shadow part: a region
two bold lines: a boundary

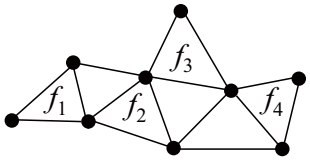
Fig. 2 Example of faces, regions, and boundaries.

From now on let G denote any planar graph that is connected and triangulated. Below we will usually regard triangular faces as vertices of a related graph called "face-vertex graph", which is different from the standard dual graph. The *face-vertex* graph of G is a graph denoted as $G' = (V', E')$ where V' is the set of triangle faces of G and E' is the set of pairs (f_1, f_2) of triangle faces of G such that f_1 and f_2 share a vertex in G. We add prefix "tr-" to distinguish terms for face-vertex graphs. For example, a vertex of G', which corresponds to some triangular face of G, is called a *tr-vertex*, and an edge of G' is called as a *tr-edge*. A path of G' consisting of tr-edges is called a *tr-path*; we often regard a *tr-path* as a sequence of faces.

The distance between two vertices v_1 and v_2 (denoted as $\mathrm{dist}(v_1, v_2)$) is the length (i.e., the number of edges) of the shortest path between them. Similarly, for any tr-vertices f_1 and f_2



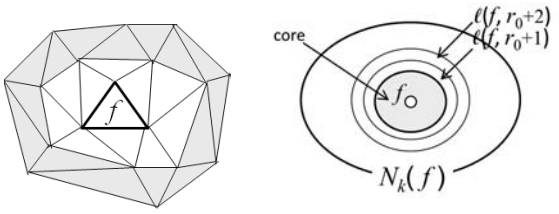
white circles: faces in G and tr-vertices in G'



tr-path (f_1, f_2, f_3, f_4):
a shortest tr-path from f_1 to f_4

Fig. 3 Example of face-vertex graph and tr-path.

in G', the distance between f_1 and f_2 (denoted by $\mathrm{dist}(f_1, f_2)$) is the length (i.e., the number of tr-edges) of the shortest tr-path between f_1 and f_2 in G'. Note that for any v_1 and v_2 of G that lie on triangle faces f_1 and f_2 respectively, we have $\mathrm{dist}(v_1, v_2) \leq \mathrm{dist}(f_1, f_2) + 1$. For a tr-vertex f and an integer r, let $\ell(f, r)$ denote the set of tr-vertices that are at distance exactly r from f. For any $d \geq 1$, the *d-radius ball* around f is the set $B_d(f) = \{g \in V' \mid \mathrm{dist}(f, g) \leq d\}$. Given a tr-vertex f and a region R, the distance between f and R is defined by $\mathrm{dist}(f, R) = \min_{g \in R}\{\mathrm{dist}(f, g)\}$.



$\ell(f, 2)$ $r_0 = \max_r\{|\ell(f, r)| \leq \sqrt{k}\}$

Fig. 4 Example of $\ell(f, r)$, $N_k(f)$, and core.

Let $k(n)$ be a function of n. The notion of $k(n)$-neighborhood defined below is central to the algorithm of Gazit and Miller [9].

Definition 3. For any tr-vertex f of G', the *$k(n)$-neighborhood of f* (denoted as $N_{k(n)}(f)$) is the set of $k(n)$ tr-vertices closest to f with respect to the distance function dist. More formally,

$$N_{k(n)}(f) = B_r(f) \cup F,$$

where r is the maximum integer such that $|B_r(f)| \leq k(n)$, and F is an edge-connected subset of $\ell(f, r+1)$ so that $|N_{k(n)}(f)|$ becomes exactly $k(n)$.

Remark. In the above definition there could be more than one choice for F. In such cases, we fix one such F (for example, by choosing it in a greedy way) and work with it. To avoid notational clutter, from now we will use k for $k(n)$, and write $N_k(f)$ for $N_{k(n)}(f)$.

Definition 4. A set I of faces (tr-vertices) is a *k-maximal independent set* if

- for every $f, g \in I$, $N_k(f) \cap N_k(g) = \emptyset$, and
- for every $f' \notin I$, there exists a face $f \in I$ such that $N_k(f') \cap N_k(f) \neq \emptyset$.

Note that the size of a k-maximal independent set is $O(n/k)$. We can compute a k-maximal independent set with time and space stated below by a straight forward greedy algorithm that considers faces in the lexicographic order. Thus, in our discussion, we may assume that some k-maximal independent set I is given (as a part of an input).

Lemma 4. *There is an algorithm that takes a planar graph G as an input, and outputs a k-maximal independent set in polynomial time and $\widetilde{O}(n/k + k)$ space.*

Next we define the notion of "core" of a face [12], [13].

Definition 5. Let f be a tr-vertex and $N_k(f)$ be its k-neighborhood. Let r_0 be the largest number such that $\ell(f, r_0) \subseteq N_k(f)$ and $|\ell(f, r_0)| \leq \sqrt{k}$. The core of f is defined as the union of $\ell(f, i)$, $1 \leq i \leq r_0$.

Note that a core is a region. The following lemma is critical. For its proof, see, for example, [11].

Lemma 5. *For every tr-vertex f, the following holds:*

- *The size of the boundary of the core of f is at most $\sqrt{k}$.*
- *For every tr-vertex $f' \in N_k(f)$ and not in core of f, there is a tr-vertex g in the core of f such that $\mathrm{dist}(f', g) \leq \sqrt{k}$.*

We extend the notion of core a face to core of a vertex. For any vertex v of G, let f be a triangle face (for consistency we take the lexicographically smallest face) on which v lies. The *core of v* is simply the core of f.

Next, we define the notion of "Voronoi region" [9]. We fix some k-maximal independent set I. For every face g of G, we associate a unique member of I as follows: If $g \in N_k(f)$ for some $f \in I$, then g is associated to f. For $g \notin N_k(f)$ for any $f \in I$, g is associated with the lexicographically first $f \in I$ such that $\mathrm{dist}(N_k(f), g)$ is the smallest among all faces in I. The *Voronoi region* of $f \in I$, denoted as $V(f)$, is the set of faces that are associated with f. (The name "region" is justified by the following lemma.)

Lemma 6. *[9], [12] Every Voronoi region is edge-connected. The diameter of a Voronoi region is O(k); that is, the distance between every pair of tr-vertices in the Voronoi region is O(k).*

We first show an algorithm that identifies, for a given face, the Voronoi region to which it belongs.

Lemma 7. *There is an algorithm that takes a planar graph G, a tr-vertex g of G′, and a k-maximal independent set I, as an input, and computes $f \in I$ such that $g \in V(f)$. Moreover, this algorithm runs in polynomial time and $\widetilde{O}(k)$ space.*

Proof. For each $f \in I$, compute distance from g to $N_k(f)$ if $N_k(f)$ and $N_k(g)$ intersect. Keep track of the smallest tr-vertex f for which this distance is minimized. Distance can be computed by storing both $N_k(f)$ and $N_k(g)$. Since each k-neighborhood has k tr-vertices, the distance computation can be done in $\widetilde{O}(k)$ space. Thus the space taken by this algorithm is $\widetilde{O}(k)$, and the running time of the algorithm is polynomial. □

Based on this, we next show that there is an algorithm to construct a BFS tree of a given Voronoi region in polynomial time using small space. This algorithm will be a component of our algorithm for computing a separator. Here we regard the Voronoi region as a subgraph of the face-vertex graph G', and by "BFS tree" we mean a tr-tree visiting all tr-vertices in a breadth first manner w.r.t. the tr-distance from the root.

Lemma 8. *There is a polynomial-time and $\widetilde{O}(k)$-space algorithm that, given a planar graph G, a k-maximal independent set I, and $f \in I$, constructs a BFS tree of $V(f)$ rooted at f. The diameter of this tree is $O(k)$.*

Proof. In order to use the standard method for producing a rooted tree using small space, it suffices to specify a parent relation so that it is easy to determine the parent of a given tr-vertex $g \in V(f)$ in the BFS tree.

First note that we can construct a BFS tree of the k-neighborhood of a given tr-vertex in $O(k)$ space and polynomial time by traversing tr-vertices in the standard way. This is because the k-neighborhood is defined by the tr-distance and each k-neighborhood has only k tr-vertices,

For a given g, we construct first the BFS tree of $N_k(f)$. If $g \in N_k(f)$, then its parent is the same as the parent in the BFS tree of $N_k(f)$. Else, construct $N_k(g)$ and its BFS tree, which should have some common tr-vertices since $g \in V(f)$. Identify the tr-vertex $f' \in N_k(f) \cap N_k(g)$ such that dist(g, f') is the minimum and f' is lexicographically the first one (in case there are more than one tr-vertices with the minimum distance). Consider the tr-path P from g to f' in the BFS of $N_k(g)$. The parent of g is the tr-vertex h in P that is adjacent to g (and hence one step closer to f'). Since each k-neighborhood has only k tr-vertices, all the computation can be done is in $\widetilde{O}(k)$ space and polynomial time.

Finally, note that the depth of the BFS tree is at most $2k$; hence the diameter of the tree is $O(k)$. □

4.2 The Separator Algorithm

We first outline the algorithm. Let G be an input planar graph with n vertices. We set $k = \sqrt{n}$, and first compute a k-maximal independent set I and check whether there is some $f \in I$ such that $V(f)$ has more than $n/3$ vertices of G. If such a Voronoi region exists, then we simply use the algorithm of Lipton and Tarjan to get a $(2/3)$-separator of $V(f)$, which is also a $(8/9)$-separator of the original graph G. Since we can construct a BFS tree of $V(f)$ in small space (Lemma 8), we can implement the algorithm of Lipton and Tarjan on a Voronoi region using small space in polynomial time.

Nontrivial treatment is necessary for the case where all Voronoi regions are small. In this case, we construct a small *weighted planar subgraph*, H of G (and a planar embedding) with a rational weight assigned to each face where the weights sum to 1. We then compute a *weighted separator* of H. This weighted separator of H will also be a separator of G.

For any ρ, $0 < \rho < 1$, a subset S of vertices of H is called a *ρ-weight-separator* if removal of S disconnects H into two components so that each component's total weight is at most ρ. We use the following theorem due to Miller to construct the weighted separator.

Theorem 9 ([14])**.** *There is a polynomial-time and $\widetilde{O}(m)$-space algorithm that takes as an input a weighted planar graph H satisfying the following three conditions and outputs a $(2/3)$-weight-separator of size $O(d\sqrt{m})$: (i) H has m faces, (ii) the maximum face-size is d, and (iii) there is no face with a weight more than $2/3$.*

The rest of the section makes the above outline formal. Recall that the boundary of each Voronoi region is a collection of simple cycles [14]. For the ease of presentation and for explaining the main idea behind our algorithm, we consider in this extended abstract only the case where the boundary of every Voronoi region is one simple cycle. When the boundary of some Voronoi region is not one simple cycle, borrowing ideas from [9] and [12], we can reduce to the case where the boundary of every Voronoi region is one simple cycle. The details will be given in the full version.

We need one more notion. For any Voronoi region, consider its boundary and vertices (of G) on the boundary. All such vertices belong to at least two Voronoi regions. On the other hand, there may be some vertex that belongs to three or more triangle faces each of which belongs to a different Voroni region; we call such vertices *Voronoi vertices* (Figure 5). The other vertices (on the boundary) are called *non Voronoi vertices*.

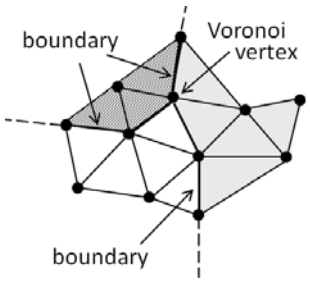


Fig. 5 Voronoi vertex

We now establish our main lemma which will be used in the proof of Theorem 2.

Lemma 10. *There is a polynomial-time, $\widetilde{O}(n/k + k)$-space algorithm, that takes a planar graph $G = (V, E)$ (with n vertices) as input and outputs either*

(1) a Voronoi region $V(f)$ such that the number of vertices in $V(f)$ is at least $n/3$.
or
(2) a weighted planar subgraph H of G with the following properties:
  (a) Every weight in H is less than $2/3$.
  (b) The number of faces in H is $O(n/k)$.
  (c) The size of each face of H is $O(\sqrt{k})$.
  (d) Any weight-separator of H is a separator of G.

Proof. The algorithm first computes a k-maximal independent set I using Lemma 4, and stores this set in the memory. This takes $\widetilde{O}(n/k + k)$ space and polynomial-time. For each $f \in I$, count the number of vertices in $V(f)$ as follows: Initialize a counter to zero. Consider a vertex $v \in V$. By cycling through all faces on which on which v lies, using Lemma 7, check if some face belongs to $V(f)$. If some face belongs to $V(f)$, then increment the counter. If the counter reaches $n/3$, then return $V(f)$. Checking whether a face belongs to $V(f)$ or not can be done in $\widetilde{O}(k)$ space and polynomial-time due to Lemma 7. Thus the total time taken to output $V(f)$ is polynomial and the space is $\widetilde{O}(n/k + k)$.

Assume that for every $f \in I$, the number of vertices in $V(f) < n/3$. Now we define our weighted planar graph H. We first describe the graph part, and later describe the weights. It is essentially a subgraph of $G = (V, E)$ consisting of a subset of edges of E (and their adjacent vertices). For a given G and its k-maximal independent set I, our algorithm executes the following three sub steps.

(1) For each $f \in I$, output the boundary of the core of f.
(2) For every $v \in V$, determine if it is a Voronoi vertex. If v is a Voronoi vertex, then output the boundary of the core of v.
(3) For every pair f and g of tr-vertices in I such that $\mathcal{B}(V(f))$ and $\mathcal{B}(V(g))$ intersect, do the following: Compute all Voronoi vertices that are common to $\mathcal{B}(V(f))$ and $\mathcal{B}(V(g))$. For every such vertex v, compute[*3] a path $\widehat{P}_{f,v}$ from f to v based on the tr-path in the BFS tree of $V(f)$. Then select the part of $\widehat{P}_{f,v}$ that lies outside of the cores of f and v, and output it as $P_{f,v}$. Similarly, output $P_{g,v}$.

Next we specify the way to assign weights to the faces of H. For each face h of H, assign a weight n_h/n, where n_h is the number of vertices of G that lie inside of h. We can define some simple rule so that vertices of H (that are also vertices of G) are counted once at some face.

Now our task is to show that the graph H satisfies the desired properties. First it is clear that H is planar and the sum of all weights is 1. Also from the above way of assigning weights it is clear that any ρ-weight-separator of H is an ρ-separator of G.

Next we examine the faces of H and their size parameters. Note that a face is defined by edges and that every edge of H is either a part of the boundary of some core or a part of path $P_{f,v}$ for some $f \in I$ and some Voronoi vertex v. We can classify all faces to the following two types (see Figure 6).

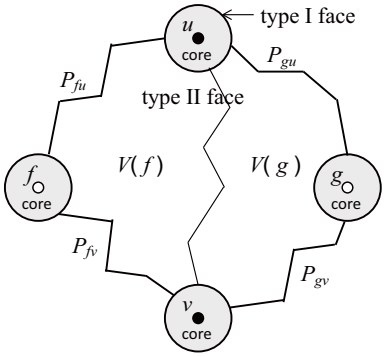


Fig. 6 Type I and Type II faces of H

Type I: A face consisting of edges from the boundary of some core that are produced by sub steps (1) or (2).

Type II: A face consisting of the edges of four paths $P_{f,u}, P_{f,v}, P_{g,u}, P_{g,v}$ and some edges in boundaries of the cores of f, g, u, and v, where f and g are tr-vertices of G such that $\mathcal{B}(V(f))$ and $\mathcal{B}(V(g))$ intersect, and u and v are Voronoi vertices that appear in both $V(f)$ and $V(g)$. Note that the edges belonging to these paths are produced by sub step (3) and the edges from the cores are produced by sub steps (1) or (2).

[*3] Precisely speaking, obtain first the tr-path from f to a triangle face in $V(f)$ containing v; then define $\widehat{P}_{f,v}$ by selecting a set of edges from this tr-path in any appropriate way.

Claim 1. *All faces of H are either type I or type II.*

Proof. It follows from the fact that there is no face of H that contains vertices of G from more than two Voronoi regions. This fact also implies that there are only two Voronoi vertices in any Type II face. □

Claim 2. *The face-size of all faces of H is* $O(\sqrt{k})$.

Proof. The $O(\sqrt{k})$ bound for the face-size of Type I faces is immediate from the part (1) of Lemma 5. On the other hand, for any Type II face and any path $P_{f,v}$ of this face, we can bound its length by $O(\sqrt{k})$ from the part (2) of Lemma 5. Then the claim follows since there are at most eight segments of length $O(\sqrt{k})$ paths. □

Claim 3. *The number of faces of H is* $O(n/k)$

Proof. We first bound the number of Voronoi vertices. For this, consider a graph U consisting of Voronoi vertices. For any pair u and v of Voronoi vertices, $\langle u, v\rangle$ is an edge of U if and only if there is a path in G from u to v along the boundary of some Voronoi region, and this path does not contain any other Voronoi vertex. Let n_U and e_U be the number of vertices and edges of U respectively, and we bound these numbers below. Note that there is one to one correspondence between faces of U and Voronoi regions of G. Hence, the number of faces of U is $|I| = O(n/k)$. On the other hand, each Voronoi vertex has at least three neighbors in U; thus, we have $e_U \geq 3n_U/2$. Then by using Euler's formula, we can also bound both n_U and e_U by $O(n/k)$. Recall n_U is the number of Voronoi vertices.

Now we bound the number of faces of H. The number of Type I faces is $|I| + n_U$, and hence it is $O(n/k)$. Note that each Type II face corresponds to a part of the boundary of two Voronoi regions connecting two Voronoi vertices, which in fact corresponds to an edge of U. Thus, the number of Type II faces is bounded by $e_U = O(n/k)$. The bound of the claim follows from these bounds. □

We also need to show that no face has a weight larger than $2/3$. This is proved as follows by using the assumption that no Voronoi region has more than $n/3$ vertices.

Claim 4. *If there is no* $f \in I$ *such that* $V(f)$ *contains more than* $n/3$ *vertices of G, then H has no face with weight* $> 2/3$.

Proof. Clearly the weight of Type I face is less than $2/3$ (if n is sufficiently large). On the other hand, any Type II face (in G) is a subset of at most two Voronoi regions; thus, the weight bound $2/3$ is immediate from the assumption. □

Finally, we show that the following bound on the efficiency of the algorithm for computing H.

Claim 5. *The above procedure for computing H can be implemented in polynomial time and* $\widetilde{O}(n/k + k)$ *space.*

Proof. By computing $\ell(f, r)$ iteratively, we can compute a core in polynomial time and $\tilde{O}(k)$ space. Hence, sub step (1) can be

done within required time and space. The computation for sub step (2) is also easy because for a given vertex v and a given k-independent set I, we can check whether v is a Voronoi vertex or not by checking whether three faces adjacent to v belong different Voronoi regions. This can be done in polynomial-time and $\widetilde{O}(k)$-space. For sub step (3), we first need to check whether $v \in V(f)$ for a given v, which can be done in polynomial time and $\widetilde{O}(k)$ space (Lemma 7). Then $\widetilde{P}_{fv}$ can be obtained following the BFS tree that is computable in polynomial time and $\widetilde{O}(k)$ space (Lemma 8).

For computing weights, we need to count the number of vertices of G in a given face h of H. For this, it is enough to show a way to determine, for a given vertex v of G, whether it is in h or not. We can test this by considering a clockwise orientation of the edges of h and then checking whether v is connected to (any fixed vertex u of) h from the left ($\Leftrightarrow$ out) or the right ($\Leftrightarrow$ in) of the orientation. This last point can be tested by the reachability to u from v without using edges adjacent to h from the right to the orientation. The reachability test can be done in $O(\log n)$ space and hence in polynomial time. □

This concludes the proof of Lemma 10 □

Now we are ready to present the proof of Theorem 2.

Proof of Theorem 2: We explain the execution of our separator algorithm `Separator` on a given planar graph G of n vertices. As discussed above, we may assume that G is connected and triangulated and that some combinatorial embedding is also given as an input.

The algorithm sets $k = \sqrt{n}$ and runs the algorithm from Lemma 10. If the algorithm outputs a big Voronoi region $V(f)$ with more than $n/3$ vertices, then (implicitly) construct a BFS tree of $V(f)$ using the algorithm from Lemma 8. The BFS tree has diameter $O(\sqrt{n})$. Let $n' \geq n/3$ denote the number of vertices of G in $V(f)$. We now apply the algorithm of Lipton and Tarjan on the subgraph induced by $V(f)$. Given a BFS tree (with diameter d) of the input graph, the Lipton and Tarjan's algorithm can be implemented in linear time and logarithmic space to compute a separator of size $O(d)$. Since the BFS of $V(f)$ has diameter $O(\sqrt{n})$, and can be computed in polynomial-time using $\tilde{O}(\sqrt{n})$-space, we can compute a $(2/3)$-separator of $V(f)$ using space $O(\sqrt{n})$ and polynomial-time. Certainly, this separator separates some subgraph of G of size $\geq n'/3$ ($\geq n/9$) from not only $V(f)$ but also from G; thus, it is also an $(8/9)$-separator, satisfying the theorem.

If no big Voronoi region exists, the algorithm produces subgraph H of G. Now apply Miller's algorithm from Theorem 9 to compute a $(2/3)$-weight-separator of H. From Lemma 10 we know that H has $O(n/k)$ ($= O(\sqrt{n})$) faces with their face-size bounded by $O(\sqrt{k})$ ($= O(n^{1/4})$). Thus, Miller's algorithm runs in polynomial time and $\widetilde{O}(\sqrt{n})$ space, and the size of the separator is $O(\sqrt{n})$. On the other hand, any $2/3$-weighted separator for H is also a $2/3$-separator for G. Therefore, both the output and the efficiency of the algorithm satisfy the required conditions.

References

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

[2] Tetsuo Asano and Benjamin Doerr. Memory-constrained algorithms for shortest path problem.z In *Proceedings of the Twenty Third Canadian Conference on Computational Geometry* (CCCG'11), 2011.

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

[4] Eric Allender and Meena Mahajan. The complexity of planarity testing. *Information and Computation*, 189:117–134, 2004.

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

[6] Stephen A. Cook and Charles Rackoff. Space lower bounds for maze threadability on restricted machines. *SIAM Journal of Computing*, 9(3):636–652, 1980.

[7] Jeff Edmonds, Chung Keung Poon, and Dimitris Achlioptas. Tight lower bounds for st-connectivity on the NNJAG model. *SIAM Journal of Computing*, 28(6):2257–2284, 1999.

[8] Greg N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM Journal of Computing*, 16(6):1004–1022, 1987.

[9] Hillel Gazit and Gary L. Miller. A parallel algorithm for finding a separator in planar graphs. In *Proceedings of the 28th Annual Symposium on Foundations of Computer Science* (FOCS'87), pages 238–248, 1987.

[10] Monika Rauch Henzinger, Philip N. Klein, Satish Rao, and Sairam Subramanian. Faster shortest-path algorithms for planar graphs. *Journal of Computer System and Science*, 55(1):3–23, 1997.

[11] Philip Klein. On Gazit and Miller's parallel algorithm for planar separators: achieving greater efficiency through random sampling. In *Proceedings of the Fifth Annual ACM Symposium on Parallel Algorithms and Architectures* (SPAA '93), pages 43–49, 1993.

[12] Ioannis Koutis and Gary L. Miller. A linear work, $O(n^{1/6})$ time, parallel algorithm for solving planar laplacians. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms* (SODA'07), pages 1002–1011, 2007.

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

[14] Gary L. Miller. Finding small simple cycle separators for 2-connected planar graphs. *Journal of Computer System and Science*, 32(3):265–279, 1986.

[15] Chung Keung Poon. Space bounds for graph connectivity problems on node-named jags and node-ordered jags. In *Proceedings of the 34th Annual Symposium on Foundations of Computer Science* (FOCS'93), pages 218–227, 1993.

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

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

[18] Derrick Stolee and N. V. Vinodchandran. Space-efficient algorithms for reachability in surface-embedded graphs. In *Proceedings of the 27th IEEE Conference on Computational Complexity* (CCC'12), pages 326–333, 2012.

[19] Avi Wigderson. The complexity of graph connectivity. In *Proceedings of the 17th International Symposium on Mathematical Foundations of Computer Science* (MFCS'92), pages 112–132, 1992.