Electronic Colloquium on Computational Complexity, Report No. 39 (2019)

Planarity, Exclusivity, and Unambiguity

Eric Allender[*] Archit Chauhan[†] Samir Datta[‡]
Anish Mukherjee[§]

March 12, 2019

Abstract

We provide new upper bounds on the complexity of the s-t-connectivity problem in planar graphs, thereby providing additional evidence that this problem is not complete for NL. This also yields a new upper bound on the complexity of computing edit distance. Building on these techniques, we provide new upper bounds on the complexity of several other computational problems on planar graphs. All of these problems are shown to be solvable in logarithmic time on a concurrent-read exclusive-write (CREW) PRAM. The new upper bounds are provided by making use of a known characterization of CREW algorithms in terms of "unambiguous" AC^1 circuits. This seems to be the first occasion where this characterization has been used in order to provide new upper bounds on natural problems.

[*]Department of Computer Science, Rutgers University, Piscataway, NJ, USA, allender@cs.rutgers.edu. Supported by NSF grant CCF-1514164. This work was done in part while the author was visiting the Simons Institute for the Theory of Computing.

[†]CMI, Chennai, India, archit@cmi.ac.in. Partially supported by a grant from Infosys foundation.

[‡]CMI, Chennai, India, sdatta@cmi.ac.in. Partially supported by a grant from Infosys foundation and SERB grant MTR/2017/000480.

[§]CMI, Chennai, India, anish343@gmail.com. Partially supported by a grant from Infosys foundation and TCS PhD fellowship.

ISSN 1433-8092

1 Introduction

Is it easier to find a path in a planar graph than in a general directed graph?

Finding a path from s to t in a graph is the standard complete problem for NL, and it is known that if one restricts this problem to planar graphs one obtains a problem in UL [BTV09]. But since NL = UL under a plausible derandomization hypothesis [ARZ99, KvM02], the UL upper bound for planar reachability does not shed much light on the question of whether planar reachability might also be NL-complete.

In this paper, we present a new upper bound on the complexity of planar reachability; the problem can be solved in logarithmic time on a CREW-PRAM. By the work of [Lan93], it is known that log-time CREW PRAMs accept precisely the sets that lie in a circuit complexity class called UAC^1. As its name suggests, UAC^1 refers to logarithmic-depth circuits composed primarily of unbounded fan-in "unambiguous" AND and OR gates. However, the "unambiguity" represented by these hardware components has less to do with "unambiguity" as represented by UL, and is more closely related to "strong unambiguity" as represented by StUL.[1] The connection is perhaps best illustrated by considering semi-unbounded circuits. The class SAC^1 (log-depth circuits with unbounded-fan-in OR gates and bounded fan-in AND gates) coincides with the class of problems logspace-reducible to context-free languages (and hence it is also known as LogCFL [Sud78]) and is also equal to the class of problems accepted by logspace-bounded nondeterministic auxiliary pushdown automata (APDAs) in polynomial time. Similar to the NL vs UL situation, under a plausible derandomization hypothesis, no power is lost if the APDAs are unambiguous [RA00, ARZ99] (meaning that, on every input, there is at most one accepting computation path). But the semi-unbounded version of UAC^1, known as USAC^1, corresponds to APDAs that satisfy the more restrictive "strong unambiguity" condition (meaning that, between *any two configurations* there is never more than one path). Unambiguous CFLs are not known to lie in USAC^1, and there is no indication that NL (or UL) should be contained in UAC^1 (to say nothing of USAC^1). Although there is a large literature on CREW-PRAM algorithms and on planar reachability, it has been unknown until now that planar reachability has a logarithmic-time CREW algorithm. We believe that this provides stronger evidence that planar reachability is not complete for NL. It is worthwhile noting that a novel aspect of our approach is that we capitalize on the characterization of CREW algorithms in terms of unambiguous circuits; we are aware of no prior instance where unambiguous circuits were used as a tool to devise CREW algorithms.

As an interesting by-product, we give a new upper bound on the edit distance problem, which has been the subject of several important recent investigations, mostly centering on the question of whether the there is a sequential algorithm with a running time significantly better than quadratic (and showing that the

[1]For those readers who are unfamiliar with StUL, additional background and motivation can be found in Section 2.

existence of such an algorithm would have dramatic complexity-theoretic consequences); see [BI18, AHWW16] and also see the discussion in [AB17, CDG+18]. It has been known since the work of [Mat88, AALM90] that the edit distance problem reduces to the problem of computing the distance between two vertices in a weighted planar digraph. Our reachability algorithm for planar graphs also provides an algorithm for computing distance; thus we establish for the first time that the edit distance problem lies in $\mathsf{CREW}[\log n]$.

We also extend our techniques, to present logarithmic-time CREW algorithms for several other problems that previously had been known only to lie in AC^1.

The rest of the paper is organized as follows: In Section 2 we provide background on the complexity classes and models of computation that we consider. In Section 3 we present our main upper bounds on the complexity of reachability in planar graphs and related problems. In Section 4 we present implications that our reachability algorithm has, for the edit distance problem. In Section 5 we show that the problem of computing a Depth-First Search tree in a planar graph reduces to planar distance, thereby obtaining a new upper bound on the complexity of this fundamental problem. In Section 6 we present a number of other applications of our techniques. Finally, we conclude in Section 7 with some open ends.

2 Preliminaries: Unambiguity, Strong Unambiguity and CREW PRAMs

We assume that the reader is familiar with the standard complexity classes L, NL and P (see e.g. the text [AB09]) and with the logspace-uniform circuit complexity classes NC^k, AC^k, and SAC^k (see, e.g., the text [Vol99]).

A nondeterministic Turing machine is said to be *unambiguous* if, on every input x, there is at most one accepting computation path. If an unambiguous machine satisfies the additional condition that on every input x and *for every two configurations* C and D of the machine, there is at most one computation path that leads from C to D, then it is said to be *strongly unambiguous.* If we consider logspace-bounded nondeterministic Turing machines, then unambiguous machines yield the class UL, whereas strongly unambiguous machines yield the class StUL. There is some evidence that StUL is a weaker class than UL. $\mathsf{StUL} \subseteq \mathsf{DSPACE}(\log^2 n/\log\log n)$ [AL98], whereas it is widely conjectured that $\mathsf{NL} = \mathsf{UL}$. (If $\mathsf{UL} \neq \mathsf{NL}$, then every set in $\mathsf{DSPACE}(n)$ has circuits of size $2^{o(n)}$ for infinitely many input lengths n [RA00, ARZ99, KvM02].)

The notions of unambiguity and strong unambiguity have also been studied in the context of auxiliary pushdown automata. A logarithmic-space bounded *auxiliary pushdown automaton (APDA)* is a Turing machine with a read-only input tape, a read-write worktape of length $\log n$ on inputs of length n, and a pushdown store. When there is no time bound, logspace-bounded APDAs (both deterministic and nondeterministic) characterize P [Coo71]. More relevant for this paper are the classes corresponding to polynomial-time bounded logspace APDAs. For deterministic machines, $\mathsf{DAPDA}(\log,\mathsf{poly}) = \mathsf{LogDCFL}$ is exactly the class of languages accepted in logarithmic time by PRAMs that satisfy the Concurrent Read Owner Write restriction [Sud78, DR00], as well as being the class of languages logspace-reducible to deterministic context-free languages. For nondeterministic machines $\mathsf{NAPDA}(\log,\mathsf{poly}) = \mathsf{LogCFL}$ is exactly the class SAC^1 [Sud78, Ven91]. Under the same plausible derandomization hypothesis that implies $\mathsf{NL} = \mathsf{UL}$, it follows that $\mathsf{SAC}^1 = \mathsf{LogCFL} = \mathsf{NAPDA}(\log,\mathsf{poly}) = \mathsf{UAPDA}(\log,\mathsf{poly})$ [RA00, ARZ99, KvM02].

In addition to the Turing-machine models discussed above, two other models figure prominently in the study of NC and its subclasses: circuits and PRAMs. We are assuming familiarity with the circuit classes AC^0, NC^1, SAC^1 and AC^1. PRAMs come in various different flavors, depending on the types of access to shared memory that are allowed, such as concurrent-read concurrent-write (CRCW), concurrent-read exclusive write (CREW), and concurrent-read owner-write (CROW). (There are other flavors also, but they are not relevant here.) We use the notation $\mathsf{CRCW}[\log n]$, $\mathsf{CREW}[\log n]$, and $\mathsf{CROW}[\log n]$ to denote the classes of languages accepted by PRAMs running in time $O(\log n)$ on inputs of length n, using a polynomial number of processors, with the given type of access to shared memory. $\mathsf{CROW}[\log n]= \mathsf{LogDCFL}$[DR00], and $\mathsf{CRCW}[\log n]= \mathsf{AC}^1$ [SV84]. The remaining class $\mathsf{CREW}[\log n]$ is central to our investigation.

Lange [Lan93] investigated $\mathsf{CREW}[\log n]$ as it relates to a circuit-based notion of unambiguity. As studied in [Lan93], circuits for a class such as AC^1 consist of both *bounded* fan-in AND and OR gates (with fan-in two) and NOT gates (with fan-in one), and *unbounded* fan-in AND and OR gates (which are denoted as $\forall$ and $\exists$ gates, respectively). A circuit C is said to be *unambiguous* if, for every input x, each $\forall$ gate has at most one predecessor set to 0, and each $\exists$ gate has at most one predecessor set to 1. UAC^1 is the class of languages accepted by logspace-uniform circuit families of unambiguous circuits of depth $O(\log n)$ and polynomial size. USAC^1 is the class of languages accepted by logspace-uniform circuit families of unambiguous circuits of depth $O(\log n)$ and polynomial size, which have no $\forall$ gates, and where NOT gates are allowed only at the input level.

Theorem 1. *[Lan93]*

- $\mathsf{UAC}^1 = \mathsf{CREW}[\log n]$.
- $\mathsf{USAC}^1 = \mathsf{StUAPDA}(\log,\mathsf{poly})$.

3 Distance in Planar Graphs

The complexity of computing reachability and distance in planar graphs has a long history. Since the problems lie in $\mathsf{NL} \subseteq \mathsf{AC}^1$, the existence of a $\mathsf{CRCW}[\log n]$ algorithm is obvious. Attention from the algorithmic community has focused on processor-efficient algorithms, even if this pushed the running time above $O(\log n)$, as in [AALM90, STV95] or on polynomial-time small-space algorithms, as in [AKNW14, JT19]. From the standpoint of complexity theory, the main results are that planar reachability reduces to non-reachability [HRS93, ABC$^+$09] and that reachability lies in UL (and hence in $\mathsf{UL} \cap \mathsf{co\text{-}UL}$) [BTV09].

In this section, we show that planar distance lies in UAC^1, and hence by Theorem 1 lies in $\mathsf{CREW}[\log n]$. We use deterministic isolation of shortest paths in planar graphs [BTV09] as one of the main tools.[2]

Theorem 2. *The reachability problem for planar graphs is in* $\mathsf{CREW}[\log n]$.

Proof. Let G be a planar digraph with n vertices. We can compute a planar embedding for G in logspace [AM04, Rei08]. Let $w_{\text{uniq}} : E(G) \to \mathbb{N}$ be a logspace-computable weight function such that the minimum weight path between any two vertices is unique, where $w_{\text{uniq}}(e) \leq n^{O(1)}$ [BTV09, TV12]. We construct a graph $\tilde{G}$ from G by subdividing edge $e \in E(G)$ into $w_{\text{uniq}}(e)$ many edges. For the rest of the section we work with the digraph $\tilde{G}$. Let $d_{\leq}(u, v, i), d_{>}(u, v, i)$ be predicates encoding whether the distance from u to v is $\leq i$ or $> i$, respectively. For every pair (u, v) there is at most one shortest path from u to v.

Table 1 gives an inductive definition of $d_{\leq}$ and $d_{>}$.

$[(1) - (3)]$ are base cases to define the two basic distance predicates $d_{\leq}, d_{>}$ on lengths at most 1. Both (4),(5) are guarded adaptations of Savitch's algorithm. In (4) the existential quantifier evaluates to true iff the correct distance j between u, v and the correct mid point w is picked assuming that the predicates hold inductively. Since the (j, w) pair is unique, unambiguity is maintained. The argument for (5) is analogous (and dual).

The inductive definition above immediately translates into a circuit whose gates are labeled $d_{\leq}(u, v, i)$and $d_{>}(u, v, i)$, using unambiguous $\forall$ and $\exists$ gates, thereby establishing membership in $\mathsf{UAC}^1 = \mathsf{CREW}[\log n]$. Determining if there is a path from s to t in the graph involves simply evaluating $d_{\leq}(s, t, n)$. □

Although the preceding algorithm involves computing distance in a weighted graph, it does not immediately yield an algorithm for computing distance in the original graph. This is addressed in the following corollary, where we introduce the notation PlDist to refer to the problem of computing distance in a weighted planar graph.

Corollary 3. PlDist *is in* $\mathsf{CREW}[\log n]$.

[2]The reader may be wondering whether one could build on our approach, to show that NL (or UL) is contained in $\mathsf{CREW}[\log n]$, under the assumption that $\mathsf{DSPACE}(n)$ contains a problem that requires exponentially-large circuits. After all, our $\mathsf{CREW}[\log n]$ algorithm for planar reachability is based on logspace-computable weight functions that isolate minimum-weight paths [BTV09, TV12], and similar weight functions also imply $\mathsf{NL} = \mathsf{UL}$ [RA00, ARZ99]. But there is an important difference. The argument in [TV12] shows that there is a logspace-computable function that takes a planar graph G as input and produces *one* weight function as output that is good for G, and the argument in [BTV09] produces a single weight function that is good for *every* n-by-n grid graph. In contrast, the arguments in [ARZ99, KvM02] serve only to produce a *list* of weight functions, one of which is guaranteed to be good for a given graph G. It is not at all clear that one can construct *unambiguous* AC^1 circuits that incorporate such unreliable weight functions.

1. $\forall u\ d_{\leq}(u,u,0)$ and $\neg d_{>}(u,u,0)$
2. $\forall u,v$ if $(u \neq v),\ \neg d_{\leq}(u,v,0)$ and $d_{>}(u,v,0)$
3. $\forall (u,v) \in E,\ d_{\leq}(u,v,1)$ and $\neg d_{>}(u,v,1)$
4. $\forall i > 0 : \forall u,v\ d_{\leq}(u,v,i) \iff$
$$\exists w \exists j \leq i, (d_{\leq}(u,w,\lfloor j/2 \rfloor) \wedge d_{\leq}(w,v,\lceil j/2 \rceil) \wedge (d_{>}(u,w,\lfloor j/2 \rfloor - 1) \wedge d_{>}(w,v,\lceil j/2 \rceil - 1)))$$
5. $\forall i > 0 : \forall u,v\ d_{>}(u,v,i) \iff$
$$\forall w \forall j \leq i, (d_{>}(u,w,\lfloor j/2 \rfloor) \vee d_{>}(w,v,\lceil j/2 \rceil) \vee (d_{\leq}(u,w,\lfloor j/2 \rfloor - 1) \vee d_{\leq}(w,v,\lceil j/2 \rceil - 1))))$$

Table 1: The predicates $d_{\leq}, d_{>}$

Proof. Thierauf and Wagner [TW10b, Section 4] show that the techniques of [BTV09, RA00, ABC^{+}09] can be combined to show that distance in planar graphs can be computed in $\mathsf{UL} \cap \mathsf{co\text{-}UL}$, by reducing the computation of distance to the planar reachability problem. An identical argument applies, to yield a $\mathsf{CREW}[\log n]$ algorithm.

More precisely, Thierauf and Wagner observe that, given a planar graph G, the argument in [ABC^{+}09] shows how to produce a grid graph G' with certain edges labeled as "distinguished", with the property that every path p between two vertices in G can be associated with a unique path p' in G', where furthermore the length of the path p is equal to the number of "distinguished" edges in p'. (Essentially, edges in G are mapped to paths in G', and some of the edges are G' are marked as corresponding to "real" edges in G.) They then show that a modification of the [BTV09] weight function has the property that, given the weight of a path in G', one can easily determine the number of "distinguished" edges in the path, and thereby determine the distance between two vertices in G. □

Corollary 4. *Breadth-first search trees for planar directed graphs can be computed in* $\mathsf{CREW}[\log n]$.

Proof. Given a graph G, select a vertex r to be its root. (If there is a directed breadth-first search tree that spans the entire graph, then r can be selected so that there is a path from r to every other vertex, using the $\mathsf{CREW}[\log n]$ algorithm for reachability.) For each vertex v, compute the distance from r to v. Partition the vertex set into blocks V_d consisting of those nodes at distance d from r. For each $v \in V_d$, select the lexicographically first neighbor x of v in V_{d-1}, and include an edge from x to v.

All of the final steps can easily be computed in logarithmic space, and thus can be accomplished in $\mathsf{CREW}[\log n]$. ☐

The following is immediate from Theorem 3 and [LMN10]:

Corollary 5. *There is a* $\mathsf{CREW}[\log n]$ *algorithm to find a longest path in a weighted planar directed acyclic graph.*

Proof. It is shown in [LMN10] (in particular) that the longest path problem for weighted planar DAGs L-reduces to the problem of finding a shortest path in weighted planar DAGs (where weights in both graphs are polynomially bounded). ☐

We remark that the results of this section also hold for graphs of bounded genus, since embeddings of such graphs can be computed in logspace [EK14] and since reachability in such graphs reduces to the planar case [KV10].

4 Minimum Edit Distance

The edit distance or the *Levenshtein distance* is a way of measuring distance between two strings a, b over some alphabet Σ by counting the minimum number of insertions, deletions, and substitutions required to convert a to b. More formally:

Let $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_\ell\}$ be an alphabet, and $a = a_1 \ldots a_m, b = b_1 \ldots b_n$ be two strings on the alphabet. The following are the set of allowed operations on string a,

- Deletion : Delete a letter a_i appearing in the string a.
- Insertion : Insert a letter b_i in the string a.
- Substitution: Substitute a letter a_i in the string a by a letter b_j.

Substituting a character by itself has zero cost. In Levenshtein's original definition, each of these operations has unit cost, so the distance is equal to the minimum number of operations required to transform a to b. A more general definition that we consider here associates non-negative weight functions D_{a_i}, I_{b_i} and S_{a_i,b_j} with the operations. In the following we assume that these weights and m are all bounded by $\text{poly}(n)$. It is known that the edit distance can be computed in AC^1 and in $\mathsf{CREW}[\log^2 n]$ [AALM90]. In this paper we show that it can be computed in $\mathsf{CREW}[\log n]$.

The problem was shown earlier to be reducible to finding distance in a (planar) grid graph (the dynamic programming graph) $G_{a,b}$ [Mat88, AALM90]; it is clear that the reduction presented in [Mat88, AALM90] is logspace-computable.

Proposition 6. *[Mat88, AALM90] The minimum edit distance between a and b is equal the minimum weight path from the vertex* $(0,0)$ *to* (m,n) *in* $G_{a,b}$.

Corollary 7. *The edit distance problem lies in* $\mathsf{CREW}[\log n]$.

Related Problems The length of the *Longest common subsequence* (LCS) of two given strings a, b is edit distance with insertion and deletion as the only two edit operations, both at unit cost. The *longest increasing subsequence problem* is closely related to the longest common subsequence problem: the longest increasing subsequence of a sequence S is the longest common subsequence of S and S^*, where S^* is obtained by sorting S. The previous best known bounds for these problems were $\mathsf{CREW}[\log^2 n]$ [LL94] and $\mathsf{CRCW}[\log n]$ [BS97].

5 Planar Depth-First Search

In this section we show the following,

Theorem 8. *There is a $\mathsf{CREW}[\log n]$ algorithm to compute a depth-first search tree of an undirected planar graph. (More precisely, this problem is logspace-Turing-reducible to* PLDIST.*)*

Our algorithm is a refinement of an AC^1 algorithm presented by Hagerup [Hag90]; we therefore introduce some terms and notation from [Hag90]:

5.1 Notations and Definitions

Definition 9. *Let* PLDFS *be the class of all functions that map an embedded undirected (connected) planar graph G and a vertex $r \in V(G)$ to a set of arcs (or directed edges) A that constitutes a depth-first search tree rooted at r.*

Let $G = (V, E)$ be an embedded undirected planar graph. Let $\mathcal{F}$ be the set of G's faces. Consider the face-incidence graph G_D. (This is similar to, but not identical to, the *dual graph*. Namely, the vertices of G_D are the faces of G, and two faces are adjacent in G_D if they share a vertex of G.)

Consider some initial face F_0. The *type* of a face F, denoted $Type(F)$, is its distance from F_0 in G_D. For $\alpha \in V \cup E$ define $Type(\alpha) = \{k \geq 0 | \alpha$ has an incident face F with $Type(F) = k\}$.

Note that G_D need not be planar. We will show in the algorithm how to compute distances of vertices in G_D using the face-vertex incidence graph which is planar.

For every $\alpha \in V \cup E, Type(\alpha)$ is an element of the sequence $\{0\}, \{0, 1\}, \{1\}, \{1, 2\}, \{2\}, \cdots$. This allows us to define a total order $<$ on set of vertex and edge types by

$$\{0\} < \{0, 1\} < \{1\} < \{1, 2\} < \{2\} < \{2, 3\} < \ldots \tag{1}$$

Call α *white* if $|Type(\alpha)| = 1$ and *black* if $|Type(\alpha)| = 2$.

Let $G_{\mathcal{B}}$ be the subgraph of G spanned by black edges. We start from a few lemmas extracted from [Hag90]:

Lemma 10. *$G_{\mathcal{B}}$ contains all black vertices of G. Also, all vertices and edges inside a connected component of $G_{\mathcal{B}}$ have the same type.*

Lemma 11. *Let C be a simple cycle. If F_0 lies in the interior (exterior) of C and the exterior (interior) of C contains a face of type k, then C must contain a vertex of type $< k$.*

Following [Hag90], we refer to a biconnected component of a graph as a *block* of the graph.

Lemma 12. *Every block of $G_{\mathcal{B}}$ is a simple cycle.*

From now on, we will assume that F_0 is the external face of a connected graph G. It follows from Lemma 12 that the black edges of G constitute a set of cycles. Given any two black cycles in G, they are either disjoint, or they share a single vertex.

Next we diverge from [Hag90] and introduce some new machinery and terminology:

Definition 13. *Let V_k denote the (white) vertices of type $\{k\}$ and let $V_{k,k+1}$ denote the (black) vertices of type $\{k, k+1\}$. We call the set $V_k \cup V_{k,k+1}$ the $k+1^{th}$* layer*, denoted by $\mathcal{L}^{k+1}$. Observe that every vertex of G lies in some $\mathcal{L}^{k+1}$. We call the connected components of a layer* layered connected components *or* LCCs. *$\mathcal{LCC}^{k+1}$ denotes the set of connected components of the $k+1^{th}$ layer; we will use the notation L^{k+1} to refer to an element of $\mathcal{LCC}^{k+1}$.*

Observe that $\mathcal{L}^1$ consists of all edges that are adjacent to the external face F_0, along with all vertices that are adjacent to F_0. The black edges of $\mathcal{L}^1$ form a set of cycles, and every other black edge of G lies in the interior of some black cycle of $\mathcal{L}^1$. There are no cycles consisting of white edges in $\mathcal{L}^1$ (because any such cycle would necessarily enclose a face, meaning that the edge would not be white).

The vertices in $\mathcal{L}^2$ are adjacent to the external face of the graph $G - \mathcal{L}^1$ (and in general the vertices in $\mathcal{L}^k$ are adjacent to the external face of $G - \bigcup_{i<k} \mathcal{L}^i$). Thus, in particular, each $L^k \in \mathcal{LCC}^k$ is outerplanar. Also, each $L^{k+1} \in \mathcal{LCC}^{k+1}$ lies in the interior of a black cycle in $\mathcal{L}^k$, and there are no cycles consisting entirely of white edges in $\mathcal{L}^k$ for any k.

Since each $L^k \in \mathcal{LCC}^k$ is outerplanar, given any starting vertex v in L^k, it is easy to construct a depth-first search tree $T_v(L^k)$:

- The root of $T_v(L^k)$ is v.
- For a white vertex x of $T_v(L^k)$, all edges adjacent to x are white. Since there are no white cycles, attach the acyclic collection of white edges that are adjacent to x.
- For a black vertex x of $T_v(L^k)$, if x is not a cut vertex (that is, x is an element of only one black cycle C) where x is the first element of C to be added to the tree, append the path through C starting at x (but do not add the edge that comes back into x, to maintain acyclicity) There are two directions that one could choose to walk along C; arbitrarily choose the counterclockwise traversal. Call this vertex x the *lead vertex* of C in $T_v(L^k)$.

- For a black vertex x of $T_v(L^k)$, if x is a cut vertex that is an element of different cycles $C_1, \ldots C_\ell$ not already in the tree, then (as in the previous case) append the path around each C_i in the counterclockwise direction, and make x the *lead vertex* of C_i in $T_v(L^k)$.
- Repeat until $T_v(L^k)$ spans L^k.

The important properties are (1) In a depth-first traversal of $T_v(L^k)$, when the lead vertex of a cycle C is encountered, all of the other vertices of C are visited before any other vertex is visited, and (2) the lead vertex for each cycle is determined entirely by v. We can thus denote this by $lead(C, v)$.

In order to see that $T_v(L^k)$ can be constructed in logspace, note first that (using reachability in undirected graphs [Rei08]) it is easy to find all of the cut vertices (see [AM04]) and hence to list all of the cycles, and to create a graph $T'_v(L^k)$ that is a spanning tree of the graph that results when each cycle in L^k is contracted to a single vertex. $T_v(L^k)$ is trivial to compute, given $T'_v(L^k)$.

Each $L^{k+1} \in \mathcal{L}^{k+1}$ is contained in a black cycle C in $\mathcal{L}^k$. There may be more than one connection between C and L^{k+1}. A contribution of [Hag90] is to show that there is an edge which we denote $e(v, L^{k+1})$ connecting C to some vertex u in L^{k+1} such that that a depth-first search tree for G incorporating $T_v(L^k)$ and $T_u(L^{k+1})$ can be constructed using the edge $e(v, L^{k+1})$. (Some figures are provided toward the end of this article, which help illustrate the process by which LCCs are attached to the surrounding black cycle.) By induction, if we know the root of the depth-first tree for L^1, this determines the lead vertex of every black cycle in G and thus also uniquely determines the connection between every black cycle C and every LCC L connected to C. Let us denote this connection $e'(r, L)$, where $e'(r, L^{k+1})$ is $e(v, L^{k+1})$ for the choice $v = e'(r, L^k)$. Connecting the forest consisting of the trees $T_x(L)$ for various x and L along with the edges $e'(r, L)$ results in a depth-first tree for G.

We can now describe the algorithm at a high level:

1. Given an undirected graph G and a vertex r, determine if G is planar and connected, and if so, compute an embedding of G in the plane with r on the external face.
2. Construct the face-vertex incidence graph G'_D of G. This is the undirected bipartite graph on vertex set $V \cup \mathcal{F}$ that contains an edge $\{u, F\}$, for all $u \in V$ and $F \in \mathcal{F}$, exactly if F is incident on u in G. G'_D is clearly planar.
3. For each face of G (i.e., each vertex of G_D), compute its distance from the external face F_0. This is clearly half of its distance from the external face in G'_D, which is planar. This gives us distances in G_D.
4. Using this distance information, compute the type of each vertex and edge of G. This allows us to partition the vertex set of G into the graphs $\mathcal{L}^k$.
5. Partition each $\mathcal{L}^k$ into connected components, to obtain the set $\mathcal{LCC}^{k+1}$.
6. For each $L^k \in \mathcal{LCC}^{k+1}$ and for each v in L^k construct $T_v(L^k)$.

7. For each k, for each $L^{k+1} \in \mathcal{LCC}^{k+1}$, identify the black cycle C in $\mathcal{L}^k$ that contains L^{k+1}.

8. Construct a tree S whose nodes consist of vertices v_C for each black cycle of G and v_L for each LCC L. The root of S is the unique LCC $L^1 \in \mathcal{L}^1$. (It is unique because G is connected.) The children of any LCC $L^k \in \mathcal{L}^k$ are the black cycles in L^k. The children of any black cycle C in $\mathcal{L}^k$ are the LCCs L^{k+1} that are in the interior of C.

9. Label each edge (L, C) in the tree S with the name of the lead vertex of the black cycle C in the depth-first traversal of G starting at r, and label each edge (C, L) in the tree S with the name of the vertex of L that is traversed in the edge $e'(r, L)$.

10. Start constructing the tree T by incorporating the tree $T_r(L^1)$ (rooted at the root r).

11. Output the tree T that is composed of the trees $T_v(L)$ for each $L \in \bigcup_k \mathcal{L}^k$ and each $v \in L$, along with the edges $e'(r, L^{k+1})$ for each $L^{k+1} \in \mathcal{L}^{k+1}$, using the information computed in step 8. (Discard all of the trees $T_v(L)$ that are not connected to T, i.e., those trees where v is the "wrong" root.)

The first step is computable in logspace [AM04, Rei08]. The second step is computable in logspace. The third step is computable in logspace with an oracle for computing distance in planar graphs. Each of the remaining steps is easy to compute in logspace, using the fact that reachability in undirected graphs is logspace-computable [Rei08]. The step that requires the most explanation is step 9. In order to label an edge (L, C) in the tree, start a traversal of $T(r, L^1)$, which allows us in logspace to compute the lead vertex of each black cycle C' in $\mathcal{L}^1$. If C is not one of these cycles C', then the edge (L, C) is a descendent of one such C', and we can determine which one in logspace. Knowing the lead vertex of C', we can determine the edge of G that connects this cycle C' to the LCC L' such that the edge (C', L') is followed from the root of S to (L, C). Knowing this edge, we can determine the lead vertex of each cycle C'' such that (L', C'') is an edge of S. If C is also not any one of these cycles C'', then again we can in logspace determine the next node of S that needs to be traversed, and we can continue in this way until we finally reach the edge (L, C). It is important to note that we do not need to maintain the entire list of lead vertices that are encountered along the way. Note also that this procedure also gives us the information that is required to label vertices of the form (C, L). This procedure is repeated for each edge of S (and thus the lead vertices of each black cycle are recomputed many times).

The argument showing that the tree produced in this way is a depth-first tree is similar to the argument in [Hag90].

This completes the proof.

6 Other Applications

As direct applications of the $\mathsf{CREW}[\log n]$ algorithms for planar Depth-First and Breadth-First search, we directly obtain $\mathsf{CREW}[\log n]$ algorithms for problems such as finding strongly-connected (or 2-connected, 3-connected, etc.) components in planar graphs. Below we discuss some more applications.

6.1 Minimum $s - t$ Cut

Here we show that the minimum $s - t$ cut problem in undirected planar graphs logspace-reduces to finding shortest paths in the same class of graphs, via the well-known technique of cut/cycle duality [IS79]. Given an undirected graph with a *source* vertex s and a *sink* vertex t, an $s-t$ cut is a minimal set of edges X whose removal disconnects s and t. The goal here is to find an $s - t$ cut of minimum size. From the Max Flow-Min Cut Theorem this also corresponds to the size of the maximum $s - t$ flow in the graph. It was known that the problem is in $\mathsf{CRCW}[\log^2 n]$ [Joh87]. In this paper we show that it can be solved in $\mathsf{CREW}[\log n]$.

Theorem 14. *The minimum $s - t$ cut problem in undirected planar graphs logspace-Turing reduces to* PlDist.

Proof. We follow the proof technique of [IS79] and show that other than shortest path computation all other steps can be carried out in L. Let $G = (V, E)$ be the given undirected planar graph with two designated vertices s, t. Let $G^* = (V^*, E^*)$ be the dual graph of G. Consider the faces in G^* containing the vertices s and t and denote them f_s^* and f_t^*, respectively. It is not hard to show that if X is the minimum $s - t$ cut then $X^* = \{v^* | v \in X\}$ is a cycle in G^* of minimum length enclosing t^*. Such a cycle is called a *cut-cycle.*

Next, find the shortest path between some $s^* \in f_s^*$ and some $t^* \in f_t^*$ in G^* and call it $\Pi = \{v_1^*, v_2^*, \ldots, v_\ell^*\}$. (This is easy to do in logspace if one has a Breadth-First Search (BFS) tree, which can be obtained with an oracle for PlDist, by Corollary 4. Assume an ordering on the edges on Π from s^* towards t^*. Let O_Π^* denote the set of edges that have exactly one end point on Π. Consider any such edge $(v_i^*, u^*) \in O_\Pi^*$. Let us say that (v_i^*, u^*) is Π-left if u^* occurs after v_{i-1}^* and before v_{i+1}^* in the clockwise order of neighbours around v_i^*. Otherwise call it Π-right. Call a cut-cycle *nice* if it contains exactly one Π-left edge and one Π-right edge. It was shown in [IS79] that, among all of the shortest cut-cycles, there exists at least one shortest cut-cycle that is nice.

To find such a cycle we do the following. Direct the Π-left edges away from the vertices on Π and the Π-right edges towards the Π vertices. All the other edges are replaced by bi-directed edges. Call this graph $\vec{G^*}$. Now for each $v_i^* \in \Pi$ find the shortest directed path to itself which gives the shortest cycle enclosing t and so the corresponding primal edges give the minimum $s - t$ cut. □

Corollary 15. *Given an n vertex undirected planar graph with two specified vertices s, t the minimum $s - t$ cut can be found in* $\mathsf{CREW}[\log n]$.

6.2 Maximal Matching

In this section, we consider the problem of finding a maximal matching in an undirected planar graph. This should not be confused with the problem of finding a *maximum* matching in a planar graph, which has also been studied recently [DKKM18], as has the related problem of finding a perfect matching in a planar graph [AV18].

Theorem 16. *A maximal matching of an undirected planar graph with n vertices can be computed in* $\mathsf{CREW}[\log n]$.

Proof. Let $G = (V, E)$ be the given undirected planar graph. Find a Depth-First Search tree T of G using our $\mathsf{CREW}[\log n]$ algorithm from Section 5. Next, using Lemma 17 we find a matching that covers all internal vertices of the tree T.

It is easy to see that such a matching is maximal, because every edge in the graph is either a tree edge (and therefore has an internal vertex as an endpoint) or a non-tree edge (which, by the properties of a Depth-First Search tree, must have an internal vertex as an endpoint).

Lemma 17 shows that a matching of the tree covering all internal vertices can be computed in $\mathsf{LogDCFL} = \mathsf{CROW}[\log n]$, and hence the cost of computing the matching is dominated by the cost of computing the Depth-First Search tree, which can be done in $\mathsf{CREW}[\log n]$. □

Lemma 17. *Given an undirected planar graph G and a spanning tree T of G, a matching covering all the internal vertices in T can be found in* $\mathsf{LogDCFL}$.

Proof. We assume that the spanning tree T, rooted at a vertex r, is presented in the following form: for each vertex u a linked list of all children of u is given. The "eldest" child of u is provided by the link $e(u)$, and given any child v, the "next" sibling is reached by the link $\text{next}(v)$. A logspace-bounded APDA running in polynomial time can easily execute the following recursive pseudocode:

```
def match(r,already_matched):
       If r is a leaf, then return.
       Else v = e(r).
       If already_matched is true
             then \% try to find a match for v
                 match(v,false)
       Else \% match r with v, and continue
             output ((r,v))
             match(v,true)
       While v has a next sibling
             v = next(v)
             match(v,false)
       Endwhile
       return.
```

It is easy to see that the edges produced as output constitute a partial matching. Some leaves might not be included in the matching, but every internal vertex is covered. An APDA can easily implement the recursive algorithm, using the stack to store the current value of v and r. The runtime is easily seen to be polynomial in the size of T. $\square$

This also shows that a 2-factor approximation of the minimum vertex cover problem on planar graphs can be obtained in $\mathsf{CREW}[\log n]$. However, as we will see next, for minimum vertex cover and a host of other NP-hard optimization problems on planar graphs there is actually an approximation scheme computable in $\mathsf{CREW}[\log n]$.

6.3 Approximation Schemes via Baker's Method

Here we present approximation schemes, that is, $(1 \pm \epsilon)$ approximation algorithms for every $\epsilon > 0$, running in $\mathsf{CREW}[\log n]$ for a class of monadic-second-order-definable (MSO-definable) optimization problems on planar graphs which are amenable to Baker's method [Bak94].

There are two main general approaches for designing a Polynomial-Time Approximation Scheme (PTAS) for problems on planar graphs. The first approach is based on planar separators [LT80]. The second approach, first introduced by Baker [Bak94] and known as the *shifting technique*, is based on decomposition into overlapping subgraphs of bounded outerplanarity, which are of bounded treewidth. For a general account on these, see [DH08].

Baker's method was originally designed to give PTASs for a host of NP-hard optimization problems on planar graphs, such as minimum vertex cover, minimum dominating set, maximum independent set, etc., which are hard to approximate in general graphs. Many of these remain NP-hard even in planar graphs. Later the technique was generalized to a broader class of graphs called graphs of bounded local treewidth [Epp00, DH04].

Baker's Algorithm The main two computational parts are,

1. decomposing the graph into bounded treewidth graphs, and
2. solving the optimization problem on bounded treewidth graphs optimally and combining these solutions.

Step (1) requires performing BFS on the graph G and considering the induced subgraphs $G_{i,j}$ (which can be seen to have treewidth $O(k)$) that lie between layers $ki + j$ and $k(i + 1) + j$, for $i \geq 0$ and offset $0 \leq j \leq k - 1$. These subgraphs are formed by deleting (or including in both the adjacent slices) the vertices/edges in every $k = O(1/\epsilon)$-th BFS layer. (Particular details differ, depending on the problem). By choosing the right offset, we can make sure this affects the optimum solution at most by a factor of ϵ.

For Step (2), given an MSO-definable problem, using Bodlaender's theorem [Bod96] and Courcelle's theorem [Cou90] on bounded treewidth graphs, one can find the optimal solution in polynomial time.

As observed by [DK14], to get efficient parallel algorithms, for Step (2) above one can make use of the work of [EJT10], which proves a logspace version of Bodlaender's and Courcelle's theorems. The distance computation in Step (1) dominates the complexity bound. As a consequence of Corollary 4, we obtain the following:

Theorem 18. *MSO-definable optimization problems which are amenable to Baker's method have* $\mathsf{CREW}[\log n]$ *approximation schemes in planar graphs.*

7 Conclusion

The reachability problem for planar graphs is one of the most prominent examples of a problem in NL that is known to be hard for L [Ete97] and is not known to be hard for NL. After Bourke, Tewari, and Vinodchandran showed that the problem lies in UL, the isomorphism problem for an important class of planar graphs was shown to lie in UL [TW10a], before ultimately planar graph isomorphism was shown to be complete for L [DLN$^+$09]. A natural question is whether planar graph reachability is also destined to find a home in L.

At some level, it is surprising that so many natural computational problems turn out to be complete for one of the complexity classes that arise using a small vocabulary of notions from circuit complexity and resource-bounded nondeterministic or alternating Turing machines. There is no strong reason to believe that planar reachability will also turn out to be complete for one of these "standard" complexity classes, no matter how much our prior experience with other problems has conditioned us to expect that this is the "normal" outcome.

Our main contribution is to give a new upper bound on the complexity of planar reachability: a bound that is not widely believed to hold for NL-complete problems. It will be interesting to see if additional upper bounds can be placed on planar reachability; does it lie in $\mathsf{LogDCFL}$? Or in USAC^1?

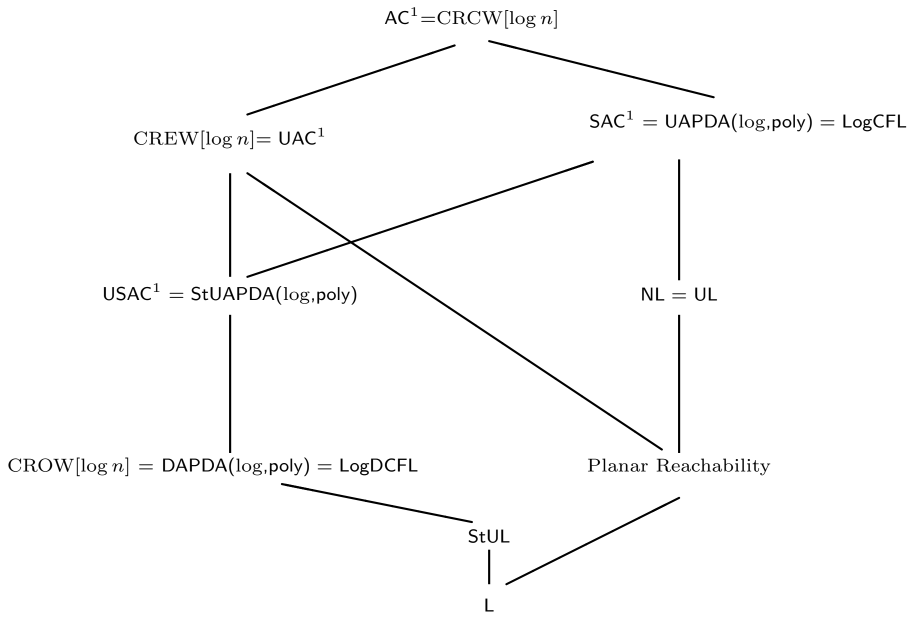
A second contribution is to show that the characterization of $\mathsf{CREW}[\log n]$ in terms of unambiguous circuits can be useful in the design of PRAM algorithms, which we have illustrated by presenting the first $\mathsf{CREW}[\log n]$ algorithms for depth-first search trees, maximum flow in planar graphs, edit distance, and other problems.

We do not believe that this is the final word on the complexity of planar reachability, and we look forward to further developments.

References

[AALM90] Alberto Apostolico, Mikhail J. Atallah, Lawrence L. Larmore, and Scott McFaddin. Efficient parallel algorithms for string editing and related problems. *SIAM J. Comput.*, 19(5):968–988, 1990.

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity, a modern approach.* Cambridge University Press, 2009.



AC^1=CRCW[$\log n$]
CREW[$\log n$]= UAC^1
SAC^1 = UAPDA(log,poly) = LogCFL
USAC^1 = StUAPDA(log,poly)
NL = UL
CROW[$\log n$] = DAPDA(log,poly) = LogDCFL
Planar Reachability
StUL
L

Figure 1: Relations among the complexity classes under consideration, under the plausible assumption that there is a set in $\mathsf{DSPACE}(n)$ that requires circuits of size $2^{\epsilon n}$ for all large n.

[AB17] Amir Abboud and Arturs Backurs. Towards hardness of approximation for polynomial time problems. In *8th Innovations in Theoretical Computer Science Conference, (ITCS)*, pages 11:1–11:26, 2017.

[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.

[AHWW16] Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. Simulating branching programs with edit distance and friends: or: a polylog shaved is a lower bound made. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, (STOC)*, pages 375–388, 2016.

[AKNW14] Tetsuo Asano, David G. Kirkpatrick, Kotaro Nakagawa, and Osamu Watanabe. $\tilde{\mathrm{O}}(\sqrt{\mathrm{n}})$-space and polynomial-time algorithm for planar directed graph reachability. In *Mathematical Foundations of Computer Science (MFCS)*, volume 8635 of *Lecture Notes in Computer Science*, pages 45–56. Springer, 2014.

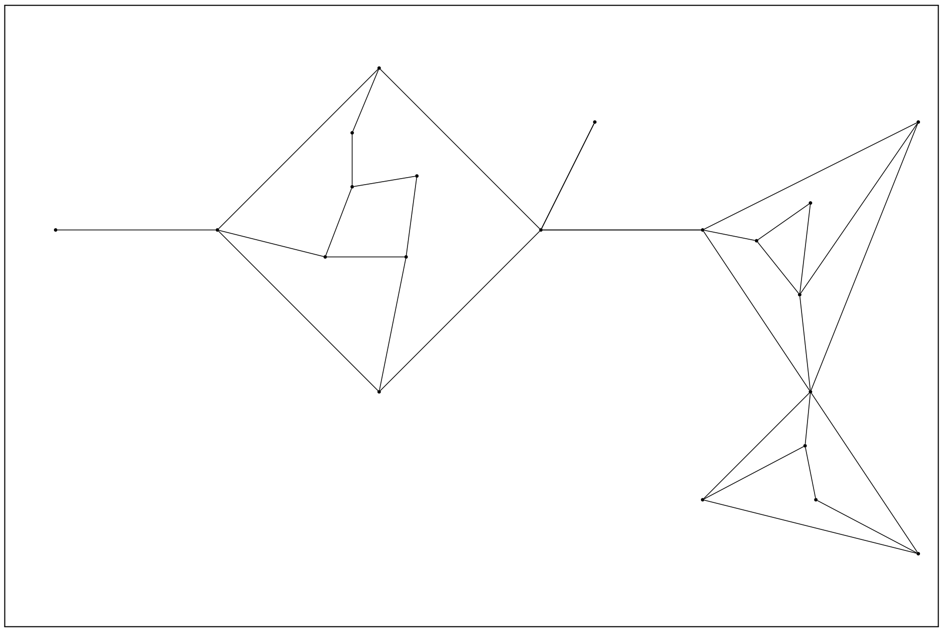


Figure 2: An example of a connected planar graph

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

[AM04] Eric Allender and Meena Mahajan. The complexity of planarity testing. *Inf. Comput.*, 189:117–134, 2004.

[ARZ99] Eric Allender, Klaus Reinhardt, and Shiyu Zhou. Isolation, matching, and counting: Uniform and nonuniform upper bounds. *Journal of Computer and System Sciences*, 59(2):164–181, 1999.

[AV18] Nima Anari and Vijay V. Vazirani. Planar graph perfect matching is in NC. In *Proc. 59th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 650–661, 2018.

[Bak94] B. S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *J. ACM*, 41(1):153–180, 1994.

[BI18] Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). *SIAM J. Comput.*, 47(3):1087–1097, 2018.

[Bod96] H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.

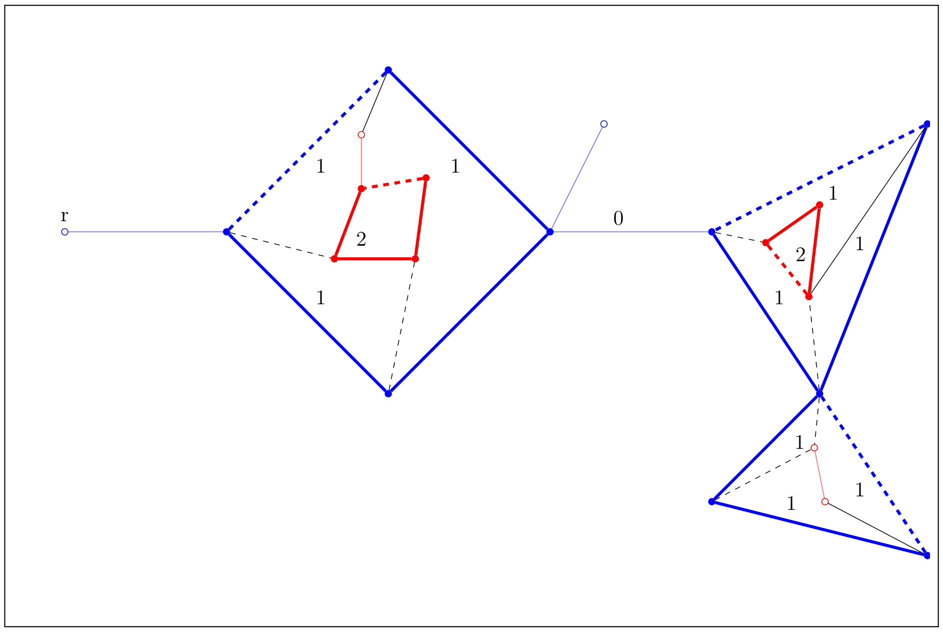


Figure 3: The example graph with layers marked. Blue edges and vertices belong to layer 1. Red edges and vertices belong to layer 2. Black edges are not part of any layer. Dotted lines are non-tree edges of a DFS tree rooted at vertex labelled r. The direction we choose for traversing cycles is counterclockwise.

[BS97] K. Nandan Babu and Sanjeev Saxena. Parallel algorithms for the longest common subsequence problem. In *Proceedings of the Fourth International on High-Performance Computing, HiPC 1997, Bangalore, India, 18-21 December, 1997*, pages 120–125, 1997.

[BTV09] Chris Bourke, Raghunath Tewari, and N. V. Vinodchandran. Directed planar reachability is in unambiguous log-space. *TOCT*, 1(1):4:1–4:17, 2009.

[CDG$^+$18] Diptarka Chakraborty, Debarati Das, Elazar Goldenberg, Michal Koucký, and Michael E. Saks. Approximating edit distance within constant factor in truly sub-quadratic time. In *59th IEEE Annual Symposium on Foundations of Computer Science, (FOCS)*, pages 979–990, 2018.

[Coo71] Stephen A. Cook. Characterizations of pushdown machines in terms of time-bounded computers. *J. ACM*, 18(1):4–18, 1971.

[Cou90] Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Information and Computation*, 85(1):12 – 75, 1990.

[DH04] E. D. Demaine and M. Taghi Hajiaghayi. Equivalence of local treewidth and linear local treewidth and its algorithmic applications. In *Fifteenth SODA 2004, New Orleans, Louisiana, USA, January 11-14, 2004*, pages 840–849, 2004.

[DH08] E. D. Demaine and M. Taghi Hajiaghayi. *Approximation Schemes for Planar Graph Problems*, pages 1–99. Springer US, Boston, MA, 2008.

[DK14] S. Datta and R. Kulkarni. Space complexity of optimization problems in planar graphs. In *11th Annual Conference, TAMC 2014, Chennai, India, April 11-13, 2014. Proceedings*, pages 300–311, 2014.

[DKKM18] Samir Datta, Raghav Kulkarni, Ashish Kumar, and Anish Mukherjee. Planar maximum matching: Towards a parallel algorithm. In *Proc. 29th International Symposium on Algorithms and Computation (ISAAC)*, volume 123 of *LIPIcs*, pages 21:1–21:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

[DLN$^+$09] Samir Datta, Nutan Limaye, Prajakta Nimbhorkar, Thomas Thierauf, and Fabian Wagner. Planar graph isomorphism is in log-space. In *Proceedings of the 24th Annual IEEE Conference on Computational Complexity (CCC)*, pages 203–214, 2009.

[DR00] Patrick W. Dymond and Walter L. Ruzzo. Parallel rams with owned global memory and deterministic context-free language recognition. *J. ACM*, 47(1):16–45, 2000.

[EJT10] Michael Elberfeld, Andreas Jakoby, and Till Tantau. Logspace versions of the theorems of Bodlaender and Courcelle. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 143–152, 2010.

[EK14] Michael Elberfeld and Ken-ichi Kawarabayashi. Embedding and canonizing graphs of bounded genus in logspace. In *ACM Symposium on Theory of Computing (STOC)*, pages 383–392, 2014.

[Epp00] D. Eppstein. Diameter and treewidth in minor-closed graph families. *Algorithmica*, 27(3):275–291, 2000.

[Ete97] Kousha Etessami. Counting quantifiers, successor relations, and logarithmic space. *Journal of Computer and System Sciences*, 54(3):400–411, Jun 1997.

[Hag90] Torben Hagerup. Planar depth-first search in O(log n) parallel time. *SIAM J. Comput.*, 19(4):678–704, June 1990.

[HRS93] Magnús M. Halldórsson, Jaikumar Radhakrishnan, and K. V. Subrahmanyam. Directed vs. undirected monotone contact networks for threshold functions. In *34th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 604–613, 1993.

[IS79] Alon Itai and Yossi Shiloach. Maximum flow in planar networks. *SIAM J. Comput.*, 8(2):135–150, 1979.

[Joh87] Donald B. Johnson. Parallel algorithms for minimum cuts and maximum flows in planar networks. *J. ACM*, 34(4):950–967, October 1987.

[JT19] Rahul Jain and Raghunath Tewari. Grid graph reachability. *CoRR*, abs/1902.00488, 2019.

[KV10] Jan Kynčl and Tomáš Vyskočil. Logspace reduction of directed reachability for bounded genus graphs to the planar case. *ACM Transactions on Computation Theory (TOCT)*, 1(3):8:1–8:11, 2010.

[KvM02] Adam R. Klivans and Dieter van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. *SIAM J. Comput.*, 31(5):1501–1526, 2002.

[Lan93] Klaus-Jörn Lange. Unambiguity of circuits. *Theor. Comput. Sci.*, 107(1):77–94, 1993.

[LL94] M. Lu and H. Lin. Parallel algorithms for the longest common subsequence problem. *IEEE Trans. Parallel Distrib. Syst.*, 5(8):835–848, August 1994.

[LMN10] Nutan Limaye, Meena Mahajan, and Prajakta Nimbhorkar. Longest paths in planar dags in unambiguous log-space. *Chicago J. Theor. Comput. Sci.*, 2010.

[LT80] R. J. Lipton and R. E. Tarjan. Applications of a planar separator theorem. *SIAM J. Comput.*, 9(3):615–627, 1980.

[Mat88] Thomas R. Mathies. A fast parallel algorithm to determine edit distance. Technical Report CMUCS-88-130, Carnegie Mellon University, 1988.

[RA00] Klaus Reinhardt and Eric Allender. Making nondeterminism unambiguous. *SIAM J. Comput.*, 29(4):1118–1131, 2000.

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

[STV95] Sairam Subramanian, Roberto Tamassia, and Jeffrey Scott Vitter. An efficient parallel algorithm for shortest paths in planar layered digraphs. *Algorithmica*, 14(4):322–339, 1995.

[Sud78] Ivan Hal Sudborough. On the tape complexity of deterministic context-free languages. *Journal of the ACM*, 25(3):405–414, 1978. Preliminary version in STOC'76.

[SV84] L. Stockmeyer and U. Vishkin. Simulation of parallel random access machines by circuits. *SIAM Journal on Computing*, 13:409–422, 1984.

[TV12] Raghunath Tewari and N. V. Vinodchandran. Green's theorem and isolation in planar graphs. *Inf. Comput.*, 215:1–7, 2012.

[TW10a] T. Thierauf and F. Wagner. The isomorphism problem for planar 3-connected graphs is in unambiguous logspace. *Theory Comput. Syst.*, 47(3):655–673, 2010.

[TW10b] Thomas Thierauf and Fabian Wagner. The isomorphism problem for planar 3-connected graphs is in unambiguous logspace. *Theory Comput. Syst.*, 47(3):655–673, 2010.

[Ven91] H. Venkateswaran. Properties that characterize LOGCFL. *J. Comput. Syst. Sci.*, 43(2):380–404, 1991.

[Vol99] H. Vollmer. *Introduction to Circuit Complexity: A Uniform Approach.* Springer-Verlag New York Inc., 1999.