CHICAGO JOURNAL OF THEORETICAL COMPUTER SCIENCE 2010, Article 8, pages 1–16
http://cjtcs.cs.uchicago.edu/

SPECIAL ISSUE FOR CATS 2009

Longest Paths in Planar DAGs in Unambiguous Log-Space *

Nutan Limaye Meena Mahajan Prajakta Nimbhorkar

Received: January 15, 2009; revised: December 2, 2009; published: June 22, 2010.

Abstract: We present a transformation from longest paths to shortest paths in sub-classes of directed acyclic graphs (DAGs). The transformation needs log-space and oracle access to reachability in the same class of graphs. As a corollary, we obtain our main result: Longest Paths in planar DAGs is in UL ∩ co-UL. The result also extends to toroidal DAGs. Further, we show that Longest Paths in max-unique DAGs where the target node is the unique sink is in UL ∩ co-UL.

We show that for planar DAGs with the promise that the number of distinct paths is bounded by a polynomial, counting paths can be done by an unambiguous pushdown automaton equipped with an auxiliary log-space worktape and running in polynomial time. This bound also holds if we want to compute the number of longest paths, or shortest paths, and this number is bounded by a polynomial (irrespective of the total number of paths). Along the way, we show that counting paths in general DAGs can be done by a deterministic pushdown automaton with an auxiliary log-space worktape and running in polynomial time, and hence is in the complexity class LogDCFL, provided the number of paths is bounded by a polynomial and the target node is the only sink.

Key words and phrases: directed acyclic graphs, reachability, longest path, planar, unambiguous log-space

1 Introduction

Consider the following problems in directed graphs.

*These results were presented at the Fifteenth Computing: The Australasian Theory Symposium (CATS2009).

DOI: 10.4086/cjtcs.2010.008

$$\begin{aligned}
\text{Reach} &= \{\, (G,s,t) \mid G \text{ contains a path from } s \text{ to } t \,\} \\
\text{Distance} &= \{\, (G,s,t,k) \mid G \text{ contains a path of length} \leq k \text{ from } s \text{ to } t \,\} \\
\text{Long-Path} &= \{\, (G,s,t,k) \mid G \text{ has a simple path of length} \geq k \text{ from } s \text{ to } t \,\} \\
\#\text{Path} &= \{\, (G,s,t,1^k) \mid G \text{ has exactly } k \text{ simple paths from } s \text{ to } t \}
\end{aligned}$$

These problems have widely differing complexities: some of the results below are folklore, some are recent advances. Reach for general graphs is complete for the class NL of languages accepted by nondeterministic log-space machines, and remains NL-hard even if the graphs are acyclic. For undirected graphs, it is complete for the class L of languages accepted by deterministic log-space machines [8], and is sandwiched between L and UL $\cap$ co-UL for planar directed graphs [4]. Here, the class UL is the class of languages accepted by unambiguous nondeterministic log-space machines, and co-UL is the class of complements of languages in UL. Clearly, Distance and Long-Path are at least as hard as Reach: $(G,s,t) \in$ Reach if and only if $(G,s,t,n) \in$ Distance if and only if $(G,s,t,0) \in$ Long-Path, where n is the number of vertices in G. Distance is NL-complete for general graphs, and remains NL-hard even if the graphs are acyclic, or if the graphs are undirected [11], but it is in UL $\cap$ co-UL for planar directed graphs [12]. Long-Path is NP-complete for general graphs, since it includes Hamiltonian paths as a special case. It remains NP-hard for planar undirected graphs. However, for directed acyclic graphs (DAGs), there is a deterministic linear time algorithm based on dynamic programming. The problem is also known to be NL-complete for DAGs. However its complexity for planar DAGs is, to the best of our knowledge, not yet studied.

In this note we consider this combination of planarity and acyclicity for Long-Path, and denote this restriction of the problem by Long-Path(Planar DAG). All the graphs we consider here onwards are directed. One of our main results is:

Theorem 1.1. Long-Path*(Planar DAG)* $\in$ UL $\cap$ co-UL.

Thus for planar DAGs, Long-Path shares the current best-known upper bounds for Reach and Distance.

Our proof of this result is a reduction from Long-Path to Distance and vice versa (Theorem 3.2), where the reduction uses log-space and oracle access to Reach. The reduction works for fairly large sub-classes of DAGs. In particular, it works over planar DAGs. A recent result in [7] shows that for an important subclass of planar DAGs, namely series-parallel graphs, the three problems of Reach, Distance, and Long-Path are indeed equivalent and are all L-complete. Our Theorem 3.2 is a finer analysis of their consruction.

Next, we give an unambigous log-space algorithm (Theorem 4.1) to compute Long-Path for graphs which are *max-unique* (for any pair of vertices, if there is a path between them, then there is a unique longest path) and have the target as a unique sink node. Our algorithm uses the technqiue of double inductive counting, first used in [9]. In conjunction with the techniques of [4], by which a planar DAG can be made max-unique, we obtain another proof of Theorem 1.1.

For graphs with embeddings on the torus, it is shown in [1] that reachability is no harder than planar reachability. We observe that Distance and Long-Path are also no harder than the planar versions (Corollary 5.1).

In the search version of Long-Path, the input consists of a DAG G and vertices s and t. The desired output is a sequence of vertices forming a longest path between s and t. We show that the search version of Long-Path shares the same upper bound for planar DAGs (Lemma 5.2).

We also address the problem of counting simple paths in graphs. In general graphs, this problem is #P-hard and remains so even for planar graphs. We consider its restriction on DAGs. The problem of counting paths between two designated nodes s and t in a DAG is known to be complete for #L, the class of functions that count accepting paths in NL machines. In [3] a restriction of this problem is considered, when the number of such paths is bounded by a polynomial, and an NL upper bound for this restriction is obtained. We consider this restriction combined with other restrictions and give new bounds, which are incomparable to the known ones. We consider the case when the DAG not only has a polynomially bounded number of s to t paths but also t is the unique sink in the DAG. We show (Theorem 1.2) that in this case, the number of paths can be computed by a deterministic AuxPDA (a pushdown automaton with an auxiliary log-space worktape; equivalently, a log-space machine augmented with a stack) running in polynomial time, namely, DAuxPDA. It is known that such machines accept exactly the class LogDCFL of languages reducible via log-space many-one reductions to some deterministic context-free language [10].

Theorem 1.2. *Let $c > 0$ be a fixed constant. There is a* DAuxPDA *that, given a DAG G with n nodes and a unique sink t, and given a node s in G such that the number of paths from s to t in G is bounded by n^c, computes this number in polynomial time.*

Another restriction is planarity. From Theorems 1.1, 1.2, it follows easily that in planar DAGs where the number of s to t paths is bounded by a polynomial, this number can be determined by an unambiguous pushdown automaton that uses a log-space auxiliary work-tape and runs in polynomial time (UAuxPDA). For planar DAGs, we also consider the problem of computing the number of longest or shortest s to t paths when this number is bounded by a polynomial, irrespective of the total number of paths. We show that this too can be done by a UAuxPDA. Thus we have the following theorem, which we prove in Section 5.4:

Theorem 1.3. *Let $c > 0$ be a fixed constant.*

1. *There is a nondeterministic* AuxPDA *that, given a planar directed acyclic graph G with n nodes, and given nodes s and t in G such that the number of paths from s to t in G is bounded by n^c, proceeds unambiguously (rejects on all except one path) and computes this number in polynomial time.*

2. *There is a nondeterministic* AuxPDA *that, given a planar directed acyclic graph G with n nodes, and given nodes s and t in G such that the number of longest paths from s to t in G is bounded by n^c, proceeds unambiguously and computes this number in polynomial time. The same is true for shortest paths.*

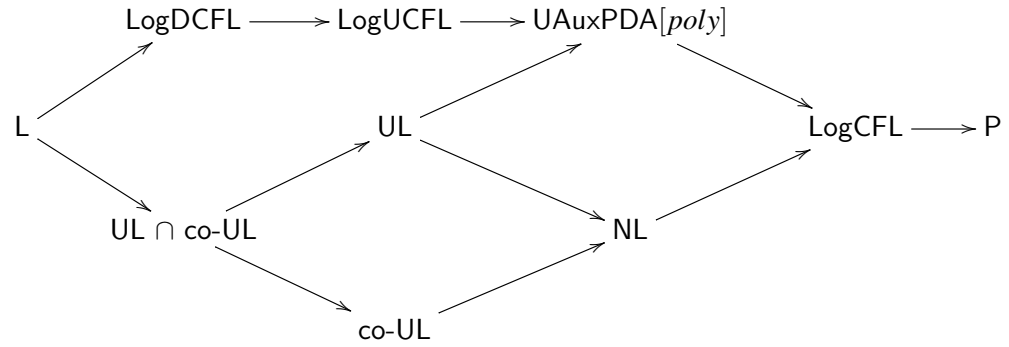
This paper is organised as follows. In Section 2 we state some known results that we use, and also describe preprocessing steps that we use in our algorithms. Section 3 describes the algorithm, based on [7] and [12], that establishes Theorem 1.1. Section 4 describes the algorithm for Long-Path in max-unique DAGs using double inductive counting. In Section 5, algorithms for Long-Path in toroidal DAGs, for the search version of Long-Path, and for the counting versions under a polynomial bound promise (Theorems 1.2, 1.3) are described.

2 Preliminaries

By L and NL, we denote the classes of languages accepted by deterministic and nondeterministic log-space machines, respectively. LogCFL and LogDCFL denote the classes of languages log-space many-one reducible to some (deterministic) context-free language.

An AuxPDA is an auxiliary pushdown automaton, that is, a pushdown automaton augmented with a log-space worktape. Alternatively, it can be viewed as a log-space machine augmented with a stack. If the machine is deterministic, it is called a DAuxPDA. The class of languages accepted by AuxPDA running in polynomial time is known to be exactly the language class LogCFL; similarly, the class of languages accepted by a DAuxPDA running in polynomial time is exactly the language class LogDCFL.

A nondeterministic machine is said to be *unambiguous on an input* if it has at most one accepting path on that input. It is said to be *unambiguous* if it is unambiguous on every input. UL denotes the class of languages accepted by unambiguous log-space machines; co-UL is the class of complements of languages in UL. LogUCFL denotes the class of languages log-space many-one reducible to some unambiguous context-free language (at most one parse tree per word), and is known to be contained in the class of languages accepted by polynomial time unambiguous AuxPDA, denoted UAuxPDA[poly]. The following relationships hold among these language classes:



The operator $\oplus$ represents languages expressible as the marked union of languages from the two classes. That is, for languages A and B, $A \oplus B$ is the language $\{0w \mid w \in A\} \cup \{1w \mid w \in B\}$, and for language classes $\mathcal{C}_1$ and $\mathcal{C}_2$, their marked union is $\mathcal{C}_1 \oplus \mathcal{C}_2 = \{A \oplus B \mid A \in C_1, B \in C_2\}$.

For any subclass $\mathcal{C}$ of graphs, let Reach($\mathcal{C}$), Distance($\mathcal{C}$), and Long-Path($\mathcal{C}$) denote the restriction of these problems to instances from $\mathcal{C}$. For directed acyclic graphs,

$$(G,s,t) \in \text{Reach} \Leftrightarrow (G,s,t,|V|) \in \text{Distance} \Leftrightarrow (G,s,t,0) \in \text{Long-Path}.$$

So Distance($\mathcal{C}$) and Long-Path($\mathcal{C}$) are at least as hard as Reach($\mathcal{C}$) for any subclass $\mathcal{C}$ of directed acyclic graphs.

We use the following results:

Lemma 2.1 ([4])**.** Reach*(Planar) is in* UL ∩ co-UL.

Lemma 2.2 ([12])**.** Distance*(Planar) is in* UL ∩ co-UL.

Lemma 2.3 ([7])**.** Distance*(Series-parallel) and* Long-Path*(Series-parallel) are equivalent.*

Lemma 2.4 ([1])**.** Reach*(Torus) reduces via log-space many-one reductions to* Reach*(Planar).*

Remark 2.5. Consider any directed acyclic instance (G,s,t,k) of Distance or Long-Path. By (parallel) queries of the form (G,s,u) or (G,u,t) to Reach, we can remove all vertices that do not figure on some s to t path. Thus in $\mathsf{L}^{\mathsf{Reach}}$ we can obtain a single-source (s) single-sink (t) graph G', and all queries to Reach involve only the graph G. In some of the algorithms, we will use this procedure.

Remark 2.6. From the results of [2, 8], planarity testing can be done in deterministic log-space. By [4], Reach(Planar) is in UL ∩ co-UL; thus checking whether a planar graph is a DAG is in $\mathsf{L}^{\mathsf{UL}\cap\mathsf{co\text{-}UL}}$ and hence in UL ∩ co-UL. Thus a non-deterministic log-space machine that is unambiguous on planar DAGs can be modified with preprocessing to yield a UL machine which outright rejects any graph that is not a planar DAG. Hence, in the following two sections where we prove Theorem 1.1, we only present unambiguous computations on planar DAGs, and conclude that the language

$$\text{Long-Path(Planar DAG)} = \left\{ (G,s,t,k) \;\middle|\; \begin{array}{l} G \text{ is a planar DAG containing a} \\ \text{path from } s \text{ to } t \text{ of length at least} \\ k \end{array} \right\}$$

is in UL ∩ co-UL. Similarly, the proofs of Lemma 5.2 and 5.3 should be considered in conjunction with such preprocessing.

A graph is said to be *max-unique* if for each pair u,v of nodes, if there is a path from u to v, then the longest path from u to v is unique. Similarly, a graph is *min-unique* if for each pair u,v, if there is a path from u to v, then the shortest path from u to v is unique.

3 Reducing Longest Paths to Shortest Paths in DAGs

Our algorithm for Long-Path in planar DAGs uses a simple extension of Lemma 2.3.

Lemma 3.1. Distance*(Planar DAG) and* Long-Path*(Planar DAG) are equivalent via log-space reductions that have access to the oracle* Reach*(Planar DAG).*

Clearly, Theorem 1.1 follows from Lemmas 2.1, 2.2, and 3.1.

The proof of Lemma 3.1 is a generalisation of the proof given in [7] for series-parallel graphs. The generalisation in fact works for any class of acyclic graphs that is closed under subdivision and vertex deletion. In particular, it works for planar DAGs. We present below, in Theorem 3.2, the result of [7] simplified by specialising to unweighted graphs, and stated for such (more general) classes of graphs. Lemma 3.1 is an obvious corollary.

Theorem 3.2. *Let $\mathcal{C}$ be any subclass of directed acyclic graphs closed under subdivisions and vertex deletions. There is a function f, computable in log-space with oracle access to* Reach$(\mathcal{C})$*, that reduces* Distance$(\mathcal{C})$ *to* Long-Path$(\mathcal{C})$ *and* Long-Path$(\mathcal{C})$ *to* Distance$(\mathcal{C})$*.*

Proof. Let $G=(V,E)$ be the given directed acyclic graph, in which we want to find the longest, or the shortest, path between given vertices s and t. Construct a new graph $G'=(V',E')$ as follows:

For each $u \in V$, define $P_u = \{x \in V \mid \text{there is a path from } x \text{ to } u \text{ in } G\}$. Note that u is in P_u for all u. Next define $E_u = \{(x,y) \mid x \in P_u, y \notin P_u\}$. Since G is acyclic, all outgoing edges of u are in E_u.

Let ρ be any s to t path. For every vertex u, ρ has at most one edge from E_u. This can be seen as follows: If there is a path from t to u ($t \in P_u$), then there is a path from every vertex on ρ to u via t, so no edge of ρ is in E_u. If there is no path from s to u, then there is no path from any vertex on ρ to u, so again no edge of ρ is in E_u. Now if $s \in P_u$ but $t \notin P_u$, then along the path ρ, we transit from being in P_u to being outside P_u exactly once. Let this transition occur on edge (x,y). Then (x,y) is in E_u, and no other edge of ρ can be in E_u. Thus

$$|\rho \cap E_u| = \begin{cases} 1, & \text{if } \ s \in P_u \text{ and } t \notin P_u, \\ 0, & \text{otherwise.} \end{cases}$$

To obtain G', we replace each edge $e = (u,v)$ by a path of length l_{uv} determined as follows:

$$\begin{aligned} l_{uv} &= 2\left(\sum_{x \in V : (u,v) \in E_x} \text{out-degree}(x)\right) - 1 \\ &= 2\left(\sum_{x \in V : u \in P_x, v \notin P_x} \text{out-degree}(x)\right) - 1 \end{aligned}$$

Since G is acyclic, the vertex u itself always qualifies in the above sum, and so l_{uv} is positive.

For any pair of vertices s,t we define the quantity K_{st} as follows:

$$K_{st} = \sum_{x \in V : s \in P_x, t \notin P_x} \text{out-degree}(x)$$

Now the crucial claim: for each pair of vertices s,t, each s to t path ρ in G of length $|\rho|$ (in terms of number of edges) is transformed by the above construction to a path in G' of length *exactly* $2K_{st} - |\rho|$. This is because the length of the transformed path is

$$\begin{aligned}
\sum_{\substack{(u,v)\in E\\(u,v)\in\rho}} l_{uv} &= \sum_{\substack{(u,v)\in E\\(u,v)\in\rho}} \left[2\left(\sum_{x\in V: u\in P_x, v\notin P_x} \text{out-degree}(x) \right) - 1 \right] \\
&= 2\left(\sum_{\substack{(u,v)\in E\\(u,v)\in\rho}} \sum_{x\in V: u\in P_x, v\notin P_x} \text{out-degree}(x) \right) - |\rho| \\
&= 2\sum_{x\in V} \left(\text{out-degree}(x) \sum_{e\in\rho\cap E_x} 1 \right) - |\rho| \\
&= 2\left(\sum_{x\in V} \text{out-degree}(x)\cdot|\rho\cap E_x| \right) - |\rho| \\
&= 2\sum_{x\in V:|\rho\cap E_x|=1} \text{out-degree}(x) - |\rho| \;=\; 2K_{st} - |\rho|
\end{aligned}$$

It thus follows that the longest (shortest) path in G is mapped to the shortest (longest, respectively) path in G'. In fact, if the s to t paths are ordered monotonically with respect to length, then the above transformation precisely reverses this ordering. Hence the reduction function f maps (G,s,t,k) to $(G',s,t,2|K_{st}|-k)$.

The reduction can be computed in log-space with oracle access to $\mathsf{Reach}(\mathfrak{C})$, where all queries involve only the graph G. This is because obtaining G' as well as computing K_{st} merely involve finding the sets P_u and E_u and adding up out-degrees. □

4 Computing Longest Paths in DAGs via Double Inductive Counting

The main result of this section is that in max-unique DAGs where t is the unique sink, the length of the unique longest s to t path can be computed unambiguously using log-space. Formally, we prove the following theorem.

Theorem 4.1. *There is a nondeterministic log-space machine M that, given as input a directed acyclic max-unique graph G with a unique sink t, and any other vertex s, finds the length of the longest s to t path unambiguously.*

In [9], double inductive counting is used to unambiguously test reachability in min-unique graphs. In [12], the same technique is used, in combination with the weighting technique from [4], to compute shortest paths in planar graphs. For computing longest paths, however, the technique cannot be used as is, but needs a further small, but crucial, modification.

The modified technique is described in detail in Algorithms 1, 2 and 3. The algorithms use two counters c_k and Σ_k. The counter c_k stores the number of vertices having a path of length at least k to t.

The counter Σ_k is the sum of the lengths of longest paths to t for those vertices whose longest path to t is of length less than k. Thus we define the following:

$$D(v) = \text{ Length of the longest path from } v \text{ to } t.$$
$$S_k = \{v \mid D(v) \geq k\}, \qquad c_k = |S_k|$$

$$\Sigma_k = \sum_{v \in V \setminus S_k} D(v), \qquad T = \sum_{v \in V} D(v)$$

While we describe and analyze the procedure in detail below, here is a quick overview for those familiar with the method from [9]. The crucial new parameter we need is T, the total length of all longest paths. At the outset, we nondeterministically guess a value M which is our estimate of T. At the end, when we have reached a value of k for which $c_k = 0$, we check whether M equals Σ_k. This allows us to make the procedure unambiguous. The additional condition that t is the unique sink allows us to initialise the counters: every vertex has a path, and hence a longest path, to t, and so $c_0 = n$.

The algorithms described here compute the longest path length when additonally s is the only source. After showing that these algorithms are correct, we discuss how to remove this restriction.

Algorithm 1 Main

Input: G, s, t
guess nondeterministically $M = \sum_{v \in V} D(v)$ with $n - 1 \leq M \leq n^2$
$c_0 \leftarrow n, \Sigma_0 \leftarrow 0, k \leftarrow 0$
while $c_k \neq 0$ **do**
  $k \leftarrow k + 1$
  Update (compute c_k and Σ_k)
end while
if $\Sigma_k \neq M$ **then**
  halt and **reject**
else
  output $D(s) = k - 1$, **accept** and **halt**
end if

It is clear that these procedures can be implemented in nondeterministic log-space. Claims 4.2, 4.3, 4.4 and 4.5, stated and proved below, show that these procedures unambiguously find the length of the longest path from s to t in a max-unique acyclic graph G where t is the only sink.

Claim 4.2. *If the guessed value of M is correct (i.e. $M = T$), then algorithm* Test*, given the correct values of c_k and Σ_k as input, reports a decision on exactly one path.*

Proof. The procedure Test, on each run R, guesses an x to t path R_x for each vertex x. Depending on its guess for $D(x) \geq k$, it adds the length of R_x to either sum or sum$'$. Finally these have to add up to M for Test to report a decision.

When $M = T$, M is indeed the sum of all $D(x)$. This can match sum + sum$'$ exactly when all the guessed paths R_x are longest. Since G is max-unique, this happens on exactly one run. □

Algorithm 2 Update: Compute c_k and Σ_k, given c_{k-1} and Σ_{k-1}

```
Input: G, s, t, k, c_{k-1}, Σ_{k-1}
c_k ← c_{k-1}, Σ_k ← Σ_{k-1}
for all v ∈ V do
  if Test(G, k-1, c_{k-1}, Σ_{k-1}, v)=true then
    if for all out-neighbours x of v,
      Test(G, k-1, c_{k-1}, Σ_{k-1}, x)=false then
      c_k ← c_k - 1, Σ_k ← Σ_k + k - 1
    end if
  end if
end for
```

Algorithm 3 Test: An unambiguous procedure to test if $D(v) \geq k$

```
Input: G, s, t, k, c_k, Σ_k, v
count = n, sum = 0, path_to_v=true, sum' = 0
for all x ∈ V do
  guess nondeterministically if D(x) ≥ k
  if guess is no then
    guess a path of length l < k from x to t.
    if this fails then
      reject and halt
    end if
    count ← count - 1
    sum ← sum + l
    if x = v then
      path_to_v =false
    end if
  else
    guess a path of length l' ≥ k from x to t
    if this fails then
      reject and halt
    end if
    sum' ← sum' + l'
  end if
end for
if count = c_k and sum = Σ_k and sum' + sum = M then
  return path_to_v
else
  reject and halt
end if
```

Claim 4.3. *For any guessed value of M, given the correct values of c_k and Σ_k as input, all paths of algorithm* Test *that do not lead to rejection always return the correct decision.*

Proof. As described in the preceding proof, each run of Test guesses a path R_x for each x. It may guess a path of length shorter than $D(x)$, but not longer. Since count is decremented only when it guesses that $D(x) < k$, and for other guesses some witnessing path of length at least k is found, at the end the value of count is at most as large as c_k.

Suppose on some run Test returns a decision. Then on this run count $= c_k$. Suppose further that the decision is wrong.
Case 1: $D(v) < k$, but Test reports that it is larger. This cannot happen, since Test has to find a witnessing path of length at least k.
Case 2: $D(v) \geq k$, but Test reports that it is smaller. Then this run of Test does not account for v in count. So at the end of the run, count $< c_k$, a contradiction. □

Claim 4.4. *If the queries $(D(v) \geq k)$ are answered correctly by* Test*, then given c_{k-1} and Σ_{k-1}, the values of c_k and Σ_k are updated correctly by algorithm* Update*.*

Proof. Update starts by assuming that $S_k = S_{k-1}$ and so $c_k = c_{k-1}$. Note that $S_k \subseteq S_{k-1}$, so Update only has to detect when to remove vertices from its current S_k.

For each v, Update checks whether $D(v) \geq k-1$ and $D(u) < k-1$ for all out-neighbours u of v. If this holds, then the longest path from v to t is of length exactly $k-1$ and $v \notin S_k$. Thus the procedure decrements c_k by 1 and increments Σ_k by $k-1$.

So if all the queries are answered correctly by Test, then what Update does is correct. □

Claim 4.5. *The algorithm* Main *is correct and unambiguous.*

Proof. Main starts with the correct values of c_0 and Σ_0. From Claims 4.3 and 4.4, the correctness of Main is immediate. In particular, the final value of Σ_k is always correct.

If $M = T$, then by Claim 4.2, procedure Test always returns a decision, unambiguously. Thus exactly one path of Main (amongst those where $M = T$ was guessed) leads to a decision, and this decision is correct.

If $M > T$, then no run of Test, at any stage k, can trace paths adding up to M. So Test, and hence Update, and Main have no accepting run.

If $M < T$, consider the runs on which Test and Update proceed to finally compute Σ_k. Since Main is correct, we know that $\Sigma_k = T$. Now the check $M = \Sigma_k$ fails and Main rejects and halts. □

A straightforward modification will handle the case when s is not the unique source. Just keep one additional special counter for the vertex s. Initialise this counter to 0. At each stage k, after c_k and Σ_k are computed, run Test to check if $D(s) \geq k$ and if so, set this counter to k. At the end of Main, report the value of this counter.

Alternate Proof of Theorem 1.1

The above algorithm can be used to give an alternate proof of Theorem 1.1. The idea is to (1) trim the graph using oracle queries to Reach(Planar) so that t is the only sink, (2) embed it in a grid with suitable

edge weights using the embedding algorithm of [1], and (3) use the weighting scheme of [4] so that the resulting planar DAG, say H, is min-unique. As further noted in [12], the shortest path in H corresponds to a shortest path in G. It is straightforward to see that H is also max-unique, with the longest path in H corresponding to some longest path in G. Thus Theorem 4.1 is applicable. Computing the longest path length in G from that in H is also straightforward.

5 Extensions of the Longest Path Algorithm

In this section we consider variations of our planar Long-Path algorithm: extending it to torodial graphs, finding a longest path, finding multiple longest paths, and counting the number of paths under some promise.

5.1 Long-Path in Toroidal Graphs

Recall Lemma 2.4: Reach(Torus) reduces to Reach(Planar) via a log-space many-one reduction. We observe here that the reduction can be modified to reduce Distance(Torus) and Long-Path(Torus) also to the corresponding problems on planar graphs.

Assume that the input graph G is not planar but can be embedded on a torus, and that the embedding on the torus is given. We use the construction of Lemma 2.4 to obtain a planar graph G' with the following properties: There are $l \in O(n)$ copies of G cut and stitched together in G', and hence there are l vertices $t_1, \ldots, t_l$ and one special vertex, say s_1, such that there is a path of length l from s to t in G if and only if there is a path of length l from s_1 to at least one of the t_is.

This gives a log-space truth-table reduction from toroidal to planar graphs, preserving path lengths. This can be converted to a many-one reduction using the technique from [1]: there, formula-truth-table reductions to Reach(Planar) are converted to many-one reductions to Reach(Planar), but it is easy to see that the conversion preserves path lengths as well. Hence we get the following corollary.

Corollary 5.1.

$$\text{Distance}(\textit{Torus}) \leq_m^{\log} \text{Distance}(\textit{Planar})$$

$$\text{Long-Path}(\textit{Toroidal DAG}) \leq_m^{\log} \text{Long-Path}(\textit{Planar DAG})$$

5.2 Finding a Longest Path

We show that for planar DAGs, the search version of Long-Path is also in UL $\cap$ co-UL.

Theorem 5.2. *A longest path between two designated nodes s and t in a planar DAG can be found in* $\mathsf{L}^{\mathsf{UL} \cap \text{co-UL}}$ *and hence in* UL $\cap$ co-UL.

Proof. The log-space machine computes the length of one of the longest paths from s to t by asking queries to the Long-Path oracle, with the values of k starting from n, till a 'yes' answer is obtained. Recall that Long-Path $= \{(G,s,t,k) \mid G$ has a simple path of length $\geq k$ from s to $t\}$. Let this length of the longest s to t path be l. Then find the neighbor v of s that has length $l-1$ path to t. Output v and continue, till finally t is output. □

5.3 Finding Multiple Longest Paths

We consider another variation of Long-Path. Given a planar DAG G, the algorithm in Section 5.2 produces a longest path ρ. Can we find the (length of the) longest path other than ρ? Note that this may well have the same length as ρ, because G itself need not be max-unique.

We proceed as follows: Let l be the length of ρ, one of the longest paths from s to t. For each edge e in ρ, let l_e denote the length of the longest path in $G \setminus \{e\}$. Then the length of the longest path other than ρ is the maximum of l_e over e in ρ, and such a path can be found by using the Algorithm of Section 5.2 on $G \setminus \{e\}$ for the appropriate edge e.

This can be generalised to finding the (length of the) k^{th} longest path, as long as k is a constant. Thus we have:

Theorem 5.3. *For each constant k, given a planar DAG G and vertices s, t in it, a list of k paths from s to t in G can be found in* UL $\cap$ co-UL *such that every s to t path not listed is no longer than any listed s to t path.*

Remark: The k longest paths may not be unique due to ties. The algorithm guarantees that any s to t path not listed by it is no longer than the shortest s to t path listed by it.

5.4 Computing the Number of Paths: A Promise Version

The problem of counting the number of paths between two designated nodes in a DAG is known to be complete for #L. Consider a restriction of this problem when the number of such paths is bounded by a polynomial. With this restriction it is natural to believe that the counting problem is easier, because even Reach, which is otherwise NL-complete, is in LogDCFL for such graphs [5].

In [3], an NL upper bound for the counting problem under this promise was obtained, further substantiating this belief. We consider an additional (easily checkable) restriction where the DAG not only has polynomially bounded number of s to t paths but also t is the unique sink in the graph. In such graphs, we show that the counting problem can be solved in LogDCFL, proving Theorem 1.2 as follows:

Proof of Theorem 1.2. Our procedure is a depth-first-search exploration of the graph. The algorithm explores the DAG starting from s. The number of s to t paths explored is stored in a variable count. It assumes an ordering on the labels of the vertices (lexicographical ordering will suffice) and an additional label 0 which is assumed to be the smallest in the ordering. It traverses the graph in a depth-first manner, putting visited vertices on the stack. The label of the vertex being visited in the current step is stored in the variable current. The traversal is started from s, taking the out-neighbour with the smallest label at each step. Whenever t is reached, count is incremented.

On reaching t, the algorithm backtracks by popping the stack, retrieving the vertex v visited just before t. The label of t is stored in a variable previous and the label of v is stored in current. If v has an out-neighbour u with a label larger than previous, the traversal is continued along the (v,u) edge, and the label of this out-neighbour is stored in the variable next. Otherwise, the backtracking process is continued by popping the stack again, storing the label of v in previous, and setting current to the newly popped label.

Thus, at any point of time, the stack contains nodes on the path from s to current (excluding current), and the vertex current is being explored in the forward or backward direction.

Note that for the algorithm to work in polynomial-time, it is essential that t is the unique sink in the graph. If there is another sink t', the algorithm will explore all s to t' paths as well, and this number may not be polynomially bounded.

Lemma 5.4. *Clearly, the algorithm correctly computes the number of s to t paths. Furthermore, for each s to t path ρ, each edge on ρ is traversed exactly once in the forward direction. During the backtracking phase, after reaching t along ρ, an edge on ρ is traversed at most once in the backward direction. Consequently, the algorithm can be implemented on a* DAuxPDA *running in polynomial time, if the number of s to t paths is bounded by a polynomial.*

This completes the proof of Theorem 1.2. □

5.5 Counting Paths in Planar DAGs: A Promise Version

We now consider applying Theorem 1.2 to planar DAGs.

The depth-first-search algorithm described in the proof of Theorem 1.2 may not be directly applicable, since the given DAG may have multiple sinks. These sinks can be removed in UL ∩ co-UL by queries to Reach, to get a planar DAG with t as the only sink. Then the depth-first-search from the proof of Theorem 1.2 can be used. Overall, the combined algorithm uses log-space in a nondeterministic unambiguous manner, and a stack in a deterministic manner, giving an algorithm that can be implemented on a UAuxPDA in polynomial time. Thus we get the first part of Theorem 1.3.

Our algorithm is similar to that given in [5] for testing reachability in the computation tree of an NL machine when each configuration of the machine is reachable by polynomially many paths.

Now consider the problem of computing the number of longest s to t paths when this number is bounded by a polynomial. Note that the total number of s to t paths need not be polynomially bounded, hence just removing the sinks other than t as above does not suffice.

Algorithm 4 describes a procedure that removes all the edges which do not appear on any longest path from s to t. It finds out the length of the longest s to t path in the planar DAG G in UL ∩ co-UL using Theorem 1.1. Then G is layered and edges that are not a part of any s to t longest paths are removed. These are exactly the edges that go across more than one layer. Thus, in the resulting DAG, all s to t paths are the longest s to t paths. Moreover, t is the only sink in this new DAG.

Now we can use the depth-first-search from the proof of Theorem 1.2 for counting s to t paths in this graph.

If instead of longest paths, we wish to count shortest paths, and this number is bounded by a polynomial, a similar approach can be used. The distance of a node v from s should be used as its layer number in Algorithm 4 instead of the length of the longest path from s to v. From Lemma 2.2 we know that distance can be computed in UL ∩ co-UL.

Combining the UL ∩ co-UL layering procedure of Algorithm 4 with the DAuxPDA procedure of depth-first-search described in the proof of Theorem 1.2, we get the second part of Theorem 1.3. As mentioned in Section 1, this number can also be determined in NL, and Theorem 1.3 gives a bound which is incomparable to this.

Algorithm 4 Procedure to delete all s to t paths shorter than the longest path.

Input: Planar DAG $G = (V, E)$, two designated vertices s, t
Output: Graph G' that has no paths other than s to t longest paths
for all $v \in V$ **do**
  $layer(v) \leftarrow$ length of s to v longest path
end for
for all edges (u, v) **do**
  if $layer(v) - layer(u) = 1$ **then**
    output (u, v)
  end if
end for

6 Discussion

This paper shows that in planar and toroidal DAGs, detecting the presence or absence of long paths can be done in unambiguous log-space. In particular, detecting long paths and detecting short paths are equivalent, modulo reachability in these graphs. This result has recently been generalised in [13], where the same upper bound for the longest path problem is obtained for directed acyclic graphs which exclude either $K_{3,3}$ or K_5 as minors. For all these graph classes, the best known lower bound for all three problems — reachability, shortest path, and longest path — is log-space hardness. There is thus a gap between the lower and upper bounds. It is open whether the upper bound of UL $\cap$ co-UL can actually be improved to log-space.

If the number of paths from s to t in a DAG is bounded by a polynomial in the graph size, then a result from [3] shows that this number can be computed in NL. In this paper, we have given a different bound for two restrictions:

1. If t is the unique sink in the DAG, then this number can be computed by a polynomial time DAuxPDA and hence in LogDCFL.

2. If the DAG is planar, then this number can be computed by a polynomial time UAuxPDA.

This is the first (natural, we believe) instance we know of a problem that is not known to be in log-space, but is accepted by both UAuxPDA[poly] and NL machines. It suggests that UL should be an upper bound for this problem; establishing this is an interesting question.

Acknowledgment

The authors are thankful to the anonymous referees for providing detailed comments which helped significantly in improving the presentation of the paper. The authors also thank the referee for pointing out that our original proof of Lemma 5.2 was needlessly complicated and giving the shorter construction that appears here.

References

[1] E. ALLENDER, S. DATTA, AND S. ROY: The directed planar reachability problem. In *Proc. 25th annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, LNCS, pp. 238–249, 2005. 2, 5, 11

[2] E. ALLENDER AND M. MAHAJAN: The complexity of planarity testing. *Information and Computation*, 189(1):117–134, 2004. 5

[3] E. ALLENDER, K. REINHARDT, AND S. ZHOU: Isolation, matching and counting uniform and nonuniform upper bounds. *Journal of Computer and System Sciences*, 59:164–181, 1999. 3, 12, 14

[4] C. BOURKE, R. TEWARI, AND N. V. VINODCHANDRAN: Directed planar reachability is in unambiguous log-space. *ACM Transactions on Computation Theory*, 1(1):1–17, 2009. 2, 4, 5, 7, 11

[5] G. BUNTROCK, B. JENNER, K.-J. LANGE, AND P. ROSSMANITH: Unambiguity and fewness for logarithmic space. In *Proc. 8th International Symposium on Fundamentals of Computation Theory (FCT)*, volume 529 of *LNCS*, pp. 168–179, 1991. 12, 13

[6] A. JAKOBY AND T. TANTAU: Computing shortest paths in series-parallel graphs in logarithmic space. In *Complexity of Boolean Functions*, 2006. Dagstuhl Seminar Proceedings. 15

[7] A. JAKOBY AND T. TANTAU: Logspace algorithms for computing shortest and longest paths in series-parallel graphs. In *Proc. 27th annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)* [6], pp. 216–227. Dagstuhl Seminar Proceedings. 2, 3, 4, 5

[8] O. REINGOLD: Undirected *st*-connectivity in logspace. In *Proc. 37th ACM Symposium on Theory of Computing (STOC)*, pp. 376–385, 2005. 2, 5

[9] K. REINHARDT AND E. ALLENDER: Making nondeterminism unambiguous. *SIAM J. Comp.*, 29:1118–1131, 2000. 2, 7, 8

[10] I. SUDBOROUGH: On the tape complexity of deterministic context-free language. *Journal of Association of Computing Machinery*, 25(3):405–414, 1978. 3

[11] T. TANTAU: Logspace optimization problems and their approximability properties. *Theory of Computing Systems*, 41(2):327–350, 2007. 2

[12] T. THIERAUF AND F. WAGNER: The isomorphism problem for planar 3-connected graphs is in unambiguous logspace. In *Proc. Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 08001 of *Dagstuhl Seminar Proceedings*, pp. 633–644, 2008. 2, 3, 4, 7, 11

[13] T. THIERAUF AND F. WAGNER: Reachability in $K_{3,3}$-free graphs and K_5-free graphs is in unambiguous log-space. *FCT*, 2009. To appear. 14

AUTHORS

Nutan Limaye
The Institute of Mathematical Sciences,
Chennai 600 113, India.
nutan [at] imsc [dot] res [dot] in

Meena Mahajan
The Institute of Mathematical Sciences,
Chennai 600 113, India.
nutan [at] imsc [dot] res [dot] in

Prajakta Nimbhorkar
The Institute of Mathematical Sciences,
Chennai 600 113, India.
nutan [at] imsc [dot] res [dot] in