Polynomial Min/Max-weighted Reachability is in Unambiguous Log-space

Anant Dhayal, Jayalal Sarma, and Saurabh Sawlani

Department of Computer Science and Engineering
Indian Institute of Technology Madras, Chennai, India

Abstract

For a graph $G(V, E)$ and a vertex $s \in V$, a weighting scheme ($w : E \to \mathbb{N}$) is called a *min-unique* (resp. *max-unique*) weighting scheme, if for any vertex v of the graph G, there is a unique path of minimum(resp. maximum) weight[1] from s to v. Instead, if the number of paths of minimum(resp. maximum) weight is bounded by n^c for some constant c, then the weighting scheme is called a *min-poly* (resp. *max-poly*) weighting scheme.

In this paper, we propose an unambiguous non-deterministic log-space (UL) algorithm for the problem of testing reachability in layered directed acyclic graphs (DAGs) augmented with a *min-poly* weighting scheme. This improves the result due to Reinhardt and Allender [11] where a UL algorithm was given for the case when the weighting scheme is *min-unique*.

Our main technique is a triple inductive counting, which generalizes the techniques of [7, 12] and [11], combined with a hashing technique due to [5] (also used in [6]). We combine this with a complementary unambiguous verification method, to give the desired UL algorithm.

At the other end of the spectrum, we propose a UL algorithm for testing reachability in layered DAGs augmented with *max-poly* weighting schemes. To achieve this, we first reduce reachability in DAGs to the longest path problem for DAGs with a unique source, such that the reduction also preserves the *max-poly* property of the graph. Using our techniques, we generalize the double inductive counting method in [8] where UL algorithms were given for the longest path problem on DAGs with a unique sink and augmented with a *max-unique* weighting scheme.

An important consequence of our results is that, to show NL = UL, it suffices to design log-space computable *min-poly* (or *max-poly*) weighting schemes for DAGs.

1998 ACM Subject Classification F.1.1 Models of Computation, F.1.2 Alternation and Nondeterminism

Keywords and phrases Reachability Problem, Space Complexity, Unambiguous Algorithms

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.597

1 Introduction

Reachability testing in graphs (Reach) is an important algorithmic problem that encapsulates central questions in space complexity. Given a graph $G(V, E)$ and two special vertices s and t, the problem asks to test if there is a path from s to t in the graph G. The problem admits a (deterministic) log-space algorithm for the case of trees and undirected graphs (by a breakthrough result due to Reingold[10]). The directed graph version of the problem captures the complexity class NL. Designing a log-space algorithm for the problem is equivalent to proving NL = L. (See [1] for a survey.) Even in the case when the graph is a layered DAG[2], the problem is known to be NL-complete.

[1] Weight of a path p is the sum of the weights of the edges appearing in p.
[2] A DAG is layered, if V can be partitioned as $V = V_1 \cup \ldots V_\ell$ s.t. edges go from V_i to V_{i+1} for some i.

34th Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2014).
Editors: Venkatesh Raman and S. P. Suresh; pp. 597–609
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

An important intermediate class of algorithms for reachability is when the non-determinism is unambiguous - when the algorithm accepts in at most one of the non-deterministic paths. The class of problems which can be solved by such restricted non-deterministic algorithms using only log-space is called Unambiguous Log-space (UL). Under a non-uniform polynomial-sized advice, the reachability problem is known to have a UL algorithm[11], thus showing NL/poly = UL/poly . Central to arriving at this complexity theoretic result was the following algorithmic result that Reinhardt and Allender [11] had established: testing reachability in a graph G augmented with a log-space computable weighting scheme that maps $w : E \to \mathbb{N}$ such that there is a unique minimum-weight path from s to any vertex v in the graph, can be done by a non-deterministic log-space algorithm unambiguously and hence is in the complexity class UL. (We call such weighting schemes as *min-unique* weighting schemes.) This also led to other important developments including an unambiguous log-space algorithm for directed planar reachability [4] - which was achieved by designing a log-space computable min-unique weighting scheme for reachability in grid-graphs (a special class of planar graphs for which reachability is as hard as planar DAG reachability[2]). An important open problem in this direction is to design a log-space min-unique weighting scheme for general graphs. The UL-computable version of this is also known to be equivalent to showing NL = UL.

Our Results: We make further progress on this algorithmic front by relaxing the restriction on the number of paths of minimum weight from one to polynomially many paths. We call a weighting scheme a *min-poly weighting scheme* if it results in at most polynomially many (in terms of $n = |V|$) paths of minimum weight from s to any vertex v in a graph $G(V, E)$.

► **Theorem 1.** *Testing reachability in layered DAGs, augmented with log-space computable min-poly weighting schemes, can be done by a non-deterministic log-space algorithm unambiguously and hence is in the complexity class* UL.

Our algorithms use a technique of triple inductive counting. The inductive counting method was originally discovered and employed as an algorithmic technique in [7] and [12] in order to design non-deterministic log-space algorithms for testing non-reachability in graphs. A double inductive version of this was used again by Reinhardt and Allender [11] for designing unambiguous non-deterministic algorithms for testing reachability in *min-unique* graphs. We use a triple inductive version of the inductive counting method, keeping track of one extra parameter (which is the sum of the number of minimum weight paths to each vertex). Along with a hashing technique (also used in [6]), this leads to a non-deterministic algorithm where each accepting configuration has at most one path leading to it on any input (the corresponding complexity class is known as FewUL). Finally, we convert this algorithm to a UL algorithm using an unambiguous complementary verification, thus completing the proof of the theorem.

A natural complementary question is if similar complexity bounds hold in the case of graphs with weighting assignments that result in unique maximum weight paths from s to any vertex v (such weighting schemes are called *max-unique* weighting schemes). In [8], the longest path problem on DAGs augmented with max-unique weighting assignments and having a unique sink t, was shown to be in UL. The corresponding weighting scheme with polynomially many paths of maximum weight will be called a *max-poly* weighting scheme. Using our triple inductive and complementary verification techniques, we adapt their algorithms to improve their results by relaxing the constraint on the weighting assignments - from max-unique to max-poly. We present our theorem in terms of the reachability problem, as we also show a reduction (Lemma 5) from the reachability problem to the longest path problem on single source DAGs, where the max-poly property of the graph is preserved.

▶ **Theorem 2.** *Testing reachability in layered DAGs augmented with log-space computable max-unique weighting schemes, can be done by a non-deterministic log-space algorithm unambiguously and hence is in the complexity class* UL*.*

▶ Remark. Observing that Theorem 1 and Theorem 2 hold even when the min-poly weighting scheme is UL-computable, and combining with the results of [9], it follows that: for any graph G there is a UL-computable min-poly weighting scheme if and only if there is a UL-computable min-unique weighting scheme. We also remark that, by a minor variant the proof technique in [9], we can show (the details are deferred to the appendix) that showing NL = UL is equivalent to designing UL-computable (min)max-unique weighting schemes which, thus, is equivalent to designing UL-computable (min)max-poly weighting schemes. However, we stress the importance of this relaxation of the constraints from uniqueness as this potentially can help designing weighting schemes for arbitrary layered DAGs.

Related Work: An important comparison of our results is with a complexity theoretic collapse result shown by [6]. FewL is the class of problems that has non-deterministic algorithms with only polynomially (in n) many accepting paths on any input of length n. Clearly, FewL contains all problems in UL - however, the converse is not known. In its algorithmic flavor, this question asks if reachability in a graph with at most polynomially many paths from s to t, can be done by a non-deterministic algorithm in log-space, producing at most one accepting path. ReachUL and ReachFewL are the corresponding complexity classes where the uniqueness and polynomially boundedness constraints are respectively applied for the number of paths from s to any other $v \in V$. Clearly, ReachUL is contained in ReachFewL and they were shown to be equal recently [6]. It is worthwhile noting that this establishes unambiguous log-space algorithms for reachability in graphs where there are only polynomially many paths from the start vertex to any vertex in the graph. The class of graphs that we discussed above (min/max-poly) also includes such graphs trivially. By assigning a weight of 1 to every edge in such a graph, there can only be polynomially many paths of minimum(or maximum) weight. Theorem 2, in particular, implies UL algorithms for reachability in graphs with max-unique weighting schemes where there need not exist a unique sink in the graph (and hence is a strengthening of the results in [8]).

2 Preliminaries

We assume basic familiarity with standard space complexity classes and reductions (see [3] for a standard textbook). The graphs considered in this paper are directed, acyclic and layered. Building on the terminology from the introduction, we say a DAG, $G(V, E)$, is min(max)-unique if it is augmented with a min(max)-unique weighting scheme. Similarly, a graph is said to be min(max)-poly if it is augmented with a min(max)-poly weighting scheme. A graph augmented with a weighting scheme $w : E \to \mathbb{N}$, can be converted to an un-weighted graph, by replacing each edge $e \in E$ with a path of length $w(e)$. Notice that this new graph also can be layered in log-space with edges allowed to jump forward, skipping layers arbitrarily. In particular, there is a log-space computable numbering for the vertices such that for each $(u, v) \in E$, u is given a smaller number as label than v. Additionally, in the algorithms presented in later sections, we also verify whether the input graph is min-poly and max-poly respectively.

In this new graph, we encode paths using numbers in the following way. Consider a path of length $k-1$, $p : (x_1, x_2, \ldots, x_k)$ where the x_is are the distinct integers representing the vertices in the path. Let us represent this path p with the integer $w_p := 2^{x_1} + 2^{x_2} + \ldots + 2^{x_k}$.

In other words, each path is represented by an n-bit integer, where the ith bit is 1 if and only if vertex i is in the path. Observe that, since the graph is a layered DAG, edges are always directed from a vertex of lower index to a vertex of higher index. Thus, a set of vertices is enough to represent a path, irrespective of their order. Hence, each path p can be represented by the unique number w_p. In the case of min(max)-poly graphs, the algorithm cannot store all $s \rightsquigarrow v$ paths to check whether they are different from each other or not. Hence, we use the following hashing technique. For $v \in V$, let P_v be a set of min(max)-length $s \rightsquigarrow v$ paths. P_s, by convention, contains one $s \rightsquigarrow s$ path of length 0. Let $S_v = \{w_p \mid p \in P_v\}$. Clearly, $|S_v| = |P_v| \leq n^c$.

Hashing the weights of paths: For any path $p : s \rightsquigarrow v$, we define $\phi_m(p) := (\sum_{u \in p} 2^u) \mod m$. We say that any integer m is *good* for a vertex $v \in V$, if no two $s \rightsquigarrow v$ paths p_1 and p_2 satisfy $\phi_m(p_1) = \phi_m(p_2)$. We say that m is *good* for a graph G, if it is *good* for all $v \in V$. The following proposition ensures that there is always a polynomial sized good m.

▶ **Proposition 1.** *[5] For every constant c there is a constant c' so that for every set S of n-bit integers with $|S| \leq n^c$ there is a $c' \log n$-bit prime number m so that for all $x, y \in S$, $x \neq y \implies x \not\equiv y \mod m$.*

Guessing paths in lexicographic order: Our algorithms often require guessing several paths to a vertex v in sequence and checking whether the guessed paths are in lexicographic order w.r.t ϕ_m. Here, we outline a method of doing this in log-space.

Keep a counter c of $\log \ell$ bits to keep track of how far we have traversed along a path. Initialize this to 0. Keep $\log n$ bits to store the current vertex ρ of the current path π. Let π' be the previous path. Keep two variables, δ_π and $\delta_{\pi'}$, of $\log m$ bits each. to store the intermediate value of $\phi_m(\pi)$ and previously calculated final value of $\phi_m(\pi')$ respectively. Repeat the following two steps until c reaches ℓ. (1) $\delta_\pi = (\delta_\pi + 2^\rho) \mod m$. (2) Increment c and choose one of ρ's neighbour vertices non-deterministically and replace ρ by this neighbour.

Setting δ_π to $\delta_{\pi'}$ and setting δ_π, ρ and c to 0, repeat the steps in the previous paragraph till we have guessed all the q paths. Each time, before updating $\delta_{\pi'}$, check if δ_π is strictly less than $\delta_{\pi'}$. If not, reject there itself.

Now we fix some notation. For any vertex $v \in V$, we denote by $d(v)$ (and $D(v)$), the minimum-distance (and maximum distance) of v from s. For any vertex $v \in V$, $p(v)$ (and $P(v)$) is the number of minimum-length (and maximum-length) $s \rightsquigarrow v$ paths.

3 FewUL Algorithm for Reach in min-poly layered DAGs

The UL algorithm given by Reinhardt and Allender [11] solves Reach for min-unique graphs. In this section, we introduce a modification of their algorithm to work for min-poly graphs. To handle polynomially many minimum-length paths, we introduce a new inductive parameter p_k which stores the sum of the number of minimum length paths from s to every vertex v with $d(v) \leq k$. To inductively compute this new parameter for each k, we will use the method of guessing paths p in lexicographic order with respect to their hashed values (ϕ_m) assuming that the guess of m is *good*.

However, we are still faced with the problem of obtaining a *good* m. In the following set of routines, we will guess the value of m and use it while simultaneously detecting if it is not *good*. Note that this routine will not be unambiguous any more, because there could be several choices of m which are *good* for the given graph. However, each choice of m will lead

Algorithm MAIN-MIN-FEWUL: Main FewUL routine to check reachability on min-poly graphs.

Input: (G, s, t)
Non-deterministically guess $2 \leq m < n^{c'}$
$k := 1$
$c_0 := 1; \Sigma_0 := 0; p_0 := 1$
$(c_1, \Sigma_1, p_1) = $ UPDATE-MIN$(G, s, 0, c_0, \Sigma_0, p_0, m)$
while $k < n - 1$ and $(c_{k-1}, \Sigma_{k-1}, p_{k-1}) \neq (c_k, \Sigma_k, p_k)$ **do**
  $(c_{k+1}, \Sigma_{k+1}, p_{k+1}) = $ UPDATE-MIN$(G, s, k, c_k, \Sigma_k, p_k, m)$
  $k := k + 1$
end while
if TEST-MIN$(G, s, t, k, c_k, \Sigma_k, p_k, m) > 0$ **then**
  Go to state ACCEPT-m
else
  REJECT
end if

Algorithm UPDATE-MIN: Deterministic (barring TEST-MIN calls) routine computing c_{k+1}, Σ_{k+1} and p_{k+1}.

Input: $(G, s, k, c_k, \Sigma_k, p_k, m)$
Output: $c_{k+1}, \Sigma_{k+1}, p_{k+1}$
$c_{k+1} := c_k$; $\Sigma_{k+1} := \Sigma_k$; $p_{k+1} := p_k$;
$num := 0$;
for $v \in V$ **do**
  if TEST-MIN$(G, s, v, k, c_k, \Sigma_k, p_k, m) = 0$ **then**
    for x such that $(x, v) \in E$ **do**
      $num := num + $ TEST-MIN$(G, s, x, k, c_k, \Sigma_k, p_k, m)$;
      if $num > n^c$ **then**
        REJECT
      end if
    end for
    if $num > 0$ **then**
      $c_{k+1} := c_{k+1} + 1; \Sigma_{k+1} := \Sigma_{k+1} + k + 1; p_{k+1} := p_{k+1} + num$;
    end if
  end if
end for

to exactly one accept state. Hence, we can label these accept states with their respective choices of m, thus making it a FewUL routine.

Algorithm: Here we give the outline of the FewUL algorithm for $L = \{\ (G(V, E), s, t) \mid \exists s \rightsquigarrow t$ path and $\forall v \in V, p(v) \leq n^c\}$, where the value of c is known. We fix some basic notations. $c_k = |\{v \in V : d(v) \leq k\}|$, $\Sigma_k = \sum_{d(v) \leq k} d(v)$. The extra parameter p_k is equal to $\sum_{d(v) \leq k} p(v)$. First, building on the central idea of [11], we design an unambiguous log-space routine (TEST-MIN) to determine if $d(v) \leq k$ and return $p(v)$ (in at most one non-deterministic path), assuming the correct values of c_k, Σ_k, p_k and m. The modification is that, for each vertex $x \in V$ the algorithm will guess the number of paths (q - in the algorithm $q = 0$ is interpreted as "guessing that $d(v) > k$") from s to x, their length ℓ, and the paths themselves in strictly decreasing order with respect to ϕ_m. Using this subroutine, we then compute inductively, the values of c_{k+1}, Σ_{k+1} and p_{k+1}. We will inductively compute $p(v)$ and check if it is greater than the polynomial bound n^c. If $p(v)$ exceeds this number, the subroutine rejects as the underlying graph is not min-poly. This is described in the pseudocode UPDATE-MIN. The main FewUL algorithm will inductively compute c_k, Σ_k and p_k starting from $k = 1$ to $n - 1$.

Algorithm TEST-MIN: Unambiguous Log-space routine to return $p(v)$ if $d(v) \leq k$ (returns 0 if $d(v) > k$, rejects if $p(v) \geq n^c$), given correct values of c_k, Σ_k, p_k and a *good* m.

Input: $(G, s, v, k, c_k, \Sigma_k, p_k, m)$
$count := 0; sum := 0; paths := 0; paths.to.v := 0;$
for $x \in V$ **do**
  Nondeterministically guess $0 \leq q \leq n^c$
  if $q \neq 0$ **then**
    Nondeterministically guess $0 \leq \ell \leq k$
    Nondeterministically guess q paths $p_1, p_2, \ldots p_q$ of length exactly ℓ each from s to x.
    if ($(\exists i < j, \phi_m(p_i) \leq \phi_m(p_j))$ OR (paths are not valid)) **then**
      REJECT
    end if
    $count := count + 1; sum := sum + \ell; paths := paths + q;$
    if $x = v$ **then**
      $paths.to.v := q;$
    end if
  end if
end for
if $count = c_k$, $sum = \Sigma_k$ and $paths = p_k$ **then**
  Return the value of $paths.to.v$
else
  REJECT
end if

▶ **Claim 1.** *If m is* good*, given the correct values of c_k, Σ_k and p_k, the algorithm* TEST-MIN *has exactly one non-rejecting path, and it returns the correct value of $p(v)$.*

Proof. We argue that, since m is *good*, there is a unique way to guess the $d(v)$ and $p(v)$ ($\forall v \in V$), to satisfy $count = c_k$, $sum = \Sigma_k$ and $paths = p_k$. We analyze this by cases.

If the algorithm, in a non-deterministic choice, guesses $q > 0$ for some vertex v (i.e. $d(v) \leq k$) for which $d(v) > k$, then it will not be able to guess any path of length $\leq k$, and hence will end up rejecting in that non-deterministic choice. If it guesses $q = 0$ for some vertex v (i.e. $d(v) > k$) for which $d(v) \leq k$, it will not increment *count*. But then, to compensate this loss, for *count* to reach c_k, the algorithm, in this non-deterministic choice, will have to guess $q > 0$ for some vertex u for which $d(u) > k$, and thereby will reject.

If the algorithm, in a non-deterministic choice, guesses $\ell < d(v)$ ($q > p(v)$) for any v, then it will not be able to find - a path of such length (that many paths) and hence will end up rejecting in that non-deterministic choice. If it guesses $\ell > d(v)$ ($q < p(v)$), then to compensate, it will have to guess $\ell < d(u)$ ($q > p(u)$) for some other vertex u, and hence will reject in that non-deterministic path.

Hence, only the path in which, for all vertices, q and ℓ are guessed correctly and all q paths of length ℓ are guessed in lexicographical order w.r.t. ϕ_m, will be a non-reject path and will return the value of $p(v)$ correctly. ◀

▶ **Claim 2.** *If the algorithm* TEST-MIN *works correctly for parameter k, then given the correct values of c_k, Σ_k and p_k, the algorithm* UPDATE-MIN *computes the correct values of c_{k+1}, Σ_{k+1} and p_{k+1}.*

Proof. The algorithm first assigns $c_{k+1} := c_k, \Sigma_{k+1} := \Sigma_k$ and $p_{k+1} := p_k$. Now, to update these values we need the exact set of vertices with $d(v) = k + 1$. The algorithm, for each v, checks if $d(v) > k$ and for each of its neighbours x, checks if $d(x) \leq k$. For the neighbours passing this test, we know that $d(x) = k$. If any of the neighbours passes the test ($num > 0$ in line 13), $d(v) = k + 1$. Hence, c_{k+1} is incremented by 1, Σ_{k+1} is incremented by $k + 1$, and p_{k+1} is incremented by $\sum_{(x,v)\in E, d(x)=k} p(x)$ (which is stored in num after loop 7-12). Hence all the three parameters get updated correctly and hence the proof. ◀

▶ **Observation 1.** *Observe that, since we begin with the correct values of c_0, Σ_0 and p_0, by induction, Claims 1 and 2 imply that the values of c_k, Σ_k and p_k calculated at any time in the algorithm are always correct.*

▶ **Claim 3.** *If m is* good*, the algorithm* MAIN-MIN-FEWUL *has at most one path to state ACCEPT-m.*

Proof. Using Observation 1 and Claim 1, we know that there is exactly one non-rejecting path in each call to TEST-MIN. Thus, there is exactly one non-rejecting path in each call to UPDATE-MIN, as UPDATE-MIN is deterministic barring the calls to TEST-MIN. Similarly, there is exactly one non-rejecting path in MAIN-MIN-FEWUL, as MAIN-MIN-FEWUL - for a particular choice of m - is deterministic barring the calls to UPDATE-MIN. If t is indeed reachable from s, this non-rejecting path goes to ACCEPT-m, as m is guessed initially and is not changed thereafter. ◀

▶ **Claim 4.** *If m is not* good*, given the correct values of c_k, Σ_k and p_k, the algorithm* TEST-MIN *(and hence both* UPDATE-MIN *and* MAIN-MIN-FEWUL*) always rejects.*

Proof. If m is not *good*, then there exists a vertex v such that there exist at least two $s \rightsquigarrow v$ paths p_1 and p_2 for which $\phi_m(p_1) = \phi_m(p_2)$. So, if we guess $q = p(v)$, then the paths cannot be in strictly decreasing order w.r.t. ϕ_m and the algorithm will reject. If we guess $q > p(v)$, then the algorithm will fail to find q paths and reject. If we guess $q < p(v)$, then *paths* will never be equal to p_k, as the q for some other vertex u will then need to be greater than $p(u)$ (for *paths* to become equal to p_k), which is not possible. ◀

▶ **Theorem 3.** *The algorithm* MAIN-MIN-FEWUL *is correct and* FewUL.

Proof. If the value of m guessed is not *good*, then the algorithm MAIN-MIN-FEWUL always rejects (by Claim 4 and Observation 1), and if it is *good*, there is at most one path which reaches ACCEPT-m (Claim 3). As there are polynomially many possible values of m, MAIN-MIN-FEWUL is in FewUL. After covering all the reachable vertices, the *while* loop (line 6-9) in MAIN-MIN-FEWUL terminates with correct values of c_k, Σ_k and p_k (Observation 1) and before reaching ACCEPT-m we do a final check to see whether or not vertex t has been covered. As this case occurs only when m is *good* (Claims 3 and 4), the correct values of $p(v)$ will be returned (Claim 1) and thus the final decision will be correct. ◀

4 UL Algorithm for Reach in min-poly layered DAGs

The algorithm presented in the previous section is not unambiguous because there can be more than one *good* m. To address this, we modify the MAIN-MIN-FEWUL routine in such a way that we always use the least *good* m (let us call this integer f). The TEST-MIN and UPDATE-MIN routines are already unambiguous and need no change.

The idea is to non-deterministically guess f, and to verify that f is the smallest *good* integer for the graph G. This is done by running an unambiguous routine which checks all integers $m < f$ and, for each value, verifies that it is not *good* and proceeds to the next value. Finally it reaches f, and accepts if and only if it is *good* and there is a path from s to t.

Algorithm UPDATE-FAULT-MIN: UL routine to verify our choice of f.

1: **Input**: (G, s, m)
2: non-deterministically guess $1 < k_1 < n$
3: $c_0 := 1; \Sigma_0 := 0; p_0 := 1; k := 1$
4: **while** $k < k_1$ **do**
5: $\quad (c_k, \Sigma_k, p_k) = \text{UPDATE-MIN}(G, s, k, c_{k-1}, \Sigma_{k-1}, p_{k-1}, m)$
6: $\quad k := k + 1$
7: **end while**
8: $match\_found := false$
9: **for** $v \in V$ **do**
10: $\quad$ **if** $\text{TEST-MIN}(G, s, v, k-1, c_{k-1}, \Sigma_{k-1}, p_{k-1}, m) = 0$ **then**
11: $\quad\quad valid := false$
12: $\quad\quad$ **for** x such that $(x, v) \in E$ **do**
13: $\quad\quad\quad$ **if** $\text{TEST-MIN}(G, s, x, k-1, c_{k-1}, \Sigma_{k-1}, p_{k-1}, m) > 0$ **then**
14: $\quad\quad\quad\quad valid := true$
15: $\quad\quad\quad$ **end if**
16: $\quad\quad$ **end for**
17: $\quad\quad$ **if** $valid$ **then**
18: $\quad\quad\quad$ **for** $(a, b)|(a, v)$ and $(b, v) \in E$ **do**
19: $\quad\quad\quad\quad \alpha := \text{TEST-MIN}(G, s, a, k-1, c_{k-1}, \Sigma_{k-1}, p_{k-1}, m)$
20: $\quad\quad\quad\quad \beta := \text{TEST-MIN}(G, s, b, k-1, c_{k-1}, \Sigma_{k-1}, p_{k-1}, m)$
21: $\quad\quad\quad\quad$ **if** $(\alpha > 0) \wedge (\beta > 0) \wedge (\text{FIND-MATCH}(G, s, k, a, b, \alpha, \beta, m) = true)$ **then**
22: $\quad\quad\quad\quad\quad$ RETURN
23: $\quad\quad\quad\quad$ **end if**
24: $\quad\quad\quad$ **end for**
25: $\quad\quad$ **end if**
26: $\quad$ **end if**
27: **end for**
28: REJECT

If an integer $m < f$ is not *good*, there must be a least integer $k_1(m)$ (from s) such that there exists a vertex v for which $d(v) = k_1(m)$ and for which m is not *good*. It suffices to find this vertex in order to certify that m is not good. For any such vertex v, there must exist $a, b \in V$ such that a, b are in-neighbours of v at distance $k_1(m) - 1$ from s and there must be two paths, p_a through a and p_b through b such that $\phi_m(p_a) = \phi_m(p_b)$. Indeed, $a \neq b$, since otherwise it contradicts the choice of $k_1(m)$. This is done by an unambiguous non-deterministic algorithm FIND-MATCH$((G, s, k, a, b, \alpha, \beta, m)$, which guesses α(respectively β) number of $s \rightsquigarrow a$ ($s \rightsquigarrow b$) paths and pairwise checks for collision with respect to ϕ_m between $s \rightsquigarrow a$ and $s \rightsquigarrow b$ paths. This is used as a subroutine in UPDATE-FAULT-MIN.

▶ **Theorem 4.** *The algorithm* MAIN-MIN-UL *is correct and unambiguous* log-*space.*

Proof. Let f' be the smallest *good* value for graph G. We first argue that, if m is not *good* then there exists exactly one non-reject path in UPDATE-FAULT-MIN. We do this by considering the following cases : If $k_1 > k_1(m)$, then in the while loop (lines 4-7), when $k = k_1(m)$, UPDATE-MIN will find two paths p_1 and p_2 satisfying $\phi_m(p_1) = \phi_m(p_2)$ and will reject. If $k_1 < k_1(m)$ then FIND-MATCH will never find two paths p_1 and p_2 satisfying $\phi_m(p_1) = \phi_m(p_2)$. So, it will always return $false$ and thus, UPDATE-FAULT-MIN will reject at line 28. If $k_1 = k_1(m)$: let u be the lexicographically first vertex such that there exist two $s \rightsquigarrow u$ paths p_1 and p_2 satisfying $\phi_m(p_1) = \phi_m(p_2)$. Hence, in line 22, when $v = u$, the algorithm will return, and this is the only non-reject path.

Now we argue that, if m is *good* then UPDATE-FAULT-MIN rejects. Notice that, irrespective of the value of k_1 guessed, FIND-MATCH will not be able to find two paths p_1 and p_2 such that $\phi_m(p_1) = \phi_m(p_2)$ as m is *good*. Hence, in line 22, UPDATE-FAULT-MIN algorithm will never return and thus will reject in line 28.

Now we are ready to argue unambiguity of MAIN-MIN-UL. More specifically, we argue that if $f = f'$, MAIN-MIN-UL accepts in at most one path, and if $f \neq f'$, MAIN-MIN-UL rejects. Consider the case $f = f'$. In each iteration of the first while loop (lines 4-7) in MAIN-MIN-UL, m is not *good* and thus by the above argument, the while loop terminates in exactly one path. The rest of the algorithm (lines 8-19) is identical to MAIN-MIN-FEWUL. So, by Claim 3 there is at most one accept path. Note that here, unlike in MAIN-MIN-FEWUL, we will reach a unique accept state corresponding to $m = f = f'$.

Now consider $f \neq f'$. If $f < f'$, then at line 7, when the first while loop terminates, $m = f < f'$, and UPDATE-MIN with f as parameter will reject because of Claim 4 and Observation 1. If $f > f'$, then when in first while loop $m = f'$ (and hence m is good), UPDATE-FAULT-MIN will reject (as shown above).

Now we argue correctness. As argued, line 15 in MAIN-MIN-UL will be reached only when $f = f'$. At this point, c_k, Σ_k, p_k are calculated correctly, as Observation 1 still holds. Thus, by Claim 1, TEST-MIN outputs the correct value of $p(t)$ as $m = f'$ is *good* and thus the final result is correct. ◀

5 Reach in max-poly layered DAGs

In order to arrive at the algorithm for REACH in max-poly graphs, we solve a harder problem on a more specific class of graphs. This is a variant of the LONG-PATH problem (Given (G, s, t, j) where s and t are vertices in the graph G, and j is an integer - the LONG-PATH problem asks to check if there is a path from s to t of length at least j) where the graph G has a unique source s. We first give the reduction from REACH to this special case of LONG-PATH.

▶ **Lemma 5.** *There is a function f, computable in log-space, that transforms an instance $(G(V,E), s, t)$ of* REACH *to an instance $(G'(V',E'), s', t, 2n+1)$ of* LONG-PATH, *where $n = |V|$, such that t is reachable from s in G if and only if there exists a path of length at least $2n+1$ from s' to t in G'. In addition, if G is max-unique (max-poly), then G' is max-unique (max-poly).*

Proof. As mentioned in the preliminaries, without loss of generality, we can assume that the vertices of the graph $G(V,E)$ are numbered such that, edges always go from a lower numbered vertex to a higher numbered vertex. Let $V = \{v_1, v_2, \ldots, v_n\}$ be this numbering. We will construct $G'(V',E')$ as follows: In addition to the edges among the vertices in V, we add a new source vertex s' and add edges from s' to all other vertices in V. We assign weights to the newly added edges (which we later remove by replacing the edges with paths of length equal to the weight of the edge). The weight of the edge $(s', s) = 2n$ and for vertices $v_i \neq s$, weight of (s', v_i) is $2i$. Note that G' has exactly one source vertex s' and hence is a valid input for our algorithm to solve LONG-PATH.

Now we argue the that if G had a unique path (polynomially many paths) of maximum length from s to any vertex v, then so will be the case with G'. This condition is easily seen for $v \notin V$. For a vertex $v_i \in V$, we claim that among all the paths not going through s, there is exactly one path of maximum length and this is the path corresponding to the edge (s', v_i) of length $2i$. If not, choose a longest path (say p) which is not corresponding to the edge (s', v_i). Let v_j $(j < i)$ be the first vertex in p from V. Clearly, p must use the path

Algorithm MAIN-MIN-UL: Main UL routine to check reachability.

Input: (G, s, t)
Non-deterministically guess $2 \leq f < n^{c'}$
$m := 2$
while $m < f$ **do**
    UPDATE-FAULT-MIN(G, s, m)
    $m := m + 1$
end while
$k := 1$
$c_0 := 1; \Sigma_0 := 0; p_0 := 1$
$(c_1, \Sigma_1, p_1) =$ UPDATE-MIN$(G, s, 0, c_0, \Sigma_0, p_0, m)$
while $k < n - 1$ and $(c_{k-1}, \Sigma_{k-1}, p_{k-1}) \neq (c_k, \Sigma_k, p_k)$ **do**
    $(c_{k+1}, \Sigma_{k+1}, p_{k+1}) :=$ UPDATE-MIN$(G, s, k, c_k, \Sigma_k, p_k, m)$
    $k := k + 1$
end while
if TEST-MIN$(G, s, t, k, c_k, \Sigma_k, p_k, m) > 0$ **then**
    ACCEPT
else
    REJECT
end if

Algorithm (FIND-MATCH): UL routine to find paths with matching ϕ_m values.

Input: $(G, s, k, a, b, \alpha, \beta, m)$
for $i = 1$ to α **do**
    Guess a path π of length $k - 1$ from s to a
    if $(i \geq 2) \wedge (\phi_m(\pi) \geq X)$ **then**
        REJECT
    end if
    $X := \phi_m(\pi)$
    for $j = 1$ to β **do**
        Guess a path π' of length $k - 1$ from s to b
        if $(j \geq 2) \wedge (\phi_m(\pi') \geq Y)$ **then**
            REJECT
        end if
        $Y := \phi_m(\pi)$
        if $X = Y$ **then**
            Return *true*
        end if
    end for
end for
Return $false$

corresponding to the weighted edge (s', v_j). Hence, the length of the path p can at most be $2j + (i - j) = i + j < 2i$. This contradicts the choice of p.

Thus, for a vertex $v_i \in V$ that is not reachable from s, the maximum length path in G' is unique. For a vertex $v_i \in V$ that is reachable from s, the maximum length path not through s is of weight exactly $2i$, but the paths from s' to v_i through s are of length at least $2n + 1 > 2i$. Additionally, we can see that, if there were ℓ paths of maximum length from s to any vertex v_i in G, then the number of maximum length paths from s' to v_i is also ℓ.

We now argue correctness of our reduction. Suppose that t is not reachable from s in G. In this case, none of the paths from s' to t will pass through s. Hence, using the above argument, we know that the length of any path from s' to t cannot be greater than $2n$. On the other hand, if t is reachable from s in G (say by path p), then the path (s', s) concatenated with p is a path of length $\geq 2n + 1$ from s' to t. ◀

Now we turn to this special case of the Long-path problem. As mentioned in the introduction, Long-Path for max-unique graphs with a unique source has been studied by [8]. The UL algorithm in [8] is for Long-Path on max-unique graphs having a single sink t. In our version of Long-Path, we will consider paths from s (as opposed to paths to t in [8]) and hence we will consider only graphs with a unique source s. We will extend their algorithm to max-poly graphs, by first giving a FewUL algorithm, and then converting it to a UL algorithm using a strategy similar to the min-poly Reach algorithm in Section 4 .

5.1 FewUL Algorithm for Reach in max-poly Layered DAGs

In a way similar to our adaptation of the algorithm for min-unique graphs of [11] to work with min-poly layered DAGs, we adapt the algorithm proposed in [8] for max-unique graphs (with a unique sink) to the case for max-poly graphs with a unique source. Along with the reduction we mentioned above from Reach to Long-path in such graphs (preserving the max-unique or max-poly property), this gives an algorithm for reachability testing in such graphs. We build the intuition through an example setting where the idea used in the min-poly algorithm (Test-min) fails. Suppose we have the correct values of c_k, Σ_k and p_k. Even then, suppose for a vertex v, we guess $D(v) < k$ whereas actually $D(v) \geq k$. The algorithm, in this non-deterministic choice can still compensate and make it to the original summation by guessing for another u that $D(u) \geq k$ where actually $D(u) < k$. This is possible because the algorithm does not verify guesses of the kind $D(u) \geq k$ (that is, $q = 0$). In [8], this problem is addressed by introducing a new parameter $M = \sum_{v \in V} D(v)$. The value of M is also non-deterministically guessed, which if guessed correctly, will facilitate verification of the guess $D(u) \geq k$.

In a similar way, corresponding to the inductively computed parameter p_k, we introduce $P = \sum_{v \in V} P(v)$. In what follows, we will outline a FewUL algorithm with this new parameter and give a proof sketch.

Overview of the Algorithm: We introduce notation required for our exposition. We reuse c_k to denote the number of vertices $v \in V$ for which $D(v) \geq k$. $\Sigma_k = \sum_{v:D(v)<k} D(v)$, $p_k = \sum_{v:D(v)<k} P(v)$. Notice that $c_0 = n$.

We first introduce Test-max$(G, s, v, c_k, \Sigma_k, p_k, m)$, which given the correct values of c_k, Σ_k and p_k, tests unambiguously whether $D(v) \geq k$ and outputs $(D(v), P(v))$ if $D(v) < k$ or outputs $(0, 0)$ if $D(v) \geq k$. We then initialize *count* $= n$ and $\sum$ and *paths* to 0. For each vertex x, we guess if $D(x) \geq k$. If we guess NO, then the algorithm runs on similar lines as Test-min, where we guess the maximum path length, the number of paths of that length from s to x, and the paths themselves in strictly decreasing order with respect to ϕ_m. We decrement *count*, and increment *sum* and *paths* appropriately. If we guess YES, then we perform a similar check by guessing the maximum path length, the number of paths of that length (now at least k) from s to x, and the paths themselves in strictly decreasing order with respect to ϕ_m. However this time, we increment sum' and $paths'$ (instead of *sum* and *path*) respectively. Once we run through all the vertices, we verify the guesses of the kind $D(v) < k$ by matching *count* with c_k, *sum* with Σ_k and *paths* with p_k. In addition, we verify the guesses of the kind $D(v) \geq k$ by matching $sum + sum' = M$ and $paths + paths' = P$.

The inductive computation of c_{k+1}, Σ_{k+1} and p_{k+1} from c_k, Σ_k, and p_k is done by the routine UPDATE-MAX (along the lines of UPDATE-MIN). For each vertex with $D(v) = k$, it decrements c_k by 1, Σ_k by k and p_k by $\sum_{(x,v)\in E, D(x)=k-1} P(v)$ to compute c_{k+1}, Σ_{k+1} and p_{k+1} respectively. In order to find vertices with $D(v) = k$, this routine, for each node v, verifies if $D(v) = k$ by invoking the routine TEST-MAX on v and its in-neighbours.

The main reachability test algorithm, given (G', s', t') as the input, constructs, in log-space, the instance (G, s, t, j) of the special case of LONG-PATH problem. It runs the remaining algorithm with this new graph. The algorithm guesses m, M and P, and inductively computes c_k, Σ_k and p_k until they stabilize (which happens only at $c_k = 0$, since G is a single source graph). Finally, to answer the original reachability problem, it suffices to test if $D(t) \geq j$. Since c_k, Σ_k and p_k are available, this can be decided using the TEST-MAX algorithm.

Proof (Sketch) of Correctness and Unambiguity. Let T and S be the correct values of M and P respectively. We claim that, irrespective of the guessed values of M and P, if the input values c_k, Σ_k and p_k are correct, then all non-reject paths of TEST-MAX return the correct values of $P(v)$ and $D(v)$ for v if $D(v) < k$. (For other vertices it returns $(0, 0)$). If, in addition, M and P were correct and m is 'good', then there is exactly one non-reject path in TEST-MAX and hence in MAIN-MAX-FEWUL.

It can be seen that if either M or P are guessed larger than the correct value, then $sum + sum' = M$ ($paths + paths' = P$) will never be true. If at least one of them is guessed lesser than their correct value, then for the integer k such that $D(v) < k$ for all vertices $v \in V$, we will obtain $sum = \Sigma_k$ ($paths = p_k$) and $sum' = 0$ ($paths' = 0$). However, due to the correctness of the value of Σ_k (p_k), $\Sigma_k = T$ ($p_k = S$), the check $sum + sum' = M$ ($paths + paths' = P$) will fail. Hence the algorithm is correct and is FewUL. ◀

5.2 UL Algorithm for Reach in max-poly Layered DAGs

The FewUL algorithm presented in Section 3 is not unambiguous because there could be several choices of m which are *good* for G. However, there is a conceptual difficulty in guessing the lexicographically first *good* m (which we call f). Unlike in the case of min-poly graphs, here, for the each vertex $v \in V$, the guesses $D(v) \geq k$ also require verification. Suppose, $m < f$ is not *good* - i.e., there are two paths p_1 and p_2 to a vertex u with $D(u) = k_1 - 1$ (let k_1 be the least such integer) such that $\phi_m(p_1) = \phi_m(p_2)$. For any vertex x with $D(x) \geq k_1 - 1$, the value of m is not guaranteed to be *good*. Hence there could be several computation paths on which the algorithm rejects and there is no unambiguous way to skip to $m + 1$.

We outline an idea to fix this issue, which leads to the design of a UL algorithm. We defer the details to the full version of this paper. As in the case of min-poly graphs, for each m, the algorithm UPDATE-FAULT-MAX guesses the least integer k_1 such that there is a u with $D(u) = k_1 - 1$, and two $s \rightsquigarrow u$ paths p_1 and p_2 with $\phi_m(p_1) = \phi_m(p_2)$. Prior to this point, we run UPDATE-MAX with ϕ_m and ϕ_f both being calculated for the paths - and ϕ_m being computed only for the paths to vertices with $D(v) < k$. We verify whether there are two such paths with the same end point v with $D(v) = k_1 - 1$ using FIND-MATCH. In this modified algorithm, the guesses $D(v) > k$ can be verified by using ϕ_f values, since we are assuming that f is *good* (which is later verified).

If FIND-MATCH does not return *true*, the algorithm rejects. If FIND-MATCH returns *true*, then the algorithm continues in a unique path to complete the computation beyond this point, but only for f and not for m. This way, M, T and f are verified (although it is done $f - 1$ times). In the same way, we move through every $m < f$ and if the algorithm does not reject anywhere, it means our initial choice of f was correct.

References

1. Eric Allender. Reachability problems: An update. In *Proc. of CiE 2007*, pages 25–27, 2007.
2. Eric Allender, David A. Mix Barrington, Tanmoy Chakraborty, Samir Datta, and Sambuddha Roy. Planar and grid graph reachability problems. *Theor. Comp. Sys.*, 45(4):675–723, July 2009.
3. Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, New York, NY, USA, 1st edition, 2009.
4. Chris Bourke, Raghunath Tewari, and N. V. Vinodchandran. Directed planar reachability is in unambiguous log-space. *ACM Trans. Comput. Theory*, 1(1):4:1–4:17, February 2009.
5. Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with 0(1) worst case access time. *J. ACM*, 31(3):538–544, June 1984.
6. Brady Garvin, Derrick Stolee, Raghunath Tewari, and N.V. Vinodchandran. ReachFewL = ReachUL. *computational complexity*, 23(1):85–98, 2014.
7. Neil Immerman. Nondeterministic space is closed under complementation. *SIAM J. Comput.*, 17(5):935–938, October 1988.
8. Nutan Limaye, Meena Mahajan, and Prajakta Nimbhorkar. Longest paths in planar dags in unambiguous logspace. In *Proc. of CATS 2009*, pages 101–108, 2009.
9. Aduri Pavan, Raghunath Tewari, and N. V. Vinodchandran. On the power of unambiguity in log-space. *Computational Complexity*, 21(4):643–670, 2012.
10. Omer Reingold. Undirected st-connectivity in log-space. In *Proceedings of STOC 2005*, pages 376–385, 2005.
11. Klaus Reinhardt and Eric Allender. Making nondeterminism unambiguous. *SIAM J. Comput.*, 29(4):1118–1131, 2000.
12. R. Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Inf.*, 26(3):279–284, November 1988.