Trading Determinism for Time in Space Bounded Computations

Vivek Anand T Kallampally[1] **and Raghunath Tewari**[2]

1 **Department of Computer Science & Engineering, Indian Institute of Technology, Kanpur, India**
`vivekana@cse.iitk.ac.in`

2 **Department of Computer Science & Engineering, Indian Institute of Technology, Kanpur, India**
`rtewari@cse.iitk.ac.in`

Abstract

Savitch showed in 1970 that nondeterministic logspace (NL) is contained in deterministic $\mathcal{O}(\log^2 n)$ space but his algorithm requires quasipolynomial time. The question whether we can have a deterministic algorithm for every problem in NL that requires polylogarithmic space and simultaneously runs in polynomial time was left open.

In this paper we give a partial solution to this problem and show that for every language in NL there exists an unambiguous nondeterministic algorithm that requires $\mathcal{O}(\log^2 n)$ space and simultaneously runs in polynomial time.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes, F.2.3 Tradeoffs between Complexity Measures

Keywords and phrases space complexity, unambiguous computations, Savitch's Theorem

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.10

1 Introduction

Deciding reachability between a pair of vertices in a graph is an important computational problem from the perspective of space bounded computations. It is well known that reachability in directed graphs characterizes the complexity class nondeterministic logspace (NL). For undirected graphs the problem was known to be hard for the class deterministic logspace (L) and in a breakthrough result Reingold showed that is contained in L as well [20]. Several other restrictions of the reachability problem are known to characterize other variants of space bounded complexity classes [12, 5, 6].

Unambiguous computations are a restriction of general nondeterministic computations where the Turing machine has at most one accepting computation path on every input. In the space bounded domain, unambiguous logspace (in short UL) is the class of languages for which there is a nondeterministic logspace bounded machine that has a unique accepting path for every input in the language and zero accepting path otherwise. UL was first formally defined and studied in [8, 2]. In 2000 Reinhardt and Allender showed that the class NL is contained in a non-uniform version of UL [21]. In a subsequent work it was shown that under the hardness assumption that deterministic linear space has functions that cannot be computed by circuits of size $2^{\epsilon n}$, it can be shown that NL $=$ UL [1]. Although it is widely believed that NL and UL are the same unconditionally and in a uniform setting, the question still remains open.

Savitch's Theorem states that reachability in directed graphs is in $\mathsf{DSPACE}(\log^2 n)$, however the algorithm requires quasipolynomial time [22]. On the other hand standard graph traversal algorithms such as DFS and BFS can decide reachability in polynomial time (in fact linear time) but require linear space. Wigderson asked the question that can we solve reachability in $\mathcal{O}(n^{1-\epsilon})$ space and polynomial time simultaneously, for some $\epsilon > 0$ [26]. Barnes et. al. gave a partial answer to this question by giving a $\mathcal{O}(n/2^{\sqrt{\log n}})$ space and polynomial time algorithm for the problem [4]. Although this bound has been improved for several subclasses such as planar graphs [16], layered planar graphs [10], minor-free and bounded genus graphs [9], for general directed graphs (and hence for the class NL) we still do not have a better deterministic space upper bound simultaneously with polynomial time.

1.1 Main Result

In this paper we show that directed graph reachability can be decided by an unambiguous $\mathcal{O}(\log^2 n)$ space algorithm that simultaneously requires only polynomial time. Thus we get an improvement in the time required by Savitch's algorithm by sacrificing determinism. Formally, we show the following theorem.

▶ **Theorem 1.** $\mathsf{NL} \subseteq \mathsf{poly}\text{--}\mathsf{USPACE}(\log^2 n)$.

For the remainder of this paper all graphs that we consider are directed graphs unless stated otherwise.

1.2 Min-uniqueness of Graphs

An important ingredient of our proof is the *min-uniqueness* property of graphs. A graph G is said to be min-unique with respect to an edge weight function W if the minimum weight path between every pair of vertices in G is unique with respect to W. This turns out to be an important property and has been studied in earlier papers [27, 15, 21]. In fact, the fundamental component of Reinhardt and Allender's paper is a UL algorithm for testing whether a graph is min-unique and then deciding reachability in min-unique graphs in UL [21]. They achieve this by proposing a *double inductive counting* technique which is a clever adaptation of the inductive counting technique of Immerman and Szelepcsényi [17, 23]. As a result of Reinhardt and Allender's algorithm, in order to show that reachability in a class of graphs can be decided in UL, one only needs to design an efficient algorithm which takes as input a graph from this class and outputs an $\mathcal{O}(\log n)$ bit weight function with respect to which the graph is min-unique. This technique was successfully used to show a UL upper bound on the reachability problem in several natural subclasses of general graphs such as planar graphs [7], graphs with polynomially many paths from the start vertex to every other vertex [19], bounded genus graphs [11] and minor-free graphs [3]. For the latter two classes of graphs reachability was shown to be in UL earlier as well by giving reductions to planar graphs [18, 24]. Note that Reinhardt and Allender defines min-uniqueness for unweighted graphs where the minimum length path is unique, whereas we define it for weighted graphs where the minimum weight path is unique. However it can easily be seen that both these notions are equivalent.

1.3 Overview of the Proof

We prove Theorem 1 in two parts. We first show how to construct an $\mathcal{O}(\log^2 n)$ bit weight function W with respect to which the input graph G becomes min-unique. Our construction of the weight function W uses an iterative process to assign weights to the edges of G. We start by considering a subgraph of G having a fixed radius and construct an $\mathcal{O}(\log n)$ bit weight function with respect to which this subgraph becomes min-unique. For this we first observe that there are polynomially many paths in such a subgraph and then use the prime based hashing scheme of Fredman, Komlós and Szemerédi [14] to give distinct weights to all such paths. Thereafter, in each successive round of the algorithm, we construct a new weight function with respect to which a subgraph of double the radius of the previous round becomes min-unique and the new weight function has an additional $\mathcal{O}(\log n)$ bits. Hence in $\mathcal{O}(\log n)$ many rounds we get a weight function which has $\mathcal{O}(\log^2 n)$ bits and with respect to which G is min-unique. We show that this can be done by an unambiguous, polynomial time algorithm using $\mathcal{O}(\log^2 n)$ space. This technique is similar to the isolating weight construction in [13], but their construction is in $\mathsf{quasi{-}NC}$.

We then show that given a graph G and an $\mathcal{O}(\log^2 n)$ bit weight function with respect to which G is min-unique, reachability in G can be decided by an unambiguous, polynomial time algorithm using $\mathcal{O}(\log^2 n)$ space. Note that a straightforward application of Reinhardt and Allender's algorithm will not give the desired bound. This is because "unfolding" a graph with $\mathcal{O}(\log^2 n)$ bit weights will result in a quasipolynomially large graph. As a result we will not achieve a polynomial time bound. We tackle this problem by first observing that although there are $2^{\mathcal{O}(\log^2 n)}$ many different weight values, the weight of a shortest path can only use polynomial number of distinct such values. Using this observation we give a modified version of Reinhardt and Allender's algorithm that iterates over the "good" weight values and ignores the rest. This allows us to give a polynomial time bound.

The rest of the paper is organized as follows. In Section 2 we define the various notations and terminologies used in this paper. We also state prior results that we use in this paper. In Section 3 we give the proof of Theorem 1.

2 Preliminaries

For a positive integer n, let $[n] = \{1, 2, \dots, n\}$. Let $G = (V, E)$ be a directed graph on n vertices and let $E = \{e_1, e_2, \dots, e_m\}$ be the set of edges in G. Let s and t be two fixed vertices in G. We wish to decide whether there exists a path from s to t in G. The *length* of a path P is the number of edges in P and is denoted as $\text{len}(P)$. The *center* of a path P is a vertex x in P such that the length of the path from either end point of P to x is at most $\lceil \text{len}(P)/2 \rceil$ and x is no farther from the tail of P than from the head of P.

A *weight function* $w : E \to \mathbb{N}$ is a function which assigns a positive integer to every edge in G. The weight function w is said to be *polynomially bounded* if there exists a constant k such that $w(e) \leq \mathcal{O}(n^k)$ for every edge e in G. We use G_w to denote the weighted graph G with respect to a weight function w. For a graph G_w, the *weight of a path* P denoted by $w(P)$ is defined as the sum of weights of the edges in the path. A *shortest path* from u to v in G_w is a path from u to v with minimum weight. Let $\mathcal{P}^i_w(u, v)$ denote the set of shortest paths from u to v of length at most i in G_w. Thus in particular, the set of shortest paths from u to v in G_w, $\mathcal{P}_w(u, v) = \mathcal{P}^n_w(u, v)$.

We define the *distance* function with respect to a weight function and a nonnegative integer i as

$$\text{dist}^i_w(u, v) = \begin{cases} w(P) & \text{for } P \in \mathcal{P}^i_w(u, v) \\ \infty & \text{if } \mathcal{P}^i_w(u, v) = \emptyset \end{cases}$$

Correspondingly we define the function l which represents the minimum length of such paths as

$$\mathrm{l}_w^i(u,v) = \begin{cases} \min_{P \in \mathcal{P}_w^i(u,v)} \{\mathrm{len}(P)\} & \text{if } \mathcal{P}_w^i(u,v) \neq \emptyset \\ \infty & \text{otherwise} \end{cases}$$

A graph G_w is said to be *min-unique* for paths of length at most i, if for any pair of vertices u and v, the shortest path from u to v with length at most i, is unique. G_w is said to be min-unique if G_w is min unique for paths of arbitrary length. Define weight function

$$w_0(e_i) := 2^{i-1}, \text{ where } i \in [m].$$

It is straightforward to see that for any graph G, w_0 is an n bit weight function and G_{w_0} is min-unique. Wherever it is clear from the context that there is only one weight function w, we will drop the subscript w in our notations.

For a graph G_w, vertex u in G, length i and weight value k, we define the quantities $c_k^i(u)$ and $D_k^i(u)$ as the number of vertices at a distance at most k from u, using paths of length at most i and the sum of the distances to all such vertices respectively. Formally,

$$c_k^i(u) = |\{v \mid \mathrm{dist}_w^i(u,v) \leq k\}|$$
$$D_k^i(u) = \sum_{v \mid \mathrm{dist}_w^i(u,v) \leq k} \mathrm{dist}_w^i(u,v).$$

An *unambiguous Turing machine* is a nondeterministic Turing machine that has at most one accepting computation path on every input [25]. We shall consider unambiguous computations in the context of space bounded computations. $\mathsf{USPACE}(s(n))$ denotes the class of languages decided by an unambiguous machine using $\mathcal{O}(s(n))$ space. In particular, $\mathsf{UL} = \mathsf{USPACE}(\log n)$. $\mathsf{TIME{-}USPACE}(t(n), s(n))$ denotes the class of languages decided by an unambiguous machine using $\mathcal{O}(s(n))$ space and $\mathcal{O}(t(n))$ time simultaneously. In particular, when $t(n)$ is a polynomial, we define

$$\mathsf{poly{-}USPACE}(s(n)) = \bigcup_{k \geq 0} \mathsf{TIME{-}USPACE}(n^k, s(n)).$$

For graphs having polynomially many paths, we use the well known hashing technique due to Fredman, Komlós and Szemerédi [14] to compute a weight function that assigns distinct weights to all such paths. We state the result below in a form that will be useful for our purpose.

▶ **Theorem 2.** *[14, 19] 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 p so that for all $x \neq y \in S$, $x \not\equiv y \bmod p$.*

Henceforth we will refer to Theorem 2 as the FKS hashing lemma.

3 Min-unique Weight Assignment

Reinhardt and Allender [21] showed that for every n there is a sequence of n^2 $\mathcal{O}(\log n)$ bit weight functions such that every graph G on n vertices is min-unique with respect to at least one of them. For each weight function they construct an unweighted graph (say G_w) by replacing every edge with a path of length equal to the weight of that edge. Since the weights are $\mathcal{O}(\log n)$ bit values therefore G_w is polynomially large in n. Next they show

Algorithm 1: Computes a min-unique weight function and checks for an $s-t$ path in G

```
  Input: $(G, s, t)$
  Output: weight function $W := W_q$, true if there is a path from $s$ to $t$ and false
          otherwise
1 begin
2     $q := \log n$; $W_0 := 0$
3     for $j \leftarrow 1$ to $q$ do
4         $i := 2^j$; $p := 2$
5         repeat
              /* By the FKS hashing lemma $p$ is bounded by a polynomial in $n$,
                 say $n^{c'}$. We define $B := n^{c'+2}$.                      */
6             $W_j := B \cdot W_{j-1} + (w_0 \bmod p)$
7             Check whether $(G, W_j, i)$ is min-unique using Algorithm 2
8             $p :=$ next prime
9         until $(G, W_j, i)$ is min-unique
10    endfor
11    if $\mathrm{dist}^n_{W_q}(s,t) \leq B^q$ then return $(W_q, \mathsf{true})$
12    else return $(W_q, \mathsf{false})$
13 end
```

that using the double inductive counting technique one can check unambiguously using a logspace algorithm if G_w is min-unique, and if so then check if there is a path from s to t as well. They iterate over all weight functions until they obtain one with respect to which G_w is min-unique and use the corresponding graph G_w to check reachability. Since we use an $\mathcal{O}(\log^2 n)$ bit weight function with respect to which the input graph is min-unique, we cannot construct an unweighted graph by replacing every edge with a directed path of length equal to the corresponding edge weight.

In Section 3.1 we give an algorithm that computes an $\mathcal{O}(\log^2 n)$ bit, min-unique weight function and decides reachability in directed graphs. In Section 3.2 we check if a graph is min-unique. Although we use $\omega(\log n)$ bit weight functions, our algorithm still runs in polynomial time. In Section 3.3 we show how to compute the $\mathrm{dist}^i_w(u,v)$ function unambiguously.

3.1 Construction of the weight function

Theorem 3 shows how to construct the desired weight function.

▶ **Theorem 3.** *There is a nondeterministic algorithm that takes as input a directed graph G and outputs along a unique computation path, an $\mathcal{O}(\log^2 n)$ bit weight function W such that G_W is min-unique, while all other computation paths halt and reject. For any two vertices s and t the algorithm also checks whether there is a path from s to t in G. The algorithm uses $\mathcal{O}(\log^2 n)$ space and runs in polynomial time.*

Since directed graph reachability is complete for NL, Theorem 1 follows from Theorem 3.

Proof of Theorem 3. To prove Theorem 3 we design an algorithm that outputs the desired weight function. The formal description of the construction is given in Algorithm 1. The algorithm works in an iterative manner for $\log n$ number of rounds. Initially we consider all paths in G of length at most l where $l = 2^1$. The number of such paths is bounded by n^l and therefore by the FKS hashing lemma there exist a $c' \log n$ bit prime p_1 such that with respect to the weight function $W_1 := w_0 \bmod p_1$, G_{w_1} is min-unique for paths of length at most l. To find the right prime p_1 we iterate over all $c' \log n$ bit primes and use Lemma 7 to check whether G_{w_1} is min-unique for paths of length at most l.

We prove this by induction on the number of rounds, say j. Assume that $G_{W_{j-1}}$ is min-unique for paths of length at most 2^{j-1}. In the j-th round, the algorithm considers all paths of length at most 2^j. By applying Lemma 4 we get a weight function W_j from W_{j-1} which uses $\mathcal{O}(j \cdot \log n)$ bits and G_{W_j} is min-unique for paths of length at most 2^j. Hence in $\log n$ many rounds we get a weight function $W := W_{\log n}$ such that G_W is min-unique. Note that the inner **repeat-until** loop runs for at most $n^{c'}$ iterations due to the FKS hashing lemma.

Let p_j be the prime used in the j-th round of Algorithm 1. Define $p' := \max\{p_j \mid j \in [\log n]\}$. By the FKS hashing lemma p' is bounded by a polynomial in n, say $n^{c'}$. We set $B := n^{c'+2}$. This implies that for any weight function of the form $w = w_0 \bmod p_j$ and any path P in G, $w(P) < B$. Observe that with respect to the final weight function W, for any path P in G, $W(P) < B^q$.

Once we compute an $\mathcal{O}(\log^2 n)$ bit weight function W such that G_W is min-unique, there exist a path from s to t if and only if $\mathrm{dist}^n_W(s,t) \leq B^q$. This can be checked using Algorithm 5 in $\mathcal{O}(\log^2 n)$ space and polynomial time. Also Algorithm 5 is a nondeterministic algorithm which returns true or false along a unique computation path while all other computation paths halt and reject.

In each round the size of W_j increases by $\mathcal{O}(\log n)$ bits and after $\log n$ rounds $W_{\log n}$ is an $\mathcal{O}(\log^2 n)$ bit weight function. By Lemma 7 checking whether a graph is min-unique with respect to an $\mathcal{O}(\log^2 n)$ bit weight function requires $\mathcal{O}(\log^2 n)$ space. Thus the total space complexity of Algorithm 1 is $\mathcal{O}(\log^2 n)$.

The FKS hashing lemma guarantees that in each round only a polynomial number of primes need to be tested to find a weight function which is min-unique for paths of length at most 2^j. By Lemma 7 checking whether a graph is min-unique for paths of length at most 2^j can be done in polynomial time. Thus each round runs in polynomial time. There are only $\log n$ many round and hence Algorithm 1 runs in polynomial time.

By Lemma 7, Algorithm 2 is a nondeterministic algorithm which outputs its answer along a unique computation path, while all other computation paths halt and reject. All other steps in Algorithm 1 are deterministic. This shows the unambiguity requirement of the theorem. ◀

▶ **Lemma 4.** *There is a nondeterministic algorithm $\mathcal{A}$, that takes as inputs (G, w) where G is a graph on n vertices and w is a k bit weight function such that G_w is min-unique for paths of length at most l. $\mathcal{A}$ outputs a $(k + \mathcal{O}(\log n))$ bit weight function w' such that $G_{w'}$ is min-unique for paths of length at most $2l$, along a unique computation path while all other computation paths halt and reject. $\mathcal{A}$ uses $\mathcal{O}(k + \mathcal{O}(\log n))$ space and runs in polynomial time.*

▶ Remark. The encoding of the output weight function w' is the concatenation of the k bit representation of the input weight function w and an $\mathcal{O}(\log n)$ bit prime number p. The output weight function w' is calculated as $w' := B \cdot w + w_0 \bmod p$, where B is the number defined in Algorithm 1. Multiplication using B is used just to left shift w and make room for the new function $w_0 \bmod p$.

Lemma 4 proves the correctness of each iteration of the outer **for** loop of Algorithm 1. Before proving the lemma, we will show that if G_w is min-unique for paths of length at most l, then the number of minimum weight paths with respect to w of length at most $2l$ is bounded by a polynomial independent of l. Hence it allows us to use the FKS hashing lemma to isolate such paths.

▶ **Lemma 5.** *Let G be a graph with n vertices and w be a weight function such the graph G_w is min-unique for paths of length at most l. Then for any pair of vertices u and v, $\left|\mathcal{P}_w^{2l}(u,v)\right|$ is at most n.*

Proof. Let P be a shortest path from u to v in G_w with length at most $2l$ with center vertex x. That is $P \in \mathcal{P}_w^{2l}(u,v)$. Let P_1 and P_2 be the subpaths from u to x and x to v. Since x is the center of P, P_1 has length at most l. Note that P_1 is the unique shortest path of length at most l from u to x in G_w. This is because if there exists another path of length at most l with a smaller weight than P_1 from u to x then replacing P_1 with this path in P will result in a path of length at most $2l$ from u to v with a lower weight than P. But this cannot happen since P is a shortest path from u to v.

▶ **Claim 6.** *There is only one shortest path of length at most $2l$ from u to v with x as its center.*

Proof. Assume there is another shortest path P' of length at most $2l$ from u to v with x as its center. Let P'_1 be the subpath of P' from u to x. Since x is the center of P', P'_1 is of length at most l. Similar to P_1, P'_1 is a shortest path of length at most l from u to x. This means there are two shortest paths of length at most l from u to x. This is a contradiction since G is min-unique for paths of length at most l. ◀

Therefore each vertex can be the center of at most one path of length at most $2l$ from u to v. Thus the total number of shortest paths of length at most $2l$ from u to v in G_w is at most n. Hence $\left|\mathcal{P}_w^{2l}(u,v)\right| \leq n$. This completes the proof of Lemma 5. ◀

When we sum over all possible pairs of u and v, the total number of shortest paths of length at most $2l$ in G_w is at most n^3.

Proof of Lemma 4. G_w is min-unique for paths of length at most l. Therefore by Lemma 5 the number of shortest paths between all pairs of vertices with at most $2l$ edges in G is at most n^3. Let $\mathcal{S}$ be the set of these n^3 shortest paths. With respect to the weight function w_0 (see Section 2) each element of $\mathcal{S}$ gets a distinct weight. So by using the FKS hashing lemma we get a constant c' and a $c' \log n$ bit prime number p such that with respect to the weight function $\widehat{w}$ such that $\widehat{w} := w_0 \bmod p$, each element of $\mathcal{S}$ gets a distinct weight. Moreover, in G between any pair of vertices the shortest path in $\mathcal{S}$ is unique.

Let B be the number as defined in Algorithm 1. Now consider the weight function $w' := B \cdot w + \widehat{w}$. Since w is a k bit weight function and $\widehat{w}$ is an $\mathcal{O}(\log n)$ bit weight function therefore w' is a $(k + \mathcal{O}(\log n))$ bit weight function. Clearly w has higher precedence than $\widehat{w}$ in w'. So for any two paths P_1 and P_2 in G , we have if $w'(P_1) < w'(P_2)$ then either $w(P_1) < w(P_2)$ or both the predicates $w(P_1) = w(P_2)$ and $\widehat{w}(P_1) < \widehat{w}(P_2)$ are true. Additionally if $w'(P_1) = w'(P_2)$ then $w(P_1) = w(P_2)$ and $\widehat{w}(P_1) = \widehat{w}(P_2)$.

All the unique shortest paths of length at most $2l$ in G_w, will be unique shortest paths of length at most $2l$ in $G_{w'}$ also. If there are multiple shortest paths of length at most $2l$ from u to v in G_w, $\widehat{w}$ gives a unique weight to each of these paths. So $G_{w'}$ is min-unique for paths of length at most $2l$.

Algorithm 2: Check whether G is min-unique for paths of length at most i

```
Input: $(G, w, i)$
Output: true if $G_w$ is not min-unique for paths of length at most $i$ and false otherwise
1  begin
2      BAD.WEIGHT := false
       /* BAD.WEIGHT is set to true whenever the weight function does not
          make the graph min-unique. Otherwise it remains false. It is a
          boolean variable shared between Algorithms 4 and 2              */
3      for each vertex $v$ do
4          $c_0^i(v) := 1$; $D_0^i(v) := 0$; $k' := 0$
5          repeat
6              $k := k'$; $c_k^i(v) := c_{k'}^i(v)$; $D_k^i(v) := D_{k'}^i(v)$
7              Find next $k'$ from $(G, w, v, i, k, c_k^i(v), D_k^i(v))$ using Algorithm 3
8              if $k' = \infty$ then break
9              Compute $(c_{k'}^i(v), D_{k'}^i(v))$ from $(G, w, v, i, k, c_k^i(v), D_k^i(v), k')$ using
               Algorithm 4
10         until BAD.WEIGHT = true
11         if BAD.WEIGHT = true then break
12     endfor
13     return BAD.WEIGHT
14 end
```

We can check whether a graph $G_{w'}$ is min-unique for paths of length at most $2l$ using Lemma 7. Since p is an $c' \log n$ bit prime number, we can iterate over all the $c' \log n$ bit primes and find p. ◀

3.2 Checking for min-uniqueness

The next lemma shows how to check whether G_w is min-unique for paths of length at most l in an unambiguous manner.

▶ **Lemma 7.** *There is a nondeterministic algorithm that takes as input a directed graph G, a k bit weight function w and a length i and outputs along a unique computation path whether or not the graph G_w is min-unique for paths of length at most i, while all other computation paths halt and reject. The algorithm uses $\mathcal{O}(k + \log n)$ space and runs in polynomial time.*

For every vertex v in the G_w we check whether there are two minimum weight paths of length at most i to some other vertex in G. Algorithm 2 gives a formal description of this process. The algorithm iterates over all shortest path weight values that can be achieved by some path of length at most i.

In the k-th stage of the algorithm it considers a ball of radius k consisting of vertices which have a shortest path of weight at most k from v and length at most i. $c_k^i(v)$ denotes the number of vertices in this ball and $D_k^i(v)$ denotes the sum of the weights of the shortest paths to all such vertices. Initially $k = 0$, $c_0^i(v) = 1$ (consisting of only the vertex v) and $D_0^i(v) = 0$.

A direct implementation of the double inductive counting technique of Reinhardt and Allender [21] does not work since this would imply that we cycle over all possible weight values, which we cannot afford. We bypass this hurdle by considering only the relevant

Algorithm 3: Find the next smallest weight value $k' > k$ among all paths of length at most i from u

```
Input: $(G, w, u, i, k, c_k^i(u), D_k^i(u))$
Output: $k' := \min\{\text{dist}_w^i(u,v) \mid \text{dist}_w^i(u,v) > k,\ v \in V\}$
begin
    $k' := \infty$
    for each vertex $v$ do
        if $\neg(\text{dist}_w^i(u,v) \leq k)$ then
            $\text{min.dist}_w^i(u,v) := \infty$
            for each $x$ such that $(x,v)$ is an edge do
                if $\text{dist}_w^i(u,x) \leq k$ and $\text{l}_w^i(u,x) + 1 \leq i$ then
                    if $\text{min.dist}_w^i(u,v) > \text{dist}_w^i(u,x) + w(x,v)$ then
                        $\text{min.dist}_w^i(u,v) := \text{dist}_w^i(u,x) + w(x,v)$
                    endif
                endif
            endfor
            if $k' > \text{min.dist}_w^i(u,v)$ then $k' := \text{min.dist}_w^i(u,v)$
        endif
    endfor
    return $k'$
end
```

weight values. We compute the immediate next shortest path weight value k', and use k' as the weight value for the next stage of the algorithm. This computation is implemented in Algorithm 3). Lemma 8 proves the correctness of this process. Note that the number of shortest path weight values from a fixed vertex is bounded by the number of vertices in the graph. This ensure that the number of iterations of the inner **repeat-until** loop of Algorithm 2 is bounded by n.

▶ **Lemma 8.** *Given $(G, w, u, i, k, c_k^i(u), D_k^i(u))$, Algorithm 3 correctly computes the value $\min\{\text{dist}_w^i(u,v) \mid \text{dist}_w^i(u,v) > k,\ v \in V\}$.*

To see the correctness of Lemma 8 observe that for every vertex v such that $\text{dist}_w^i(u,v) > k$, the algorithm cycles through all vertices x such that there is an edge from x to v and the length of the path from u to x is at most $i-1$. It computes the minimum weight of such a path and store it in the variable $\text{min.dist}_w^i(u,v)$. It then computes the minimum value of $\text{min.dist}_w^i(u,v)$ over all possible vertices v and outputs it as k', as required.

After we get the appropriate weight value k', we then compute the values of $c_{k'}^i(v)$ and $D_{k'}^i(v)$ by using a technique similar to Reinhardt and Allender (implemented in Algorithm 4). Additionally we also maintain a shared flag value BAD.WEIGHT between Algorithms 2 and 4, which is set to true if G_w is not min-unique for paths of length at most i, else it is false.

3.3 Computing the $\text{dist}_w^i(u,v)$ function

In Algorithms 3 and 4, an important step is to check whether $\text{dist}_w^i(u,v) \leq k$ and if so, get the values of $\text{dist}_w^i(u,v)$ and $\text{l}_w^i(u,v)$. These values are obtained from Algorithm 5. Algorithm 5 describes a nondeterministic procedure that takes as input a weighted graph G_w, which is min-unique for paths of length at most i and weight at most k from a source

Algorithm 4: Compute $c^i_{k'}(u)$ and $D^i_{k'}(u)$ and check whether G_w is min-unique for paths with length at most i and weight at most k' from u

```
   Input: $(G, w, u, i, k, c^i_k(u), D^i_k(u), k')$
   Output: $(c^i_{k'}(u), D^i_{k'}(u))$ and also flag BAD.WEIGHT
1  begin
2      $c^i_{k'}(u) := c^i_k(u)$; $D^i_{k'}(u) := D^i_k(u)$
3      for each vertex $v$ do
4          if $\neg(\mathrm{dist}^i_w(u,v) \leq k)$ then
5              for each $x$ such that $(x, v)$ is an edge do
6                  if $\mathrm{dist}^i_w(u,x) \leq k$ and $\mathrm{dist}^i_w(u,x) + w(x,v) = k'$ and $\mathrm{l}^i_w(u,x) + 1 \leq i$
                   then
7                      $c^i_{k'}(u) := c^i_{k'}(u) + 1$; $D^i_{k'}(u) := D^i_{k'}(u) + k'$
8                      for each $x' \neq x$ such that $(x', v)$ is an edge do
9                          if $\mathrm{dist}^i_w(u,x') \leq k$ and $\mathrm{dist}^i_w(u,x') + w(x',v) = k'$ and
                           $\mathrm{l}^i_w(u,x') + 1 \leq i$ then
10                             BAD.WEIGHT := true
11                         endif
12                     endfor
13                 endif
14             endfor
15         endif
16     endfor
17     return $(c^i_{k'}(u), D^i_{k'}(u))$
18 end
```

vertex u and the values $c^i_k(u)$ and $D^i_k(u)$. For any vertex v, if $\mathrm{dist}^i_w(u,v) \leq k$ then it outputs true and the values of $\mathrm{dist}^i_w(u,v)$ and $\mathrm{l}^i_w(u,v)$ along a unique computation path. Otherwise it outputs false along a unique computation path with ∞ as the values of $\mathrm{dist}^i_w(u,v)$ and $\mathrm{l}^i_w(u,v)$. All other computation paths halt and reject. As a result we can compute the predicate $\neg(\mathrm{dist}^i_w(u,v) \leq k)$ along a unique path as well.

Note that Algorithm 5 is the only algorithm where we use non-determinism. The algorithm is similar to the unambiguous subroutine of Reinhardt and Allender [21] with the only difference being that here we consider weight of a path instead of length of a path. The algorithm assumes that the subgraph induced by all the paths of length at most i and weight at most k from u is min-unique.

In Line 5 of Algorithm 5, for each vertex x the routine non-deterministically guesses whether $\mathrm{dist}^i_w(u,x) \leq k$ and if the guess is 'true', it then guesses a path of length at most k from u to x. If the algorithm incorrectly guesses for some vertex x that $\mathrm{dist}^i_w(u,x) > k$, then the variable *count* will never reach $c^i_k(u)$ and the routine will reject. If it guesses incorrectly that $\mathrm{dist}^i_w(u,x) \leq k$ it will fail to guess a correct path in Line 7 and again reject that computation. Thus the only computation paths that exit the **for** loop in Line 16 and satisfy the first condition of the **if** statement in Line 17, are the ones that correctly guess exactly the set $\{x \mid \mathrm{dist}^i_w(u,x) \leq k\}$. If the algorithm ever guesses incorrectly the weight d of the shortest path to x, then if $\mathrm{dist}^i_w(u,x) > d$ no path of weight d will be found, and if $\mathrm{dist}^i_w(u,x) < d$ then the variable *sum* will be incremented by a value greater than $\mathrm{dist}^i_w(u,x)$. In the latter case, at the end of the algorithm, *sum* will be greater than $D^i_k(u)$, and the routine will reject.

Algorithm 5: An unambiguous routine to determine if $\mathrm{dist}^i_w(u,v) \le k$ and find $\mathrm{dist}^i_w(u,v)$ and $\mathrm{l}^i_w(u,v)$

```
Input: $(G, w, u, i, k, c^i_k(u), D^i_k(u), v)$
Output: (true or false), $\mathrm{dist}^i_w(u,v)$, $\mathrm{l}^i_w(u,v)$
begin
    $count := 0$; $sum := 0$; $path.to.v :=$ false
    $\mathrm{dist}^i_w(u,v) := \infty$; $\mathrm{l}^i_w(u,v) := \infty$
    for each $x \in V$ do
        Guess non deterministically if $\mathrm{dist}^i_w(u,x) \le k$ in $G_w$
        if the guess is $\mathrm{dist}^i_w(u,x) \le k$ then
            Guess a path of weight $d \le k$ and length $l \le i$ from $u$ to $x$
            (If this fails then halt and reject)
            $count := count + 1$; $sum := sum + d$
            if $x = v$ then
                $path.to.v :=$ true
                $\mathrm{dist}^i_w(u,v) := d$
                $\mathrm{l}^i_w(u,v) := l$
            endif
        endif
    endfor
    if $count = c^i_k(u)$ and $sum = D^i_k(u)$ then
        return $(path.to.v, \mathrm{dist}^i_w(u,v), \mathrm{l}^i_w(u,v))$
    else
        halt and reject
    endif
end
```

Since G_w is min-unique for paths of length at most i and weight at most k from u, only for exactly one computation path *sum* and *count* will match with $c^i_k(u)$ and $D^i_k(u)$. So except one computation path which made all the guesses correct, all other paths halt and reject. If $\mathrm{dist}^i_w(u,v) \le k$ then even though the algorithm uses non-deterministic choices, it outputs 'true' along a single computation path while all other paths halt and reject. Also if $\mathrm{dist}^i_w(u,v) > k$, the algorithm outputs 'false' along a single computation path while all other paths halt and reject. The space complexity of the algorithm is bounded by the size of the weight function w.

As a corollary of Theorem 1 we get the following result.

▶ **Corollary 9.** *For $s(n) \ge \log n$,* $\mathsf{NSPACE}(s(n)) \subseteq \mathsf{TIME{-}USPACE}(2^{\mathcal{O}(s(n))}, s^2(n))$.

References

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

2 Carme Àlvarez and Birgit Jenner. A very hard log-space counting class. *Theoretical Computer Science*, 107:3–30, 1993.

3 Rahul Arora, Ashu Gupta, Rohit Gurjar, and Raghunath Tewari. Derandomizing isolation lemma for $k_{3,3}$-free and k_5-free bipartite graphs. In *33rd Symposium on Theoretical Aspects*

of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France, pages 10:1–10:15, 2016.

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

5 David A. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *Journal of Computer and System Sciences*, 38:150–164, 1989.

6 David A. Mix Barrington, Chi-Jen Lu, Peter Bro Miltersen, and Sven Skyum. Searching constant width mazes captures the AC^0 hierarchy. In *15th International Symposium on Theoretical Aspects of Computer Science (STACS)*, Volume 1373 in Lecture Notes in Computer Science, pages 74–83. Springer, 1998.

7 Chris Bourke, Raghunath Tewari, and N. V. Vinodchandran. Directed planar reachability is in unambiguous log-space. In *22nd Annual IEEE Conference on Computational Complexity (CCC 2007), 13-16 June 2007, San Diego, California, USA*, pages 217–221, 2007. doi:10.1109/CCC.2007.9.

8 Gerhard Buntrock, Birgit Jenner, Klaus-Jörn Lange, and Peter Rossmanith. Unambiguity and fewness for logarithmic space. In *Proceedings of the 8th International Conference on Fundamentals of Computation Theory (FCT'91)*, Volume 529 Lecture Notes in Computer Science, pages 168–179. Springer-Verlag, 1991.

9 Diptarka Chakraborty, Aduri Pavan, Raghunath Tewari, N. V. Vinodchandran, and Lin F. Yang. New time-space upperbounds for directed reachability in high-genus and h-minor-free graphs. In *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014, December 15-17, 2014, New Delhi, India*, pages 585–595, 2014.

10 Diptarka Chakraborty and Raghunath Tewari. An $O(n^\epsilon)$ space and polynomial time algorithm for reachability in directed layered planar graphs. In *In 26th International Symposium on Algorithms and Computation, ISAAC 2015, Nagoya, Japan, December 9-11, 2015*, pages 614–624, 2015.

11 Samir Datta, Raghav Kulkarni, Raghunath Tewari, and N. V. Vinodchandran. Space Complexity of Perfect Matching in Bounded Genus Bipartite Graphs. In *28th International Symposium on Theoretical Aspects of Computer Science (STACS 2011)*, volume 9 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 579–590, Dagstuhl, Germany, 2011. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.STACS.2011.579.

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

13 Stephen A. Fenner, Rohit Gurjar, and Thomas Thierauf. Bipartite perfect matching is in quasi-nc. *CoRR*, abs/1601.06319, 2016. URL: http://arxiv.org/abs/1601.06319.

14 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. doi:10.1145/828.1884.

15 Anna Gál and Avi Wigderson. Boolean complexity classes vs. their arithmetic analogs. *Random Struct. Algorithms*, 9(1-2):99–111, 1996. URL: http://onlinelibrary.wiley.com/doi/10.1002/(SICI)1098-2418(199608/09)9:1/2<99::AID-RSA7>3.0.CO;2-6/abstract.

16 T. Imai, K. Nakagawa, A. Pavan, N.V. Vinodchandran, and O. Watanabe. An $O(n^{1/2+\epsilon})$-Space and Polynomial-Time Algorithm for Directed Planar Reachability. In *Computational Complexity (CCC), 2013 IEEE Conference on*, pages 277–286, 2013. doi:10.1109/CCC.2013.35.

17 Neil Immerman. Nondeterministic space is closed under complement. *SIAM Journal on Computing*, 17:935–938, 1988.

18 Jan Kynčl and Tomáš Vyskočil. Logspace reduction of directed reachability for bounded genus graphs to the planar case. *ACM Transactions on Computation Theory*, 1(3):1–11, 2010. `doi:10.1145/1714450.1714451`.

19 Aduri Pavan, Raghunath Tewari, and N. V. Vinodchandran. On the power of unambiguity in log-space. *Computational Complexity*, 21(4):643–670, 2012. `doi:10.1007/s00037-012-0047-3`.

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

21 Klaus Reinhardt and Eric Allender. Making nondeterminism unambiguous. *SIAM J. Comput.*, 29(4):1118–1131, 2000. `doi:10.1137/S0097539798339041`.

22 Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4:177–192, 1970.

23 Robert Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26:279–284, 1988.

24 Thomas Thierauf and Fabian Wagner. Reachability in $K_{3,3}$-free Graphs and K_5-free Graphs is in Unambiguous Log-Space. In *17th International Conference on Foundations of Computation Theory (FCT)*, Lecture Notes in Computer Science 5699, pages 323–334. Springer-Verlag, 2009.

25 Leslie G. Valiant. Relative complexity of checking and evaluating. *Inf. Process. Lett.*, 5(1):20–23, 1976. `doi:10.1016/0020-0190(76)90097-1`.

26 Avi Wigderson. The complexity of graph connectivity. *Mathematical Foundations of Computer Science 1992*, pages 112–132, 1992.

27 Avi Wigderson. $NL/poly \subseteq \oplus L/poly$ (preliminary version). In *Proceedings of the Ninth Annual Structure in Complexity Theory Conference, Amsterdam, The Netherlands, June 28 - July 1, 1994*, pages 59–62, 1994. `doi:10.1109/SCT.1994.315817`.