Inductive Tracing and the Complexity of Finding Hamiltonian Path in DAGs

Ronak Bhadra[1] and Raghunath Tewari[1]

Indian Institute of Technology, Kanpur
{ronakb,rtewari}@cse.iitk.ac.in

Abstract. The HamiltonianPath problem is a classic decision problem that is NP-complete in general graphs, but solvable in linear time and in nondeterministic logspace (NL) for directed acyclic graphs (DAGs).
We show that the HamiltonianPath problem for DAGs lies in $\mathsf{UL} \cap \mathsf{coUL}$, providing an improved upper bound on its complexity. To the best of our knowledge, this is the first instance where an unambiguous space upper bound has been established for a natural problem without relying on any variant of the classic Reinhardt-Allender double inductive counting technique, which is itself based on the Immerman-Szelepcsényi inductive counting method.
Our proof introduces a novel technique, which we call *inductive tracing*. While reminiscent of inductive counting, it diverges in a key respect: it does not perform any counting across recursive stages. We also apply this routine to obtain a parameterized unambiguous space bound for the LongPath problem in DAGs.

Keywords: Unambiguous Logspace Computations · Hamiltonian Path · Directed Acyclic Graphs

1 Introduction

A Hamiltonian path in a graph G is a path in G that visits each vertex of G exactly once. Deciding whether a graph has a Hamiltonian path or not is known as the HamiltonianPath problem. A graph having a Hamiltonian path is called a *traceable* graph. The HamiltonianPath problem is a well-studied problem in the field of computational complexity, especially because of the fact that it is famously known to be NP-complete [6]. However, for certain restricted classes of graphs, the HamiltonianPath problem can be shown to be much easier. For example, the HamiltonianPath problem can be solved in linear time for rectangular grid graphs [4]. Similarly, for Directed Acyclic Graphs or DAGs, the HamiltonianPath problem can also be solved in linear time. Moreover, the HamiltonianPath problem for DAGs can be shown to be in the complexity class NL as well. This raises the question whether we can further reduce the computational complexity of the HamiltonianPath problem for DAGs.

A related problem of relevance is the LongPath problem, which is deciding whether there is a path of length at least k from one vertex to another in a

graph. The HamiltonianPath problem is a special case of the LongPath problem. Though the LongPath problem is NP-complete for general graphs and even for planar graphs [2], but it is known to be NL-complete for DAGs.

The complexity class UL is the unambiguous subclass of NL and coUL is the class of the complements of the languages (problems) in UL. A decision problem L is in UL if and only if there exists a non-deterministic Turing Machine M deciding L in logspace such that, for every input instance x, M has at most one accepting computation path. The first major breakthrough in the study of the class UL came with the work of Reinhardt-Allender [9], where they showed that the classes UL and NL are equivalent in non-uniform settings. They also showed that the Reachability problem in min-unique graphs is in $\mathsf{UL} \cap \mathsf{coUL}$. Using their technique of double inductive counting, Reachability in several other restricted classes of graphs (for example, planar graphs [1]) as well as some other problems (for example, longest paths in planar DAGs [7], 3-connected planar graph isomorphism [11]) have been shown to be in $\mathsf{UL} \cap \mathsf{coUL}$. Apart from this, using the double inductive counting technique, it has also been shown that directed graph reachability is solvable in unambiguous $O(\log^2 n)$ space and polynomial time simultaneously [5], which was later further improved to unambiguous $O(\log^{1.5} n)$ space and polynomial time[8]. In fact, to the best of our knowledge, any upper bound given till now on the unambiguous space complexity of any natural problem has been proved using some variant of the double inductive counting technique of Reinhardt-Allender[9].

In this paper, we show that the HamiltonianPath problem in Directed Acyclic Graphs or DAGs is in $\mathsf{UL} \cap \mathsf{coUL}$. For this purpose, we give an unambiguous and co-unambiguous non-deterministic algorithm that decides the HamiltonianPath problem in G. Our algorithm is akin to the inductive counting algorithm of Immerman-Szelepcsényi [3,10] and the double inductive counting algorithm of Reinhardt-Allender [9], which is inspired from the former. However, unlike the above mentioned algorithms, our algorithm doesn't count anything inductively. Rather, it inductively traces the Hamiltonian path if any such path exists, or else rejects the input if at any point it fails to do so. We call this novel technique *inductive tracing*. As far as we are aware, this is the first natural problem for which an upper bound on the unambiguous space complexity is proved without relying on the double inductive counting technique of Reinhardt-Allender[9]. We also give a parametrized upper bound on the unambiguous space complexity of the LongPath problem in DAGs using our $\mathsf{UL} \cap \mathsf{coUL}$ routine for deciding HamiltonianPath in DAGs.

This result is slightly non-intuitive (if not, surprising) because in general, the HamiltonianPath problem is a much more difficult problem than the Reachability problem. In certain restricted subclasses of graphs, the HamiltonianPath problem may be easier but it still intuitively seems to be at least as difficult as the Reachability problem for those subclasses. Any prior result regarding HamiltonianPath does not violate this intuition to the best of our knowledge. For example, in planar DAGs, though HamiltonianPath is in $\mathsf{UL} \cap \mathsf{coUL}$ but Reachability is also in $\mathsf{UL} \cap \mathsf{coUL}$. In fact, the proof for this [7] explicitly uses Reachability to solve

LongPath or HamiltonianPath for that matter. However, Reachability in DAGs is NL-complete. Our result thus implies that the HamiltonianPath problem is in some sense easier than the Reachability problem for DAGs. This of course may not be true if NL = UL, but this is a long-standing open question anyway.

Another interesting aspect that we want to highlight is that HamiltonianPath in DAGs has a natural UL algorithm because of the fact that there is at most one Hamiltonian path in a DAG. However, this automatically doesn't imply that HamiltonianPath is in coUL. In fact, proving a similar result for st-reachability in graphs where there is at most one path from s to t (for which again there is a natural UL algorithm) would prove that UL = coUL. This is unlike most other problems shown to be in UL ∩ coUL, which have no natural UL algorithm apart from some inductive counting algorithm based on the algorithm of Reinhardt-Allender [9].

The rest of the paper is organized as follows. In section 2, we will be proving the following theorem, which serves as our main result.

Theorem 1. *The problem of deciding whether a given Directed Acyclic Graph has a Hamiltonian path or not is in* UL ∩ coUL.

In section 3, we discuss some consequences of our main result. We give a parametrized upper bound on the unambiguous space complexity of the LongPath problem in DAGs. We also show that reachability in traceable DAGs is in UL ∩ coUL.

2 Proof of Theorem 1

In this section, we will prove theorem 1. First we will show that a DAG has at most one Hamiltonian path and how this fact can help us devise a natural UL algorithm for HamiltonianPath problem. In order to show that HamiltonianPath is also in coUL, we will provide an inductive non-deterministic algorithm and then prove that this algorithm works unambiguously (that is, accepts or rejects exactly along one computation path) for all input instances. We will then argue for the correctness of the algorithm in deciding whether a DAG has a Hamiltonian path or not.

We will first show that there is at most one Hamiltonian path in a DAG.

Lemma 1. *There is at most one Hamiltonian path in a Directed Acyclic Graph.*

Proof. We prove this by contradiction. Let us assume there are two Hamiltonian paths in a DAG G, say P_1 and P_2. There must be at least one vertex v which is at different positions in paths P_1 and P_2, say i and j respectively. Otherwise, P_1 and P_2 are identical. Without loss of generality, let us assume $i < j$. There must be some vertex u which comes after v in path P_1 but comes before v in path P_2. Now, if we take the subpath from v to u in P_1 and then the subpath from u to v in P_2, we get a cycle $(v \dots u \dots v)$. This is a contradiction to our assumption that G is acyclic.

Now, our unambiguous logspace routine to decide if a DAG G has a Hamiltonian path or not is simple. Non-deterministically guess a walk of length n (number of nodes in G) in G. Accept if the guess succeeds. Reject if the guess fails. Since there is only one Hamiltonian path in a DAG and every walk in a DAG is a path, therefore we can guarantee that our algorithm will accept along exactly one computation path, if a Hamiltonian path exists in G and will reject along all computation paths if no Hamiltonian path exists in G.

The existence of a unique Hamiltonian path in a DAG is sufficient to show that the HamiltonianPath problem for a DAG is in UL. However, it doesn't automatically imply that the HamiltonianPath problem for a DAG is in coUL as well. For this purpose, we devise algorithm 1 which uses algorithm 2 as a subroutine. Algorithm 1 is an inductive algorithm akin to the inductive counting algorithms of Immerman–Szelepcsényi [3,10] and Reinhardt-Allender [9], but unlike those algorithms, our algorithm doesn't actually *count* any parameter inductively. Rather, it inductively computes the next vertex in the (possible) Hamiltonian path from the previous vertex. In this way, the algorithm is able to trace the Hamiltonian path if it exists and otherwise, it is able to detect that such a path doesn't exist.

The algorithm first checks if the graph has a single source (and sink) or not. If the answer is yes, the algorithm proceeds. Otherwise, it rejects the input (that is, it declares that there is no Hamiltonian path in the graph G). The algorithm then starts by initializing a variable v as the (unique) source vertex (say s) of G. Now, the algorithm, in its i-th iteration, counts the number of vertices (say m_i) which have an incoming edge from v but no incoming edges from vertices other than those that belong to the (unique) path of length $i-1$ from s to v. This is done using the function COUNT, which non-deterministically guesses a path of length l from s to v and simultaneously counts the number of incoming edges to vertex v' from vertices on this path. The algorithm rejects if in any iteration i, m_i does not turn out to be one. The algorithm accepts only when it successfully completes n iterations. In other words, the algorithm accepts if and only if the value of m_i is one for all $i \leq n$.

We now provide the pseudocode for algorithm 1 and the subroutine COUNT.

Algorithm 1 Determine whether a DAG G has a Hamiltonian path

Require: $G = (V, E)$
Check if the graph has exactly one source and one sink.
if no **then**
    return reject
else
    set s to be the source vertex of G
Initialize $v = s$, $l = 1$
while $l \neq n$ **do**
    Initialize $m = 0$, $w = NULL$
    for each $v' \in V$ such that $(v, v') \in E$ **do**
        $d =$COUNT(G, s, v, v', l)
        if $d =$"?" **then**
            return "?"
        if indegree(v')=d **then**
            $m = m + 1$
            $w = v'$
    if $m \neq 1$ **then**
        return reject
    else
        $v = w$
        $l = l + 1$
return accept

Algorithm 2 Count the number of vertices, in a path of length l from s to v in G, from which there is an edge to v'

function COUNT$(G = (V, E), s, v, v', l)$
    Initialize $x = s$, $d = 0$, $i = 0$
    while $x \neq v$ and $i \neq l$ **do**
        if $(x, v') \in E$ **then**
            $d = d + 1$
        non-deterministically guess a vertex x' such that $(x, x') \in E$
        if guess is correct **then**
            $i = i + 1$
            $x = x'$
        else
            return "?"
    if $(x, v') \in E$ **then**
        $d = d + 1$
    if $x = v$ and $i = l$ **then**
        return d
    else
        return "?"

In order to prove that HamiltonianPath problem is in $\mathsf{UL} \cap \mathsf{coUL}$, we need to show the following things:

- algorithm 1 accepts or rejects an input along exactly one computation path for all input instances (DAGs).
- algorithm 1 accepts an input (a DAG G) if and only if G has a Hamiltonian path.

In section 2.1, we prove that the algorithm works unambiguously for all input instances. In section 2.2, we argue for the correctness of the algorithm in determining whether a DAG is Hamiltonian or not. For the sake of easier readability of the proof, we will define v_i to be the value of the variable v at the beginning of the i-th iteration of the while loop (in line 7) of algorithm 1. Note that, by this definition, v_1 happens to be the source vertex of the input graph G, if G has a unique source vertex. We will also assume the input to algorithm 1 is a DAG named G.

2.1 Unambiguity of Algorithm 1

In this subsection, we will show that algorithm 1 works unambiguously, that is, accepts or rejects along exactly one computation path, for all input instances.

We first prove the following lemma.

Lemma 2. *Let v_i be the value of the variable v at the beginning of the i-th iteration of the while loop (in line 7) of algorithm 1. There is a unique path $(v_1, v_2, \ldots, v_i)$ of length $(i-1)$ from the source v_1 to vertex v_i in G for all i. Also, there are no incoming edges to vertex v_i from any vertex other than $v_1, v_2, \ldots, v_{i-1}$.*

Proof. We will prove this by induction.

The base case is trivial. There is a unique path of length 0 from v_1 to v_1 (itself), which is the source vertex itself (v_1). There are no incoming edges to vertex v_1 since v_1 is the source vertex of G.

Let us assume that the lemma is true for all $i < j$. We will show that the lemma then must also be true for $i = j$.

Now let us assume there is another path (say P) of length $(j-1)$ from vertex v_1 to v_j, other than $(v_1, v_2, \ldots, v_j)$. There must be at least one vertex other than vertices $v_1, \ldots, v_j$ in P. Let u be the last such vertex in P. Let the vertex succeeding u in P be v_k. As per our assumption, $v_k \in \{v_1, v_2, \ldots, v_j\}$. Now, this implies that there is an edge from vertex u to vertex v_k. However, according to our induction hypothesis, if $k < j$, there is no incoming edge to v_k from any vertex other than $v_1, v_2, \ldots, v_{k-1}$, which will be a contradiction.

During the $(j-1)$-th iteration of the while loop in line 7 of algorithm 1, the algorithm counts the total number d of incoming edges to a vertex v' (which has an edge from vertex v_{j-1}) from all vertices in the path of length $(j-2)$ from vertex v_1 to v_{j-1} (which are the vertices $v_1, v_2, \ldots, v_{j-1}$ by our induction hypothesis). The algorithm checks if d is equal to the indegree of vertex v' and

increments the variable m by 1 if it is. At the end of the for loop, m indicates the number of vertices which have an incoming edge from v_{j-1} but has no incoming edge from vertices other than $v_1, v_2, \ldots, v_{j-1}$. The algorithm proceeds further only if $m = 1$ and sets the variable v to be the vertex for which the above mentioned criteria is satisfied. By definition, the value of the variable at the end of the $(j-1)$-th iteration is v_j. Thus, v_j does not have any incoming edge from vertices other than $v_1, v_2, \ldots, v_{j-1}$. Hence, having an edge from vertex u to vertex v_j will also be a contradiction. Therefore, there cannot be any path of length $(j-1)$ from vertex v_1 to vertex v_j other than $(v_1, v_2, \ldots, v_{j-1}, v_j)$.

Lemma 2 implies that whenever the function COUNT is called in line 10 of algorithm 1, it returns a value of d other than "?" along exactly one computation path. Any computation path where COUNT returns "?" is ignored (returns "?" in line 12). Only the computation path where COUNT returns a non "?" value proceeds further and can either accept or reject depending on the input. Thus, for all input instances, algorithm 1 accepts or rejects along exactly one computation path. All other computation paths are ignored (returns "?").

Remark 1. *"?"* is like a don't-care state. Roughly speaking, if we want an UL algorithm for our problem, we can consider *"?"* to be *reject*. If we want a coUL algorithm, we can consider *"?"* to be *accept*.

2.2 Correctness of Algorithm 1

In this subsection, we will argue for the correctness of algorithm 1. First, we will show that if the input DAG G has a Hamiltonian path, then algorithm 1 must accept. For this, we first state and prove the following lemma.

Lemma 3. *Let v_i be the value of the variable v at the beginning of the i-th iteration of the while loop (in line 7) of algorithm 1. If the input DAG G has a Hamiltonian path, then v_i must be the i-th vertex of the Hamiltonian path in G.*

Proof. We will prove this by induction.

The base case is easy to see. The Hamiltonian path in a DAG must start from the source vertex of the DAG, since the source vertex cannot occur at any other position in the Hamiltonian path. As we have already mentioned, v_1 is the source vertex of G since the variable v is initialized to be the (unique) source vertex of G (in line 5 of algorithm 1).

Let's assume that v_{i-1} is the $(i-1)$-th vertex of the Hamiltonian path in G. The value of register v at the beginning of $(i-1)$-th iteration is v_{i-1}. Let the i-th vertex of the Hamiltonian path be u. Therefore, there is an edge (v_{i-1}, u) from vertex v_{i-1} to u in G. Also, there cannot be any incoming edge to vertex u other than from vertices $v_1, v_2, \ldots, v_{i-1}$. This is because, if there is an edge to u from a vertex y that is subsequent to u in the Hamiltonian path, then it implies that there is a cycle $(u \ldots yu)$ in G, which will be a contradiction since G is promised to be acyclic. Moreover, there is only one vertex in G which has an incoming edge from v_{i-1} but has no incoming edge from any vertex other

than vertices $v_1, v_2, \ldots, v_{i-1}$. This is because the vertices $v_1, v_2, \ldots, v_{i-1}$ can't have an incoming edge from vertex v_{i-1} since otherwise there will be a cycle. Any vertex which comes after u in the Hamiltonian path in G must be preceded by a vertex other than $v_1, v_2, \ldots, v_{i-1}$ and hence, must have an incoming edge from at least one vertex other than $v_1, v_2, \ldots, v_{i-1}$. Thus, u is the only vertex in G that has an incoming edge from v_{i-1} and has no incoming edge from vertices other than $v_1, v_2, \ldots, v_{i-1}$.

The COUNT routine non-deterministically guesses a path of length $(i-2)$ from v_1 to v_{i-1}. During the process, it counts the total number of incoming edges to a vertex v' from any of the vertices in the guessed path.

The COUNT routine is used for each vertex that has an incoming edge from v_{i-1}. In this way, algorithm 1 (in lines 9-15) computes the number (m) of vertices that have an incoming edge from v_{i-1} but have no incoming edge from vertices other than $v_1, v_2, \ldots, v_{i-1}$. We have already seen that m must be exactly equal to one if G is Hamiltonian. Therefore, algorithm 1 in this case will move to line 18, set v to be u and proceed for the next iteration. Thus, the value of the variable v at the beginning of the i-th iteration, that is v_i, is equal to u. Hence, v_i is the i-th vertex of the Hamiltonian path in G.

Lemma 3 implies that if the input G to algorithm 1 has a Hamiltonian path, then the algorithm will run for n iterations. At the beginning of the i-th iteration, the value of the variable v will be set to the i-th vertex of the Hamiltonian path. If algorithm 1 successfully runs for n iterations, then it must accept the input at line 21 once it comes out of the loop.

Next, we show that if algorithm 1 accepts the input DAG G, then G must have a Hamiltonian path.

Lemma 4. *If algorithm 1 accepts, then there must be a Hamiltonian path in G.*

Proof. We can see that for all i, v_i has an incoming edge from v_{i-1}. Therefore, for all i, $(v_1, v_2, \ldots, v_i)$ constitutes a walk and since the graph G is acyclic, this walk is also a path. Algorithm 1 accepts only when the while loop (in line 7) has successfully iterated for n times. Hence, there exists a path $(v_1, v_2, \ldots, v_n)$ in G. This must be a Hamiltonian path since it is a path consisting of n vertices. Thus, there always exists a Hamiltonian path in G if algorithm 1 accepts.

Lemma 3 and lemma 4 taken together prove that algorithm 1 accepts if and only if the input DAG G has a Hamiltonian path.

Remark 2. Algorithm 1 as presented here seems to work correctly under the promise that the input graph is a DAG. Identifying whether a graph is acyclic or not is NL-complete. Hence, we don't know if the promise (of being acyclic) on the input can be independently checked in $\mathsf{UL} \cap \mathsf{coUL}$. However, if a graph which is not acyclic is provided as input, algorithm 1 will reject it. We can prove by induction that for any k, the vertex v_k is not a part of any cycle. v_1 is the source of the graph and hence not a part of any cycle (base case). Let us first assume that v_k is not a part of any cycle for any $k < i$ (induction hypothesis). Now,

algorithm 1 ensures that v_i has incoming edges only from vertices $v_1, v_2, \ldots, v_{i-1}$. If v_i is a part of a cycle, then it must have an edge from a vertex which is also a part of that cycle. However, none of $v_1, v_2, \ldots, v_{i-1}$ are part of any cycle (according to induction hypothesis). Therefore v_i also cannot be part of any cycle. If a graph that is not acyclic is provided as input to algorithm 1, then the vertices in the graph which are part of some cycles can never occur as v_k, for any k. Thus, the while loop at line 7 in algorithm 1 cannot run for n times and at some point, the input will be rejected at line 16-17 in algorithm 1. Hence, the set of traceable DAGs can be decided in $\mathsf{UL} \cap \mathsf{coUL}$ even without any promise on the input.

3 Some Consequences of our Result

In this section, we discuss two consequences of our main result.

3.1 A Parametrized Upper Bound on the Unambiguous Space Complexity of the LongPath Problem

In this subsection, we provide a parametrized upper bound on the unambiguous non-deterministic space complexity of the LongPath problem in DAGs using our $\mathsf{UL} \cap \mathsf{coUL}$ routine for the HamiltonianPath problem in DAGs. We state this bound in the following theorem.

Theorem 2. *Given a directed acyclic graph G having n vertices and two vertices s and t in G, the problem of deciding whether there exists a path of length at least $n - k$ from s to t is solvable in unambiguous (and co-unambiguous) non-deterministic $O(k \log n)$ space.*

Proof. We assume we are given an $\mathsf{UL} \cap \mathsf{coUL}$ routine for HamiltonianPath in DAGs theorem 1. That is, we have a non-deterministic logspace Turing machine that decides along exactly one computation path whether the given DAG has a Hamiltonian path or not. Now, consider one by one all subsets of vertices (other than s and t) of size at most k. For each such subset, eliminate the vertices in this subset from the graph and check whether the remaining subgraph has a Hamiltonian path from s to t using the $\mathsf{UL} \cap \mathsf{coUL}$ routine. If the answer is yes, then we accept since the Hamiltonian path in the subgraph is a path of length at least $n - k$ from s to t as the size of the subgraph is at least $n - k$. If the answer is no, then we move on to the next subset of vertices and continue. If the answer is no for all subsets of vertices, then we reject.

This algorithm works because if there is a path of length at least $n-k$ from s to t, then this path is a Hamiltonian path in the subgraph that does not contain the vertices (at most k in number) not in this path. That is, there exists a path of length at least $n - k$ from s to t if and only if there is a subset of (at most k) vertices, which when removed from the graph, the remaining subgraph (of size at least $n - k$) has a Hamiltonian path from s to t.

Remark 3. Note that unlike the UL bound for the Hamiltonian path problem in DAGs, the unambiguous space bound provided in theorem 2 is not a natural or obvious one. There can be several paths of length at least $n-k$ from s to t and hence, just non-deterministically guessing a path of length at least $n-k$ from s to t doesn't necessarily give an unambiguous algorithm. Our $\mathsf{UL} \cap \mathsf{coUL}$ routine for HamiltonianPath in DAGs is crucial for providing the unambiguous space bound in theorem 2.

3.2 Reachability in Traceable DAGs

In this subsection, we prove the following theorem.

Theorem 3. *Reachability in traceable DAGs is decidable in* $\mathsf{UL} \cap \mathsf{coUL}$.

Proof. Given a traceable DAG G and two vertices u and v in G, we need to decide whether u is reachable from v or not. Let s and t be the source and sink of G respectively. Since G is traceable, therefore there is a unique Hamiltonian path from s to t in G (by lemma 2). Now, our unambiguous (and co-unambiguous) routine to decide reachability is simple. Non-deterministically guess a walk (which is also a path because G is a DAG) of length n from s to t. There will be only one computation path which will correctly guess such a path since there is only one such path in G. Return "?" along the computation paths which don't make a correct guess (see remark 1). While making the guesses of which vertex to take next in the walk, keep track of which vertex out of u and v we visit first. In the computation path that correctly guesses the Hamiltonian path from s to t, if u is visited before v, then v is reachable from u and hence accept; else, v is not reachable from u and hence reject. This routine accepts or rejects along exactly one computation path. Therefore, this routine is unambiguous as well as co-unambiguous. Thus, reachability in traceable DAGs is decidable in $\mathsf{UL} \cap \mathsf{coUL}$.

4 Acknowledgments

This work has been partly funded by Research-I foundation.

References

1. Bourke, C., Tewari, R., Vinodchandran, N.V.: 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. pp. 217–221 (2007). https://doi.org/10.1109/CCC.2007.9, http://doi.ieeecomputersociety.org/10.1109/CCC.2007.9
2. Garey, M., Johnson, D., Stockmeyer, L.: Some simplified NP-complete graph problems. Theoretical Computer Science **1**(3), 237–267 (1976). https://doi.org/https://doi.org/10.1016/0304-3975(76)90059-1, https://www.sciencedirect.com/science/article/pii/0304397576900591

3. Immerman, N.: Nondeterministic space is closed under complement. SIAM Journal on Computing **17**, 935–938 (1988)
4. Itai, A., Papadimitriou, C.H., Szwarcfiter, J.L.: Hamilton paths in grid graphs. SIAM Journal on Computing **11**(4), 676–686 (1982). https://doi.org/10.1137/0211056
5. Kallampally, V.A.T., Tewari, R.: Trading determinism for time in space bounded computations. In: 41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016, August 22-26, 2016 - Kraków, Poland. pp. 10:1–10:13 (2016). https://doi.org/10.4230/LIPIcs.MFCS.2016.10, https://doi.org/10.4230/LIPIcs.MFCS.2016.10
6. Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W. (eds.) Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA. pp. 85–103. Plenum Press, New York (1972). https://doi.org/10.1007/978-1-4684-2001-2_9, https://doi.org/10.1007/978-1-4684-2001-2_9
7. Limaye, N., Mahajan, M., Nimbhorkar, P.: Longest paths in planar dags in unambiguous logspace. In: Proceedings of the Fifteenth Australasian Symposium on Computing: The Australasian Theory - Volume 94. p. 101–108. Australian Computer Society, Inc. (2009)
8. van Melkebeek, D., Prakriya, G.: Derandomizing isolation in space-bounded settings. SIAM Journal on Computing **48**(3), 979–1021 (2019). https://doi.org/10.1137/17M1130538
9. Reinhardt, K., Allender, E.: Making nondeterminism unambiguous. SIAM J. Comput. **29**(4), 1118–1131 (2000). https://doi.org/10.1137/S0097539798339041, http://dx.doi.org/10.1137/S0097539798339041
10. Szelepcsényi, R.: The method of forced enumeration for nondeterministic automata. Acta Informatica **26**, 279–284 (1988)
11. Thierauf, T., Wagner, F.: The isomorphism problem for planar 3-connected graphs is in unambiguous logspace. In: Albers, S., Weil, P. (eds.) STACS 2008, 25th Annual Symposium on Theoretical Aspects of Computer Science, Bordeaux, France, February 21-23, 2008, Proceedings. vol. 1, pp. 633–644. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany (2008). https://doi.org/10.4230/LIPICS.STACS.2008.1327, https://doi.org/10.4230/LIPIcs.STACS.2008.1327