A Logspace Approximation Scheme for the Shortest Path Problem for Graphs with Bounded Independence Number

Till Tantau*

International Computer Science Institute
1947 Center Street
Berkeley, CA 94704, USA
tantau@icsi.berkeley.edu

Abstract. How difficult is it to find a path between two vertices in finite directed graphs whose independence number is bounded by some constant k? The independence number of a graph is the largest number of vertices that can be picked such that there is no edge between any two of them. The complexity of this problem depends on the exact question we ask: Do we only wish to tell whether a path exists? Do we also wish to construct such a path? Are we required to construct the shortest path? Concerning the first question, it is known that the reachability problem is first-order definable for all k. In contrast, the corresponding reachability problems for many other types of finite graphs, including dags and trees, are not first-order definable. Concerning the second question, in this paper it is shown that not only can we construct paths in logarithmic space, but there even exists a logspace approximation scheme for this problem. It gets an additional input $r > 1$ and outputs a path that is at most r times as long as the shortest path. In contrast, for directed graphs, undirected graphs, and dags we cannot construct paths in logarithmic space (let alone approximate the shortest one), unless complexity class collapses occur. Concerning the third question, it is shown that even telling whether the shortest path has a certain length is NL-complete and thus as difficult as for arbitrary directed graphs.

1 Introduction

Finding paths in graphs is one of the most fundamental problems in graph theory. The problem has both practical and theoretical applications in many different areas. For such problems we are given a graph G and two vertices s and t, the *source* and the *target*, and we are asked to find a path from s to t. This problem comes in different versions: The most basic one is the *reachability problem*, which just asks whether such a path *exists*. This problem is also known as 'accessibility problem' or 's-t-connectivity problem'. The *construction problem*

* Supported by a postdoc research fellowship grant of the German academic exchange service (DAAD). Work done in part at the Technical University of Berlin.

asks us to *construct* a path, provided one exists. The *optimization problem* asks us to construct not just any path, but the *shortest* one. Closely related to the optimization problem is the *distance problem*, which asks us to decide whether the distance of s and t is bounded by a given number. If the optimization problem is difficult to solve, we can consider the *approximation problem*, which asks us to construct a path that is not necessarily a shortest path, but that is only a constant factor longer than the distance of s and t.

In this paper it is shown that for directed graphs whose independence number is bounded by some constant k the reachability problem, the construction problem, and the optimization problem have fundamentally different computational complexities. The paper extends a previous paper [18] that treated only the reachability problem. The main contribution of the present paper is a logspace approximation scheme for the optimization problem and a proof that the distance problem is NL-complete. This paper presents the first example of an optimization problem that cannot be solved optimally in logarithmic space (unless L = NL), but that can be approximated well in logarithm space. Approximation theory has traditionally focused on polynomial-time computations; mostly because approximation algorithm are typically only sought for if computing optimal solutions turns out to be NP-hard, but also because computing *any* solution and computing an *optimal* solution seemed to have the same complexity for the problems considered in small space complexity theory.

The *independence number* $\alpha(G)$ of a graph G is the maximum number of vertices that can be picked from G such that there is no edge between any two of these vertices. The most prominent examples of graphs with bounded independence number are *tournaments* [17, 20], which are directed graphs with exactly one edge between any two vertices. Their independence number is 1. The reachability problem for tournaments arises naturally if we try to rank or sort objects according to a comparison relation that tells us for any two objects which 'beats' the other, but that is not necessarily acyclic.

A different example of graphs with bounded independence number, studied in [5], are directed graphs $G = (V, E)$ whose underlying undirected graph is claw-free, i. e., does not contain the $K_{1,m}$ for some constant m, and whose minimum degree is at least $|V|/3$. Their independence number is at most $3m - 3$.

To get an intuition on the behaviour of the independence number function, first note that independence is a monotone graph property: adding edges to a graph can only increase, deleting only decrease the independence number. Given two graphs with the same vertex set and independence numbers α and α', the independence number of their union is at most the minimum of α and α' and the independence number of their disjoint union is $\alpha + \alpha'$. Thus if a graph consists of, say, four disjoint tournaments with arbitrary additional edges connecting these tournaments, its independence number would be at most 4. Intuitively, a graph with a low independence number must have numerous edges and, indeed, at least $\binom{n}{2} \big/ \binom{\alpha(G)+1}{2}$ edges must be present in any n-vertex graph G. This abundance of edges might suggest that if paths between two given vertices exist, there should also exist a short path between them. While this is true for the undirected case, in the directed case (which we are interested in in this paper) the distance between two vertices can become as large as $n-1$ even in n-vertex tournaments.

1.1 How Difficult Is It to Tell Whether a Path Exists?

The reachability problem for finite directed graphs, which will be denoted REACH in the following, is well-known to be NL-complete [11, 12] and thus easy from a computational point of view. The complexity of the reachability problem drops if we restrict the type of graphs for which we try to solve it. The reachability problem REACH_{u} for finite undirected graphs is SL-complete [14] and thus presumably easier to solve. The even more restricted problem $\text{REACH}_{\text{forest}}$ for undirected forests and the problem $\text{REACH}_{\text{out}\leq 1}$ for directed graphs in which all vertices have out-degree at most 1 are L-complete [4]. Here and in the following 'completeness' always refers to completeness with respect to the restrictive $\leq_{\text{m}}^{\text{AC}^0}$-reductions, i. e., many-to-one reductions that can be computed by a family of logspace-uniform constant-depth circuits with unbounded fan-in and fan-out [2, 3].

The complexity of the reachability problem for finite directed graphs whose independence number is bounded by a constant k is much lower: somewhat surprisingly, this problem is first-order definable for all k, as shown in [18]. Formally, for each k the language $\text{REACH}_{\alpha\leq k} := \text{REACH} \cap \{\langle G, s, t\rangle \mid \alpha(G) \leq k\}$ is first-order definable, where $\langle\,\rangle$ denotes a standard binary encoding. Languages whose descriptive complexity is first-order are known to be very simple from a computational point of view. They can be decided by a family of logspace-uniform AC^0-circuits [15], in constant parallel time on concurrent-read, concurrent-write parallel random access machines (CRCW-PRAMs) [15], and in logarithmic space. Since it is known that L-hard sets cannot be first-order definable [1, 6], $\text{REACH}_{\alpha\leq k}$ is *unconditionally* easier to solve than REACH, REACH_{u}, and $\text{REACH}_{\text{forest}}$.

When studying the complexity of a graph problem, one usually assumes (as done above) that the input graph is encoded as a binary string 'in some standardized way'. Which particular way of encoding is chosen is of little or no concern for the computational complexity of the problem. This is no longer true if the input graphs are encoded *succinctly*, as is often the case for instance in hardware design. Succinctly represented graphs are given indirectly via a program or a circuit that decides the edge relation of the graph. Papadimitriou, Yannakakis, and Wagner [19, 23, 24] have shown that the problems SUCCINCT-REACH, $\text{SUCCINCT-REACH}_{\text{u}}$, $\text{SUCCINCT-REACH}_{\text{forest}}$, and $\text{SUCCINCT-REACH}_{\text{out}\leq 1}$ are all PSPACE-complete. Opposed to this, $\text{SUCCINCT-REACH}_{\alpha\leq k}$ is Π_2^{P}-complete for all k, see [18] once more.

1.2 How Difficult Is It To Construct a Path?

The low complexity of the reachability problem seemingly settles the complexity of finding paths in graphs with bounded independence number. At first sight, the path construction problem appears to reduce to the reachability problem via a simple algorithm: Starting at the source vertex, for each successor of the current vertex check whether we can reach the target from it (for at least one successor this test will be true); make that successor the current vertex; and repeat until we have reached the target. Unfortunately, this algorithm is flawed since it can lead us around in endless cycles for graphs that are not acyclic. A correct algorithm does not move to any successor, but to the successor that is *nearest* to the target. This corrected algorithm does not only produce *some* path, but the shortest one. However, the algorithm now needs to compute the distance between two vertices internally, which is conceptually a more difficult problem than deciding whether two vertices are connected.

Nevertheless, we shall see that a path between any two connected vertices can be constructed in logarithmic space in graphs with bounded independence number. There even exists a *logspace approximation scheme* for this problem. This means that for each $r > 1$ and each k there exists a logspace-computable function that maps an input $\langle G, s, t\rangle$ with $\alpha(G) \leq k$ to a path from s to t of length at most r times the distance of s and t. If no path exists, the function outputs 'no path exists'.

1.3 How Difficult Is It To Construct the Shortest Path?

How difficult is it to construct the shortest path in a graph with bounded independence number? We show that, again surprisingly, even for tournaments this problem is as difficult as constructing the shortest path in an arbitrary graph. As pointed out above, the complexity of constructing the shortest path hinges on the complexity of the *distance problem* $\text{DISTANCE}_{\text{tourn}} := \big\{\langle G, s, t, d\rangle \mid G$ is a tournament in which there is a path from s to t of length at most $d\big\}$. This problem is shown to be NL-complete. Thus DISTANCE and $\text{DISTANCE}_{\text{tourn}}$ are $\leq_{\text{m}}^{\text{AC}^0}$-equivalent, but REACH and $\text{REACH}_{\text{tourn}}$ are not. The succinct version of $\text{DISTANCE}_{\text{tourn}}$ is shown to be PSPACE-complete.

1.4 Organization of This Paper

This paper is organized as follows. In Section 2 graph-theoretic terminology and known results on the reachability problem for graphs with bounded independence number are reviewed. In Section 3 a logspace approximation scheme for the shortest path problem for graphs with bounded independence number is presented. In Section 4 the distance problem for tournaments is shown to be NL-complete and its succinct version is shown to be PSPACE-complete.

2 Review of Known Results

In this section graph-theoretic terminology and known results on the reachability problem in graphs with bounded independence number are reviewed.

A *(directed) graph* is a nonempty set V of vertices together with a set $E \subseteq V \times V$ of directed edges. A graph is *undirected* if its edge relation is symmetric. A *tournament* is a graph with exactly one edge between any two different vertices and $(v, v) \notin E$ for all $v \in V$. A *forest* is an undirected, acyclic graph. A *tree* is a connected forest.

A *path of length* ℓ in a graph $G = (V, E)$ is a sequence $(v_0, \ldots, v_\ell)$ of distinct vertices with $(v_i, v_{i+1}) \in E$ for $i \in \{0, \ldots, \ell - 1\}$. A vertex t is *reachable* from a vertex s if there is a path from s to t. The *distance* $\mathrm{d}(s, t)$ of two vertices is the length of the shortest path between them or ∞, if no path exists. For $i \in \mathbb{N}$, a vertex $u \in V$ is said to *i-dominate* a vertex $v \in V$ if there is a path from u to v of length at most i. A set $U \subseteq V$ is an *i-dominating set for G* if every vertex $v \in V$ is i-dominated by some vertex $u \in U$. The *i-domination number* $\beta_i(G)$ is the minimal size of an i-dominating set for G. A set $U \subseteq V$ is an *independent set* if there is no edge in E connecting vertices in U. The maximal size of independent sets in G is its *independence number* $\alpha(G)$.

Fact 2.1 ([18]). *Let $G = (V, E)$ be a finite graph with at least two vertices, $n := |V|$, $\alpha := \alpha(G)$, and $c := (\alpha^2 + \alpha)/(\alpha^2 + \alpha - 1)$. Then*

1. $\beta_1(G) \leq \lceil \log_c n \rceil$ *and*
2. $\beta_2(G) \leq \alpha$.

For tournaments G, Fact 2.1 yields $\beta_1(G) \leq \lceil \log_2 n \rceil$ and $\beta_2(G) = 1$. The first result was first proved by Megiddo and Vishkin in [16], where it was used to show that the dominating set problem for tournaments is not NP-complete, unless $\mathrm{NP} \subseteq \mathrm{DTIME}\big[n^{O(\log n)}\big]$. The second result is also known as the Lion King Lemma, which was first noticed by Landau [13] in the study of animal societies, where the dominance relations on prides of lions form tournaments. It has applications in the study of P-selective sets [9] and many other fields.

The next fact states that the complexity of the reachability problem for graphs with bounded independence number is low: $\mathrm{REACH}_{\alpha \leq k}$ is first-order definable for all k. *First-order definability* is a language property studied in descriptive complexity theory. It can be defined as follows for the special case of languages $A \subseteq \{\langle V, E, s, t \rangle \mid (V, E) \text{ is a finite graph, } s, t \in V\}$: Let $\tau = (\mathrm{E}^2, \mathrm{s}, \mathrm{t})$ be the *signature of graphs with two designated vertices.* A *first-order τ-formula* is a first-order formula that contains, other than quantifiers, variables, and connectives, only the binary relation symbol E and the constant symbols s and t. An example is the formula $\exists x \big[\mathrm{E}(\mathrm{s}, x) \wedge \mathrm{E}(x, \mathrm{t})\big]$. A *$\tau$-structure* is a tuple (V, E, s, t) consisting of a graph (V, E) and two vertices $s, t \in V$. A τ-structure is a *model* of a τ-formula if the formula holds when we interpret the relation symbol E as the edge relation E and the constant symbols s and t as the vertices s and t. For example, the τ-formula $\exists x \big[\mathrm{E}(\mathrm{s}, x) \wedge \mathrm{E}(x, \mathrm{t})\big]$ is a model of every τ-structure (V, E, s, t) in which there is a path from s to t in the graph (V, E) of length exactly 2. The language A is *first-order definable* if there exists a τ-formula ϕ such that $\langle V, E, s, t \rangle \in A$ iff (V, E, s, t) is a model of ϕ.

Fact 2.2 ([18]). *For each k, $\mathrm{REACH}_{\alpha \leq k}$ is first-order definable.*

The complexity of the reachability problem for graphs with bounded independence number is also interesting in the succinct setting. Succinctly represented graphs are given implicitly via a description in some description language. Since succinct representations allow the encoding of large graphs into small codes, numerous graph properties are (provably) harder to check for succinctly represented graphs than for graphs coded in the usual way. Papadimitriou et al. [19, 24] have shown that most interesting problems for succinctly represented graphs are PSPACE-complete or even NEXP-complete. The following formalization of succinct graph representations follows Galperin and Wigderson [7], but others are also possible [24, 8].

Definition 2.1. *A* succinct representation *of a graph* $G = \big(\{0,1\}^n, E\big)$ *is a* $2n$*-input circuit* C *such that for all* $u, v \in \{0,1\}^n$ *we have* $(u,v) \in E$ *iff* $C(uv) = 1$*.*

The circuit tells us for any two vertices of the graph whether there is a directed edge between them or not. Note that C will have size at least $2n$ since it has $2n$ input gates.

Definition 2.2. *Let* $A \subseteq \{\langle G, s, t\rangle \mid G = (V,E)$ *is a finite graph,* $s,t \in V\}$*. Then* SUCCINCT-A *is the set of all codes* $\langle C, s, t\rangle$ *such that* C *is a succinct representation of a graph* G *with* $\langle G, s, t\rangle \in A$*.*

Fact 2.3 ([18]). *For each* k*,* SUCCINCT-$\text{REACH}_{\alpha \le k}$ *is* Π_2^{P}*-complete.*

3 Complexity of the Approximation Problem

In this section it is shown that for graphs with bounded independence number we can not only tell in logarithmic space whether a path exists between two vertices, but we can also construct such a path. While it seems difficult to construct the *shortest* path in logarithmic space (by the results of the next section this is impossible unless L = NL), it is possible to find a path that is *approximately* as long as the shortest path. Even better, there exists a *logspace approximation scheme* for constructing paths whose length is as close to the length of the shortest path as we would like:

Theorem 3.1. *For all* k *there exists a deterministic Turing machine* M *with read-only access to the input tape and write-only access to the output tape such that:*

1. *On input* $\langle G, s, t, m\rangle$ *with* $\langle G, s, t\rangle \in \text{REACH}_{\alpha \le k}$ *and* $m \ge 1$*, it outputs a path from* s *to* t *of length at most* $(1 + 1/m)\,\mathrm{d}(s,t)$*.*
2. *On input* $\langle G, s, t, m\rangle$ *with* $\langle G, s, t\rangle \notin \text{REACH}_{\alpha \le k}$ *it outputs 'no path exists'.*
3. *It uses space* $O(\log m \log n)$ *on the work tapes, where* n *is the number of vertices in* G*.*

For the proof of the theorem we need two lemmas. The second lemma is a 'constructive version' of Savitch's theorem [21].

Lemma 3.2. *There exists a function in* FL *that maps every input* $\langle G, s, t\rangle \in \text{REACH}_{\text{forest}}$ *to the shortest path from* s *to* t *in* G *and all other inputs to 'no path exists'.*

Proof. The problem $\text{REACH}_{\text{forest}}$ is L-complete as shown in [4]. In order to compute the shortest path from s to t we iterate the following procedure, starting at s: For each neighbour v of the current vertex, we check whether t is reachable from v in the forest obtained by removing the edge connecting the current vertex and v. There is exactly one vertex for which this test succeeds. We output this vertex, make it the new current vertex, and repeat the procedure until we reach t. □

Lemma 3.3. *There exists a deterministic Turing machine M with read-only access to the input tape and write-only access to the output tape such that:*

1. *On input $\langle G, s, t\rangle \in$ REACH it outputs a shortest path from s to t and uses space $O\big(\log \mathrm{d}(s,t) \log n\big)$ on the work tapes, where n is the number of vertices in G.*
2. *On input $\langle G, s, t\rangle \notin$ REACH it outputs 'no path exists'. It uses space $O(\log^2 n)$ on the work tapes, where n is the number of vertices in G.*

Proof. We augment Savitch's algorithm [21] by a construction procedure that outputs paths. If there are several paths, the procedure 'decides on one of them' and does so 'within the recursion'.

Let *reachable*(u, v, ℓ) be Savitch's procedure for testing whether there is a path from u to v of length at most ℓ: For $\ell = 1$, it checks whether $(u, v) \in E$ or $u = v$. For larger ℓ, it checks whether for some vertex z both the calls *reachable*$(u, z, \lfloor \ell/2 \rfloor)$ and *reachable*$(z, v, \ell - \lfloor \ell/2 \rfloor)$ succeed. As noted by Savitch, we can compute *reachable*(u, v, ℓ) in space $O(\log \ell \log n)$ since we can reuse space.

We next define a procedure *construct*(u, v, ℓ) that writes a path of length ℓ from u to v onto an output tape, provided *reachable*(u, v, ℓ) holds. In order to simplify the assemblage of outputs of different calls to *construct*, the last vertex of the path, i. e., the vertex v, will be omitted. For $\ell = 1$, *construct* simply outputs u. For larger ℓ, it finds the first vertex z for which both the calls *reachable*$(u, z, \lfloor \ell/2 \rfloor)$ and *reachable*$(z, v, \ell - \lfloor \ell/2 \rfloor)$ succeed. For this vertex z it first calls *construct*$(u, z, \lfloor \ell/2 \rfloor)$ and then *construct*$(z, v, \ell - \lfloor \ell/2 \rfloor)$.

The machine M iteratively calls *reachable*(s, t, ℓ) for increasing values of ℓ. For the first value ℓ for which this test succeeds, it calls *construct*(s, t, ℓ), appends the missing vertex t, and quits. If the tests do not succeed for any $\ell \leq n$, it outputs 'no path exists'. □

Proof (of Theorem 3.1). Let an input $\langle G, s, t, m\rangle$ be given. Let $G = (V, E)$ and $n := |V|$. For a set U of vertices let $\mathrm{d}(U, t) := \min\{\mathrm{d}(u, t) \mid u \in U\}$.

We first check, in space $O(\log n)$, whether $\langle G, s, t\rangle \in \text{REACH}_{\alpha \leq k}$ holds and output 'no path exists' if this is not the case. Otherwise we enter a loop in which we construct a sequence $U_1, U_2, \dots, U_\ell \subseteq V$ of vertex sets with $U_1 = \{s\}$ and $U_\ell = \{t\}$. For the construction of U_{i+1} we access only U_i and use space $O(\log m \log n)$. Once we have constructed U_{i+1} we erase U_i and reuse the space it occupied.

The set U_i is obtained from U_{i-1} as follows: If $\mathrm{d}(U_{i-1}, t) \leq 2m + 1$, let $U_i := \{t\}$. Otherwise let $S_i := \{v \in V \mid \mathrm{d}(U_{i-1}, v) = 2m + 2\}$ and choose $U_i \subseteq S_i$

as a 2-dominating, size-k vertex subset the graph $G' := \big(S_i, E \cap (S_i \times S_i)\big)$ induced on the vertices in S_i. Since $\alpha(G') \leq \alpha(G) \leq k$, such a 2-dominating set U_i exists by Fact 2.1. We can obtain it in space $O(\log m \log n)$ since the question '$v \in S_i$?' can be answered in space $O(\log m \log n)$ using the procedure *reachable* from Lemma 3.3.

The sets U_i have the following properties for $i \in \{2, \dots, \ell - 1\}$:

1. All elements of U_i are reachable from s.
2. $|U_i| \leq k$.
3. $\mathrm{d}(U_{i-1}, u) = 2m + 2$ for all $u \in U_i$.
4. $\mathrm{d}(U_i, t) \leq \mathrm{d}(U_{i-1}, t) - 2m$ and hence $\mathrm{d}(U_i, t) \leq \mathrm{d}(s, t) - 2m(i - 1)$.

To see that the last property holds, note that $\mathrm{d}(U_i, t) \leq \mathrm{d}(S_i, t) + 2$ and that $\mathrm{d}(S_i, t) = \mathrm{d}(U_{i-1}, t) - 2m - 2$. For $i = \ell$, the first two properties are also true and the third one becomes $\mathrm{d}(U_{i-1}, t) \leq 2m + 1$.

Intuitively, in each iteration we reduce the distance between U_i and t by at least $2m$ and each U_{i-1} can be connected to the next U_i by a path of length $2m + 2$. It remains to explain how to connect the U_i's correctly.

In order to output the desired path from s to t of length at most $(1 + 1/m)\,\mathrm{d}(s, t)$, we first construct a forest that contains this path. The forest is not actually written down anywhere (we are allowed only a logarithmic amount of space). Rather, as in the proof of FL being closed under composition, the forest's code is dynamically recalculated in space $O(\log m \log n)$ whenever one of its bits is needed. Finding the shortest path in a forest can be done in logarithmic space by Lemma 3.2, and the shortest path in the forest will be the desired path.

To define the forest F, for each $i \in \{2, \dots, \ell\}$ we first define a 'small' forest F_i as follows: For each $u \in U_i$ it contains the vertices and edges of the shortest path from U_{i-1} to u. This path is constructed by calling the machine M from Lemma 3.3 on input $\langle G, u', u \rangle$ for the first vertex $u' \in U_{i-1}$ for which $\mathrm{d}(u', u)$ is minimal. Since $\mathrm{d}(u', u) \leq 2m + 2$, this call needs space $O(\log m \log n)$. The graph F_i is, indeed, a forest since if two paths output by M for the same source vertex split at some point, they split permanently. Let F be the union of all the forests F_i constructed during the run of the algorithm. This union is a forest since every tree in a forest F_i has at most one vertex in common with any other tree in a forest F_j with $j \neq i$.

Consider the shortest path from s to t in the forest F. This path passes through all U_i. For $i \in \{1, \dots, \ell\}$ let $u_i \in U_i$ be the last vertex of U_i on this path. The total length of the path is given by $\sum_{i=1}^{\ell-1} \mathrm{d}(u_i, u_{i+1})$. We have $\mathrm{d}(u_i, u_{i+1}) = 2m + 2$ for $i \in \{1, \dots, \ell - 2\}$. Thus the total length is

$$\begin{aligned}
(2m+2)(\ell-2) + \mathrm{d}(u_{\ell-1}, t) &= (2m+2)(\ell-2) + \mathrm{d}(U_{\ell-1}, t) \\
&\leq (2m+2)(\ell-2) + \mathrm{d}(s,t) - 2m(\ell-2) \\
&= \mathrm{d}(s,t) + 2(\ell-2) \leq \mathrm{d}(s,t) + \mathrm{d}(s,t)/m.
\end{aligned}$$

For the two inequalities, we both times used the last property of $U_{\ell-1}$, by which $\mathrm{d}(U_{\ell-1}, t) \leq \mathrm{d}(s, t) - 2m(\ell - 2)$ and hence also $2(\ell - 2) \leq \mathrm{d}(s, t)/m$. □

The space bound from Theorem 3.1 is optimal in the following sense: Suppose we could construct a machine M' that uses space $O(\log^{1-\epsilon} m \log n)$ and achieves the same as M. Then $\text{DISTANCE}_{\text{tourn}} \in \text{DSPACE}[\log^{2-\epsilon} n]$, because M' outputs the *shortest* path for $m = n + 1$. The results of the next section show that this would imply $\text{NL} \subseteq \text{DSPACE}[\log^{2-\epsilon} n]$.

4 Complexity of the Distance Problem

In this section we study the complexity of the distance problem for graphs with bounded independence number. This problem asks us to decide whether the distance of two vertices in a graph is smaller than a given input number. It is shown that this problem is NL-complete even for tournaments and that the succinct version is PSPACE-complete.

The distance problem is closely linked to the problem of constructing the shortest path in a graph: As argued in the introduction, we can *construct* the shortest path in graph if we have oracle access to the distance problem for this graph. The other way round, we can easily solve the distance problem if we have oracle access to an algorithm that constructs shortest paths. Because of this close relationship, the completeness result bashes any hope of finding a logspace algorithm for constructing shortest path in tournaments, unless L = NL.

Theorem 4.1. *The problem* $\text{DISTANCE}_{\text{tourn}}$ *is* NL*-complete.*

Proof. We show $\text{REACH} \leq_{\text{m}}^{\text{AC}^0} \text{DISTANCE}_{\text{tourn}}$. Let an input $\langle G, s, t\rangle$ be given. Let $G = (V, E)$ and $n := |V|$. The tournament $G' = (V', E')$ is constructed as follows: The vertex set V' is $\{1, \dots, n\} \times V$. We can think of this vertex set as a grid consisting of n rows and n columns. There is an edge in G' from a vertex (r_1, v_1) to a vertex (r_2, v_2) iff one of the following conditions holds:

1. $r_2 = r_1 + 1$ and $(v_1, v_2) \in E \cup \{(v, v) \mid v \in V\}$, i. e., if v_1 and v_2 are connected in G or if $v_1 = v_2$, then there is an edge leading 'downward' between them on adjacent rows.
2. $r_1 = r_2$ and $v_1 < v_2$, where $<$ is some linear ordering on V, i. e., the vertices on the same row are ordered linearly.
3. $r_2 = r_1 - 1$ and $(v_1, v_2) \notin E \cup \{(v, v) \mid v \in V\}$, i. e., if v_1 and v_2 are not connected in G and if they are not identical, then there is an edge leading 'upward' between them on adjacent rows.
4. $r_2 \leq r_1 - 2$, i. e., all edges spanning at least two rows point 'upward'.

The reduction machine poses the query 'Is there a path from $s' = (1, s)$ to $t' = (n, t)$ in G' of length at most $n - 1$?' Clearly this query can be computed by a logspace-uniform family of AC^0-circuits.

To see that this reduction works, first assume that there exists a path from s to t in G of length $m \leq n - 1$. Let $(s, v_2, \dots, v_m, t)$ be this path. Then $\big((1, s), (2, v_2), \dots, (m, v_m), (m+1, t), \dots, (n, t)\big)$ is a path in G' of length $n-1$. Second, assume that there exists a path from s' to t' in G' of length $m \leq n - 1$. Then

$m = n - 1$ since any path from the first row to the last row must 'brave all rows'—there are no edges that allow us to skip a row. Let $(v'_1, \dots, v'_n)$ be this path. Then $v'_i = (i, v_i)$ for some vertices $v_i \in V$. The sequence $(v_1, \dots, v_n)$ is 'almost' a path from s to t in G: For each $i \in \{1, \dots, n-1\}$ we either have $v_i = v_{i+1}$ or $(v_i, v_{i+1}) \in E$. Thus, by removing consecutive duplicates and loops, we obtain a path from s to t in G. □

By the above theorem, DISTANCE and $\text{DISTANCE}_{\text{tourn}}$ are $\leq_{\text{m}}^{\text{AC}^0}$-equivalent, while REACH and $\text{REACH}_{\text{tourn}}$ are not. The 'complexity jump' from $\text{REACH}_{\text{tourn}}$ to $\text{DISTANCE}_{\text{tourn}}$ is reflected by a similar jump for the succinct versions.

Definition 4.2. *Let* $\text{SUCCINCT-DISTANCE}_{\text{tourn}}$ *denote the language that contains all coded tuples* $\langle C, s, t, d\rangle$*, where* C *is a circuit,* s *and* t *are bitstrings, and* d *is a positive integer, such that* C *is a succinct representation of a graph* G *with* $\langle G, s, t, d\rangle \in \text{DISTANCE}_{\text{tourn}}$*.*

Theorem 4.3. $\text{SUCCINCT-DISTANCE}_{\text{tourn}}$ *is* PSPACE*-complete.*

Proof. Since $\text{DISTANCE}_{\text{tourn}} \in \text{NL}$, we have

$$\text{SUCCINCT-DISTANCE}_{\text{tourn}} \in \text{NPSPACE} = \text{PSPACE}.$$

For the hardness, let $A \in \text{PSPACE}$ be an arbitrary language and let M be a polynomial-space machine that accepts A. We show that A is $\leq_{\text{m}}^{\text{AC}^0}$-reducible to $\text{SUCCINCT-DISTANCE}_{\text{tourn}}$. For an input x, let G denote the configuration graph of M on input x, let s be the initial configuration, let t be the (unique) accepting configuration, and let d be an (exponential) bound on the running time of M on input x. Let G' be the tournament constructed in Theorem 4.1 and let C be an appropriate circuit that represents G'. Then $x \in A$ iff $\langle C, s, t, d\rangle \in \text{SUCCINCT-DISTANCE}_{\text{tourn}}$.

The representing circuit C can be constructed by a logspace-uniform family of AC^0-circuits. To see this, first note that the circuit C can easily be constructed in logarithmic space since G' is highly structured. For an appropriate construction, C will depend on x only in a very limited way: For each bit of x there is a constant gate in C that 'feeds' this bit to the rest of the circuit, which does not depend on x at all. Thus we can hardwire almost all of C into the AC^0-circuit that computes it, only C's constant gates must be setup depending on x. □

5 Conclusion

The results of this paper extend the answer to the question 'How difficult is it to find paths in graphs with bounded independence number?' in two different ways. It was previously known that checking whether a path *exists* in a given graph can be done using AC^0-circuits. In this paper it was shown that *constructing* a path between two vertices can be done in logarithmic space. Constructing the *shortest* path in logarithmic space was shown to be impossible, unless $\text{L} = \text{NL}$.

These results settle the approximability of the (logspace) optimization problem 'shortest paths in graphs with bounded independence number'. This minimization problem cannot be solved exactly in logarithmic space (unless L = NL), but it can be approximated well: there exists a logspace approximation scheme for it. As we saw, the space $O(\log m \log n)$ needed by the scheme for a desired approximation ratio of $1 + 1/m$ is essentially optimal—any approximation scheme that does substantially better could be used to show unlikely inclusions like $\mathrm{NL} \subseteq \mathrm{DSPACE}[\log^{2-\epsilon} n]$. Thus it seems appropriate to call the scheme a '*fully* logspace approximation scheme' in analogy to 'fully polynomial-time approximation schemes'.

The shortest path problem for tournaments is not the only logspace optimization problem with surprising properties: In [22] it is shown that the distance problem for *undirected* graphs is also NL-complete, while the reachability problem is SL-complete. On the other hand, the distance problem for directed graphs is just as hard as the reachability problem for directed graphs. This shows that, just as in the polynomial-time setting, logspace optimization problems can have different approximation properties, although their underlying decision problems have the same complexity.

References

1. M. Ajtai. Σ_1^1 formulae on finite structures. *Annals of Pure and Applied Logic*, 24:1–48, 1983.
2. A. Chandra, L. Stockmeyer, and U. Vishkin. Constant depth reducibilities. *SIAM J. Comput.*, 13(2):423–439, 1984.
3. S. Cook. A taxonomy of problems with fast parallel algorithms. *Inform. Control*, 64(1–3):2–22, 1985.
4. S. Cook and P. McKenzie. Problems complete for deterministic logarithmic space. *J. Algorithms*, 8(3):385–394, 1987.
5. R. Faudree, R. Gould, L. Lesniak, and T. Lindquester. Generalized degree conditions for graphs with bounded independence number. *J. Graph Theory*, 19(3):397–409, 1995.
6. M. Furst, J. Saxe, and M. Sipser. Parity, circuits, and the polynomial-time hierarchy. *Math. Systems Theory*, 17(1):13–27, 1984.
7. H. Galperin and A. Wigderson. Succinct representations of graphs. *Inform. Control*, 56(3):183–198, 1983.
8. G. Gottlob, N. Leone, and H. Veith. Succinctness as a source of complexity in logical formalisms. *Annals of Pure and Applied Logic*, 97:231–260, 1999.
9. L. Hemaspaandra and L. Torenvliet. Optimal advice. *Theoretical Comput. Sci.*, 154(2):367–377, 1996.
10. N. Immerman. *Descriptive Complexity*. Springer-Verlag, 1998.
11. N. Jones. Space-bounded reducibility among combinatorial problems. *J. Comput. Syst. Sci.*, 11(1):68–85, 1975.
12. N. Jones, Y. Lien, and W. Laaser. New problems complete for nondeterministic log space. *Math. Systems Theory*, 10:1–17, 1976.
13. H. Landau. On dominance relations and the structure of animal societies, III: The condition for secure structure. *Bull. Mathematical Biophysics*, 15(2):143–148, 1953.

14. H. Lewis and C. Papadimitriou. Symmetric space-bounded computation. *Theoretical Comput. Sci.*, 19(2):161–187, 1982.
15. S. Lindell. A purely logical characterization of circuit uniformity. In *Proc. 7th Struc. in Complexity Theory Conf.*, pages 185–192, 1992. IEEE Computer Society Press.
16. N. Megiddo and U. Vishkin. On finding a minimum dominating set in a tournament. *Theoretical Comput. Sci.*, 61:307–316, 1988.
17. J. Moon. *Topics on Tournaments.* Holt, Rinehart, and Winston, 1968.
18. A. Nickelsen and T. Tantau. On reachability in graphs with bounded independence number. In O. H. Ibarra and L. Zhang, editors, *Proc. 8th Annual International Computing and Combinatorics Conf.*, volume 2387 of *Lecture Notes on Comput. Sci.*, pages 554–563. Springer-Verlag, 2002.
19. C. Papadimitriou and M. Yannakakis. A note on succinct representations of graphs. *Inform. Control*, 71(3):181–185, 1986.
20. K. Reid and L. Beineke. *Selected Topics in Graph Theory*, chapter Tournaments, pages 169–204. Academic Press, 1978.
21. W. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4(2):177–192, 1970.
22. T. Tantau. Logspace optimisation problems and their approximation properties. Technical report TR03-077, Electronic Colloquium on Computational Complexity, 2003.
23. K. Wagner. The complexity of problems concerning graphs with regularities. In *Proc. 7th Symposium on Math. Foundations of Comput. Sci.*, volume 176 of *Lecture Notes in Comput. Sci.*, pages 544–552. Springer-Verlag, 1984.
24. K. Wagner. The complexity of combinatorial problems with succinct input representation. *Acta Informatica*, 23(3):325–356, 1986.