Abstract

We consider directed graph algorithms in a streaming setting, focusing on problems concerning orderings of the vertices. This includes such fundamental problems as topological sorting and acyclicity testing. We also study the related problems of finding a minimum feedback arc set (edges whose removal yields an acyclic graph), and finding a sink vertex. We are interested in both adversarially-ordered and randomly-ordered streams. For arbitrary input graphs with edges ordered adversarially, we show that most of these problems have high space complexity, precluding sublinear-space solutions. Some lower bounds also apply when the stream is randomly ordered: e.g., in our most technical result we show that testing acyclicity in the $p$-pass random-order model requires roughly $n^{1+1/p}$ space. For other problems, random ordering can make a dramatic difference: e.g., it is possible to find a sink in an acyclic tournament in the one-pass random-order model using polylog($n$) space whereas under adversarial ordering roughly $n^{1/p}$ space is necessary and sufficient given $\Theta(p)$ passes. We also design sublinear algorithms for the feedback arc set problem in tournament graphs; for random graphs; and for randomly ordered streams. In some cases, we give lower bounds establishing that our algorithms are essentially space-optimal. Together, our results complement the much maturer body of work on algorithms for undirected graph streams.

1 Introduction

While there has been a large body of work on undirected graphs in the data stream model [20], the complexity of processing directed graphs (digraphs) in this model is relatively unexplored. The handful of exceptions include multipass algorithms emulating random walks in directed graphs [15,22], establishing prohibitive space lower bounds on finding sinks [13] and answering reachability queries [11], and ruling out semi-streaming constant-pass algorithms for directed reachability [12]. This is rather unfortunate given that many of the massive graphs often mentioned in the context of motivating work on graph streaming are directed, e.g., hyperlinks, citations, and Twitter “follows” all correspond to directed edges.

In this paper we consider the complexity of a variety of fundamental problems related to vertex ordering in directed graphs. For example, one basic problem that motivated much of this work is as follows: given a stream consisting of edges of an acyclic graph in an arbitrary order, how much memory is required to return a topological ordering of the graph? In the offline setting, this can be computed in $O(m+n)$ time using Kahn’s algorithm [16] or via depth-first trees [23] but nothing was known in the data stream setting.

We also consider the related minimum feedback arc set problem, i.e., estimating the minimum number of edges (arcs) that need to be removed such that the resulting graph is acyclic. This problem is NP-hard and the best known approximation factor is $O(\log n \log \log n)$ for arbitrary graphs [10], although a PTAS is known in the case of tournaments [18]. Again, nothing was known in the data stream model. In contrast, the analogous problem for undirected graphs is well understood in the data stream model. The number of edges required to make an undirected graph acyclic is $m - n + c$ where $c$ is the number of connected components. The number of connected components can be computed in $O(n \log n)$ space by constructing a spanning forest [2,11].

Previous Work. Some versions of the problems we study in this work have been considered previously in the query complexity model. For example, Huang et al. [14] consider the “generalized sorting problem” where $G$ is an acyclic graph with a unique topological order. The algorithm is presented with an undirected version of this graph and may query any edge to reveal its direction. The goal is to learn the topological ordering with the minimum number of queries. Huang et al. [14] and Angelov et al. [5] also studied the average case...
complexity of various problems where the input graph is chosen from some known distribution. Ailon [3] studied the equivalent problem for feedback arc set in tournaments. Note that all these query complexity results are adaptive and do not immediately give rise to small-space data stream algorithms.

Perhaps the relative lack of progress on streaming algorithms for directed graph problems stems from their being considered “implicitly hard” in the literature, a point made in the recent work of Khan and Mehta [19]. Indeed, that work and the also-recent work of Elkin [9] provide the first nontrivial streaming algorithms for computing a depth-first search tree and a shortest-paths tree (respectively) in semi-streaming space, using $O(n/\text{polylog } n)$ passes. Notably, fairly non-trivial work was needed to barely beat the trivial bound of $O(n)$ passes.

Some of our work here applies and extends the work of Guruswami and Onak [12], who gave the first super-linear (in $n$) space lower bounds in the streaming model for decision problems on graphs. In particular, they showed that solving reachability in $n$-vertex digraphs using $p$ passes requires $n^{1+\Omega(1/p)}/p^{O(1)}$ space. Via simple reductions, they then showed similar lower bounds for deciding whether a given (undirected) graph has a short $s-t$ path or a perfect matching.

1.1 Results

**Arbitrary Graphs.** To set the stage, in Section 2 we present a number of negative results for the case when the input digraph can be arbitrary. In particular, we show that there is no one-pass sublinear-space algorithm for such fundamental digraph problems as testing whether an input digraph is acyclic, topologically sorting it if it is, or finding its feedback arc set if it is not. These results set the stage for our later focus on specific families of graphs, where we can do much more, algorithmically.

For our lower bounds, we consider both arbitrary and random stream orderings. In Section 2.1, we concentrate on the arbitrary ordering and show that checking whether the graph is acyclic, finding a topological ordering of a directed acyclic graph (DAG), or any multiplicative approximation of feedback arc set requires $\Omega(n^2)$ space in one pass. The lower bound extends to $n^{1+\Omega(1/p)}/p^{O(1)}$ when the number of passes is $p \geq 1$. In Section 2.2, we show that essentially the same bound holds even when the stream is randomly ordered. This strengthening is one of our more technically involved results and it is based on generalizing a fundamental result by Guruswami and Onak [12] on $s-t$ connectivity in the multi-pass data stream model.

As a by-product of our generalization, we also obtain the first random-order super-linear (in $n$) lower bounds for the undirected graph problems of deciding (i) whether there exists a short $s-t$ path (ii) whether there exists a perfect matching.

**Tournaments.** A tournament is a digraph that has exactly one directed edge between each pair of distinct vertices. If we assume that the input graph is a tournament, it is trivial to find a topological ordering, given that one exists, by considering the in-degrees of the vertices. Furthermore, it is known that ordering the vertices by in-degree yields a 5-approximation to feedback arc set [8].

In Section 3, we present an algorithm which computes a $(1+\varepsilon)$-approximation to feedback arc set in one pass using $O(\varepsilon^{-2}n)$ space. However, in the post-processing step, it estimates the number of back edges for every permutation of vertices in the graph, thus resulting in exponential post-processing time. Despite its “brute force” feel, our algorithm is essentially optimal, both in its space usage (unconditionally) and its post-processing time (in a sense we shall make precise later). We address these issues in Section 3.4. On the other hand, in Section 3.2, we show that with $O(\log n)$ additional passes it is possible to compute a 3-approximation to feedback arc set while using only polynomial time and $O(n)$ space.

Lastly, in Section 4, we consider the problem of finding a sink in a tournament which is guaranteed to be acyclic. Obviously, this problem can be solved in a single pass using $O(n)$ space by maintaining an “is-sink” flag for each vertex. Our results show that for arbitrary order streams this is tight. We prove that finding a sink in $p$ passes requires $\Omega(n^{1/p}/p^2)$ space. We also provide an $O(n^{1/p}(\log(3p))$-space sink-finding algorithm that uses $O(p)$ passes, for any $1 \leq p \leq \log n$.

In contrast, we show that if the stream is randomly ordered, then using polylog $n$ space and a single pass is sufficient. This is a significant separation between the arbitrary-order and random-order data stream models.

**Random Graphs.** In Section 5, we consider a natural family of random acyclic graphs (see Definition 1.1 below) and present two algorithms for finding a topological ordering of vertices. We show that, for this family, $\tilde{O}(n^{4/3})$ space is sufficient to find the best ordering given $O(\log n)$ passes. Alternatively, $\tilde{O}(n^{3/2})$ space is sufficient given only a single pass, on the assumption that the edges in the stream are randomly ordered.

**Rank Aggregation.** In Section 6, we consider the problem of rank aggregation (formally defined in the next section), which is closely related to the feedback approximation.
Similar to our \((1 + \epsilon)\)-approximation of feedback arc set problem. We present a one-pass, \(O(1)\)-approx. FAS algorithm that returns \((1 + \epsilon)\)-approximation to the rank aggregation problem. The algorithm is very exponential post-processing time.

We now define the many interrelated digraph problems studied in this work. In each of these problems, the input is a digraph \(G\), presented as a stream of its edges. The ordering of the edges is adversarial unless specified otherwise.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Passes</th>
<th>Input Order</th>
<th>Space Bound</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACYC</td>
<td>1</td>
<td></td>
<td>(\Theta(n^2))</td>
<td></td>
</tr>
<tr>
<td>ACYC</td>
<td>(p)</td>
<td></td>
<td>(n^{1+\Omega(1/p)/p^{O(1)}})</td>
<td>exp. time post-processing</td>
</tr>
<tr>
<td>mult. approx. FAS</td>
<td>1</td>
<td></td>
<td>(\Theta(n^2))</td>
<td></td>
</tr>
<tr>
<td>mult. approx. FAS</td>
<td>(p)</td>
<td></td>
<td>(n^{1+\Omega(1/p)/p^{O(1)}})</td>
<td>exp. time post-processing</td>
</tr>
<tr>
<td>mult. approx. FAS</td>
<td>1</td>
<td></td>
<td>(\Theta(n^2))</td>
<td></td>
</tr>
<tr>
<td>STCONN-DAG</td>
<td>(p)</td>
<td>random</td>
<td>(n^{1+\Omega(1/p)/p^{O(1)}})</td>
<td>error probability (1/p^{O(p)})</td>
</tr>
<tr>
<td>TOPO-SORT</td>
<td>(O(\log n))</td>
<td>random</td>
<td>(\tilde{O}(n^{1+1/p}))</td>
<td></td>
</tr>
<tr>
<td>TOPO-SORT</td>
<td>1</td>
<td>random</td>
<td>(\tilde{O}(n^{1+1/p}))</td>
<td>random DAG + planted path</td>
</tr>
<tr>
<td>(1 + (\epsilon))-apx. RANK-AGGR</td>
<td>1</td>
<td>random</td>
<td>(O(\epsilon^{-2}n))</td>
<td>exp. time post-processing</td>
</tr>
</tbody>
</table>

Table 1: Summary of our algorithmic and space lower bound results. These problems are defined in Section 1.2. The input stream is adversarially ordered unless marked as “random” above. Besides the above results, we also give an oracle (query complexity) lower bound in Section 3.4.

A summary of these results is given in Table 1.
We remark that the most common definition of the minimum feedback arc set problem in the literature on optimization is to identify a small set of edges whose removal makes the graph acyclic, so FAS-SIZE is closer in spirit to this problem than FAS. As we shall see, our algorithms will apply to both variants of the problem. On the other hand, lower bounds sometimes require different proofs for the two variants. Since $\delta_G = 0$ iff $G$ is acyclic, we have the following basic observation.

**Observation 1.1.** Producing a multiplicative approximation for any of FAS, FAS-T, FAS-SIZE, and FAS-SIZE-T entails solving (respectively) TOPO-SORT, TOPO-SORT-T, ACYC, and ACYC-T.

For an ordering $\pi$ of a vertex set $V$, define $E^\pi = \{(u, v) \in V^2 : \pi(u) < \pi(v)\}$. Define $\text{Tou}(\pi) = (V, E^\pi)$ to be the unique acyclic tournament on $V$ consistent with $\pi$.

As mentioned above, we will also consider vertex ordering problems on random graphs from a natural distribution. This distribution, which we shall call a “planted path distribution,” was considered by Huang et al. [14] for average case analysis in their work on generalized sorting.

**Definition 1.1. (Planted Path Distribution)**

Let $\text{PlantDAG}_{n,q}$ be the distribution on digraphs on $[n]$ defined as follows. Pick a permutation $\pi$ of $[n]$ uniformly at random. Retain each edge $(u, v)$ in $\text{Tou}(\pi)$ with probability 1 if $\pi(v) = \pi(u) + 1$, and with probability $q$, independently, otherwise.

**Rank Aggregation.** The feedback arc set problem in tournaments is closely related to the problem of rank aggregation (RANK-AGGR). Given $k$ total orderings $\sigma_1, \ldots, \sigma_k$ of $n$ objects we want to find an ordering that best describes the “preferences” expressed in the input. Formally, we want to find an ordering that minimizes $	ext{cost}(\pi) := \sum_{i=1}^{k} d(\pi, \sigma_i)$, where the distance $d(\pi, \sigma)$ between two orderings is the number of pairs of objects ranked differently by them. That is,

$$d(\pi, \sigma) := \sum_{a, b \in [n]} I\{\pi(a) < \pi(b), \sigma(b) < \sigma(a)\},$$

where the notation $I\{\phi\}$ denotes a 0/1-valued indicator for the condition $\phi$.

In the streaming model, the input to RANK-AGGR can be given either as a concatenation of $k$ orderings, leading to a stream of length $kn$, or as a sequence of triples $(a, b, i)$ conveying that $\sigma_i(a) < \sigma_i(b)$, leading to a stream of length $k\binom{n}{2}$. Since we want the length of the stream to be polynomial in $n$, we assume $k = n^{O(1)}$.

**Lower Bounds through Communication Complexity.** Space lower bounds for data streaming algorithms are most often proven via reductions from standard problems in communication complexity. We recall two such problems, each involving two players, Alice and Bob. In the INDEX problem, Alice holds a vector $x \in \{0, 1\}^N$ and Bob holds an index $i \in [N]$; the goal is for Alice to send Bob a message allowing him to output $x_i$. In the DISJ problem, Alice holds $x \in \{0, 1\}^N$ and Bob holds $y \in \{0, 1\}^N$; the goal is for them to communicate interactively, following which they must decide whether $x$ and $y$ are disjoint, when considered as subsets of $[N]$, i.e., they must output $\bigvee_{i=1}^{N} x_i \land y_i$. In the special case DISJ$_{N,s}$, it is promised that the cardinalities $|x| = |y| = s$. In each case, the communication protocol may be randomized, erring with probability at most $\delta$. We shall use the following well-known lower bounds.

**Fact 1.1.** (See, e.g., [1, 21]) For error probability $\delta = \frac{1}{3}$, the one-way randomized complexity $R_1/3(\text{INDEX}_N) = \Omega(N)$ and the general randomized complexity $R_1/3(\text{DISJ}_{N,N/3}) = \Omega(N)$.

**Other Notation and Terminology.** We call an edge critical if it lies on a directed Hamiltonian path of length $n - 1$ in a directed acyclic graph. We say an event holds with high probability (w.h.p.) if the probability is at least $1 - 1/poly(n)$. Given a graph with a unique total ordering, we say a vertex $u$ has rank $r$ if it occurs in the $r$th position in this total ordering.

**2 General Digraphs and the Hardness of some Basic Problems**

In this section, our focus is bad news. In particular, we show that there is no one-pass sublinear-space algorithm for the rather basic problem of testing whether an input digraph is acyclic, nor for topologically sorting it if it is. These results set the stage for our later focus on tournament graphs, where we can do much more, algorithmically.

**2.1 Arbitrary Order Lower Bounds**

To begin, note that the complexity of TOPO-SORT-T is easily understood: maintaining in-degrees of all vertices and then sorting by in-degree provides a one-pass $O(n \log n)$-space solution. However, the problem becomes maximally hard without the promise of a tournament.

**Theorem 2.1.** Solving TOPO-SORT in one pass requires $\Omega(n^2)$ space.

**Proof.** We reduce from INDEX$_N$, where $N = p^2$ for a positive integer $p$. Using a canonical bijection from
being the result that the bound holds even when restricted to DAGs. It takes a little more work.

Based on their inputs, Alice and Bob create streams of edges by listing the following sets:

\[ E_x = \{ (t_i^0, r_j^0) : b \in \{0,1\}, i, j \in [p], x_{ij} = b \} , \]
\[ E_{yz} = \{ (t_x^1, t_y^0), (r_y^0, t_x^1) \} . \]

The combined stream defines the graph \( G \), where \( E = E_x \cup E_{yz} \).

We claim that \( G \) is acyclic. In the digraph \( (V,E_x) \), every vertex is either a source or a sink. So the only vertices that could lie on a cycle in \( G \) are \( t_i^0, r_j^0, t_j^1 \), and \( r_i^0 \). Either \( (t_i^0, r_j^0) \notin E \) or \( (t_j^1, r_i^0) \notin E \), so there is in fact no cycle even among these four vertices.

Let \( \sigma \) be a topological ordering of \( G \). If \( x_{yz} = 0 \), then we must, in particular, have \( \sigma(t_j^1) > \sigma(t_i^0) \). Thus, by simulating a one-pass algorithm \( \mathcal{A} \) on Alice’s stream followed by Bob’s stream, consulting the ordering \( \sigma \) produced by \( \mathcal{A} \) and outputting 0 iff \( \sigma(t_j^1) < \sigma(t_i^0) \), the players can solve INDEX-N. It follows that the space used by \( \mathcal{A} \) must be at least \( R_{1/3}^{-1}(\text{INDEX-N}) = \Omega(N) = \Omega(p^2) = \Omega(n^2) \).

For our next two results, we use reductions from STCONN-DAG. It is a simple exercise to show that a one-pass streaming algorithm for STCONN-DAG requires \( \Omega(n^2) \) space. Guruswami and Onak [12] showed that a \( p \)-pass algorithm requires \( n^{1+\Omega(1/p)}/p^{O(1)} \) space.\(^3\)

**Proposition 2.1.** Solving ACYC requires \( \Omega(n^2) \) space in one pass and \( n^{1+\Omega(1/p)}/p^{O(1)} \) space in \( p \) passes.

**Proof.** Given a DAG \( G \) and specific vertices \( s,t \), let \( G' \) be obtained by adding edge \((t,s)\) to \( G \). Then \( G' \) is acyclic iff \( G \) has no \( s \)-to-\( t \) path. By the discussion above, the lower bounds on ACYC follow.

**Corollary 2.1.** A one-pass multiplicative approximation algorithm for either FAS or FAS-SIZE requires \( \Omega(n^2) \) space. Such approximation for FAS-SIZE in \( p \) passes requires \( n^{1+\Omega(1/p)}/p^{O(1)} \) space.

**Proof.** This is immediate from Observation 1.1, Theorem 2.1, and Proposition 2.1.

Proving a similar multi-pass lower bound for FAS takes a little more work.

**Proposition 2.2.** A \( p \)-pass \( O(1) \)-approximation algorithm for FAS requires \( n^{1+\Omega(1/p)}/p^{O(1)} \) space.

**Proof.** Let \( \mathcal{A} \) be a \( c \)-approximation algorithm for FAS, with \( c = O(1) \). We reduce from STCONN-DAG as follows. Let graph \( G \) with specified vertices \( s,t \) be an input to STCONN-DAG. Make \( c+1 \) copies of \( G \). Introduce two new vertices \( s^* \) and \( t^* \). Add directed edges from \( s^* \) to each copy of \( s \), from each copy of \( t \) to \( t^* \), and from \( t^* \) to \( s^* \). Let \( G' \) be the resulting \((cn + n + 2)\)-vertex graph.

When there is no \( s \)-to-\( t \) path in \( G \), our graph \( G' \) is acyclic, so \( \mathcal{A} \) must output a topological ordering \( \sigma \): in particular, \( \sigma(t^*) < \sigma(s^*) \).

When there is an \( s \)-to-\( t \) path in \( G \), a FAS-minimizing ordering of \( G' \) must place \( s^* \) before \( t^* \), creating exactly one back edge. On the other hand, there are at least \( c+1 \) directed cycles in \( G' \) that intersect only in the edge \((t^*,s^*)\). In any ordering \( \tau \) of \( G' \) with \( \tau(t^*) < \tau(s^*) \), each of these \( c+1 \) cycles contributes at least one distinct back edge. Thus, \( \mathcal{A} \), being a \( c \)-approximation, cannot output such \( \tau \) and must therefore output \( \sigma \) with \( \sigma(s^*) < \sigma(t^*) \).

The lower bound follows as \( \mathcal{A} \)‘s placement of \( s^* \) and \( t^* \) solves STCONN-DAG on \( G \).

\[ \square \]

### 2.2 Random Order Lower Bounds

We consider the STCONN-DAG, ACYC, and FAS problems in a uniformly randomly ordered digraph stream. Recall that for adversarially ordered streams, these problems require about \( n^{1+\Omega(1/p)} \) space in \( p \) passes. The hardness ultimately stems from a similar lower bound for the SHORTPATH-DAG problem. In this latter problem, the input is an \( n \)-vertex DAG with two designated vertices \( v_s \) and \( v_t \), such that either (a) there exists a path of length at most \( 2p + 2 \) from \( v_s \) to \( v_t \) or (b) \( v_t \) is unreachable from \( v_s \). The goal is to determine which of these is the case.

Our goal in this section is to show that the same lower bound continues to hold under random ordering, provided we insist on a sufficiently small error probability, about \( 1/p^{O(1)} \). We prove this for SHORTPATH-DAG. As this is a special case of STCONN-DAG, a lower bound for SHORTPATH-DAG carries over to STCONN-DAG. Further, by the reductions in Propositions 2.1 and 2.2, the lower bounds also carry over to ACYC and FAS. We also show a barrier result arguing that this restriction to low error is necessary: for the SHORTPATH-DAG problem, if an error probability of at least \( 2/p! \) is allowed, then \( O(n) \) space is achievable in \( p \) passes.

Our proof uses the machinery of the Guruswami–Onak lower bound for SHORTPATH-DAG under an adversarial stream ordering [12]. As in their work, we derive our space lower bound from a communication lower bound for set chasing intersection (henceforth, \( \square \)

\( ^3 \)Although their paper states the lower bound for \( s \)-to-\( t \) connectivity in general digraphs, their proof in fact shows the stronger result that the bound holds even when restricted to DAGs.
However, unlike them, we need to prove a “robust” lower bound for SCI, in the sense of Chakrabarti, Cormode, and McGregor [7], as explained below. To define SCI, we first set up a special family of multilayer pointer jumping problems, described next.

Picture a layered digraph $G^*$ with $2k + 1$ layers of vertices, each layer having $m$ vertices, laid out in a rectangular grid with each column being one layer. From left to right, the layers are numbered $−k, −k + 1, \ldots, k$. Layer 0 is called the middle-layer. The only possible edges of $G^*$ are from a layer $\ell$ to layer $\ell − 1$, or from layer $−\ell$ to layer $−\ell + 1$, for $\ell \in [k]$ (i.e., edges travel from the left and right ends of the rectangular grid towards the mid-layer). We designate the first vertex in layer $−k$ as $v_s$, and the first vertex in layer $k$ as $v_t$.

Each vertex not in the mid-layer has exactly $t$ outgoing edges, numbered 1st through $t$th, possibly with repetition (i.e., $G^*$ is a multigraph). Think of these edges as pointers. An input to one of our communication problems (to be defined soon) specifies the destinations of these pointers. Thus, an input consists of $2mkt$ tokens, where each token is an integer in $[m]$ specifying which of the $m$ possibilities a certain pointer takes. The pointers emanating from layer $\ell$ of vertices constitute the $\ell$th layer of pointers. Our communication games will involve $2k$ players named $P_{−k}, P_{−k+1}, P_1, \ldots, P_k$. We say that $P_\ell$ is the natural owner of the portion of the input specifying the $\ell$th layer of pointers.

In the SCI$_{m,k,t}$ problem, the goal is to determine whether or not there exists a mid-layer vertex reachable from $v_s$ as well as $v_t$. Consider the communication game where each pointer is known to its natural owner and the players must communicate in $k − 1$ rounds, where in each round they broadcast messages in the fixed order $P_{−1}, \ldots, P_{−k}, P_1, \ldots, P_k$. Guruswami and Onak showed that this problem requires total communication $\Omega((m^{1+1/(2k)})/k^{16} \log^{3/2} m)$ in the parameter regime $t^{2k} \ll m$. This almost immediately implies a similar lower bound for SHORTPATH-DAG—simply reverse the directions of the pointers in positive-numbered layers—which then translates into a data streaming lower bound along standard lines.

The key twist in our version of the SCI problem is that each pointer is allocated to one of the $2k$ players uniformly at random: thus, most pointers are not allocated to their natural owners. The players have to determine the output to SCI communicating exactly in the same pattern as before, up to a small error probability taken over the protocol’s internal randomness as well as the random allocation. This setup potentially makes the problem easier because there is a good chance that the players will be able to “jump two pointers” within a single round. Our main technical result is to show that a lower bound of the form $m^{1+\Omega(1/k)}$ holds despite this. In the terminology of Chakrabarti et al. [7], who lower-bounded a number of communication problems under such random-allocation setups, this is a robust communication lower bound.

**Theorem 2.2.** Suppose that $t^{2k} = o(m/\text{polylog}(m))$ and that protocol II solves SCI$_{m,k,t}$ with error $\varepsilon < (2k)^{-2k−2}$ when the input is randomly allocated amongst the $2k$ players, as described above. Then, II communicates $\Omega(m^{1+1/(2k)/}k^{16} \log^{3/2} m)$ bits.

To prove this result, we consider a problem we call MPJ-MEET$_{m,k,t}$, defined next (Guruswami and Onak called this problem OR ∘ LPCE). Consider an input $G^*$ to SCI$_{m,k,t}$ and fix an $i \in [t]$. If we retain only the $i$th pointer emanating from each vertex, for each layer $\ell$, the $\ell$th layer of pointers defines a function $f_{\ell,i} : [n] \rightarrow [n]$. Let $x_i$ (respectively, $y_i$) denote the index of the unique mid-layered vertex reached from $v_s$ (respectively, $v_t$) by following the retained pointers. Formally,

$$x_i = f_{−1,i}(f_{−2,i}(\cdots f_{−k,i}(1)\cdots)),$$

$$y_i = f_{1,i}(f_{2,i}(\cdots f_{k,i}(1)\cdots)).$$

Define a function to be $r$-thin if every element in its range has at most $r$ distinct pre-images. The instance $G^*$ is said to meet at $i$ if $x_i = y_i$, and is said to be $r$-thin at $i$ if each function $f_{\ell,i}$ is $r$-thin. The desired output of MPJ-MEET is

$$\text{MPJ-MEET}(G^*) = \bigvee_{i=1}^t \{G^* \text{ meets at } i\} \lor \{G^* \text{ is not } (C \log m)\text{-thin at } i\},$$

for an appropriate universal constant $C$. The corresponding communication game allocates each pointer to its natural owner and asks them to determine the output using the same communication pattern as for SCI. Here is the key result about this problem.

**Lemma 2.1.** (Lemma 7 of Guruswami–Onak [12])

The $(k − 1)$-round constant-error communication complexity of MPJ-MEET is lower-bounded as follows:

$$R^{k−1}(\text{MPJ-MEET}_{m,k,t}) = \Omega(tm/(k^{16} \log m)) − O(kt^2).$$

Using this, we prove our main technical result.

**Proof.** [Proof of Theorem 2.2] Based on the $\varepsilon$-error protocol II for SCI$_{m,k,t}$, we design a protocol $Q$ for MPJ-MEET$_{m,k,t}$ as follows. Let $G^*$ be an instance of MPJ-MEET allocated to players as described above. The
players first check whether, for some $i$, $G^*$ fails to be $r$-thin at $i$, for $r := C \log m$: this check can be performed in the first round of communication with each player communicating a single bit. If the check passes, the protocol ends with output $1$. From now on, we assume that $G^*$ is indeed $r$-thin at each $i \in [t]$.

Using public randomness, the players randomly renumber the vertices in each layer of $G^*$, creating an instance $G'$ of SCI. The players then choose $\rho$, a random allocation of pointers as in the SCI problem. They would like to simulate $\Pi$ on $G'$, as allocated by $\rho$, but of course they can’t do so without additional communication. Instead, using further public randomness, for each pointer that $\rho$ allocates to someone besides its natural owner, the players reset that pointer to a uniformly random (and independent) value in $[m]$. We refer to such a pointer as damaged. Since there are $2k$ players, each pointer is damaged with probability $1 - 1/(2k)$. Let $G''$ denote the resulting random instance of SCI. The players then simulate $\Pi$ on $G''$ as allocated by $\rho$.

It remains to analyze the correctness properties of $\Pi$. Suppose that $G^*$ is a $1$-instance of MPJ-MEET. Then there exists $i \in [t]$ such that $G^*$ meets at $i$. By considering the unique maximal paths out of $v_s$ and $v_l$ following only the $i$th pointers at each vertex, we see that $G^*$ is also a $1$-instance of SCI. Since the vertex renumbering preserves connectivity, $G'$ is also a $1$-instance of SCI. With probability $(2k)^{-2k}$, none of the $2k$ pointers on these renumbered paths is damaged; when this event occurs, $G''$ is also a $1$-instance of SCI. Therefore, $\Pi$ outputs $1$ with probability at least $(2k)^{-2k} (1 - \text{err}(\Pi)) \geq (2k)^{-2k} (1 - \varepsilon)$.

Next, suppose that $G^*$ is a $0$-instance of MPJ-MEET. It could be that $G^*$ is a $1$-instance of SCI. However, as Guruswami and Onak show, the random vertex renumbering ensures that $\text{Pr}[\text{SCI}(G') = 1] < o(1)$. For the rest of the argument, assume that $\text{SCI}(G') = 0$. In order to have $\text{SCI}(G'') = 1$, there must exist a mid-layer vertex $x$ such that

$$x = f_{1,i_1}(f_{2,i_2}(...) f_{k,i_k}(1) \cdots))$$

(2.1)$$

$$= f_{-1,j_1}(f_{-2,j_2}(...) f_{-k,j_k}(1) \cdots))$$

for some choice of pointer numbers $i_1, \ldots, i_k, j_1, \ldots, j_k \in [t]$. We consider three cases.

- **Case 1:** None of the pointers in the above list is damaged. In this case, eq. (2.1) cannot hold, because $\text{SCI}(G') = 0$.

- **Case 2:** The layer-1 pointer in the above list is damaged. Condition on a particular realization of pointers in negative-numbered layers and let $x$ denote the mid-layered vertex reached from $v_s$ by following pointers numbered $j_k, \ldots, j_1$, as in eq. (2.1). The probability that the damaged pointer at layer $1$ points to $x$ is $1/m$. Since this holds for each conditioning, the probability that $\text{SCI}(G'') = 1$ is also $1/m$.

- **Case 3:** The layer-$\ell$ pointer is damaged, but pointers in layers $1, \ldots, \ell - 1$ are not, where $\ell \geq 2$. Again, condition on a particular realization of pointers in negative-numbered layers and let $x$ be as above. Since the functions $f$ in eq. (2.1) are all $r$-thin, the number of vertices in layer $\ell - 1$ that can reach $x$ using only undamaged pointers is at most $r^{\ell-1} \leq r^{k-1}$. The probability that the damaged pointer at layer $\ell$ points to one of these vertices is at most $r^{k-1}/m$.

Combining the cases, the probability that eq. (2.1) holds for a particular choice of pointer numbers $i_1, \ldots, i_k, j_1, \ldots, j_k \in [t]$ is at most $r^{k-1}/m$. Taking a union bound over the $i^{2k}$ choices, the overall probability $\text{Pr}[\text{SCI}(G'') = 1] < i^{2k} r^{k-1}/m = o(1)$, for the parameter regime $i^{2k} = o(m/\text{polylog}(m))$ and $r = O(\log m)$. Therefore, $\Pi$ outputs $1$ with probability at most $\text{err}(\Pi) + o(1) = \varepsilon + o(1)$.

Thus far, we have a protocol $\Pi$ that outputs $1$ with probability $\alpha$ when MPJ-MEET($G^*$) = 0 and has probability $\beta$ when MPJ-MEET($G^*$) = 1, where $\alpha \leq \varepsilon + o(1)$ and $\beta \geq (2k)^{-2k}(1 - \varepsilon)$. Recall that $\varepsilon = (2k)^{-2k} - 2\beta$, so $\beta$ is bounded away from $\alpha$. Let $\Pi'$ be a protocol where we first toss an unbiased coin: if it lands heads, we output 0 with probability $\delta := (\alpha + \beta)/2$ and 1 with probability $1 - \delta$; if it lands tails, we simulate $\Pi$. Then $\Pi'$ is a protocol for MPJ-MEET with error probability $\frac{1}{2} - (\beta - \alpha)/4$. By Lemma 2.1, this protocol must communicate $\Omega(m^{1+1/(2k)} / k^{16} \log^{3/2} m)$ bits and so must $\Pi$.

By a standard reduction from random-allocation communication protocols to random-order streaming algorithms, we obtain the following lower bound: the main result of this section.

**Theorem 2.3.** For each constant $p$, a $p$-pass algorithm that solves SHORTPATH-DAG on $n$-vertex digraphs whose edges presented in a uniform random order, erring with probability at most $1/p^{\Omega(p)}$, must use $\Omega(n^{1+1/(2p+2)} / \log^{3/2} n)$ bits of space.

---

4This step is exactly as in Guruswami-Onak [12]. Formally, each function $f_{i,j}$ is replaced by a corresponding function of form $\pi_{i,j} \circ f_{i,j} \circ \pi_{i,j}^{-1}$ (for $\ell > 0$), for random permutations $\pi_{i,j}: [m] \rightarrow [m]$. To keep things concise, we omit the full details here.

5See the final paragraph of the proof of Lemma 11 in [12].
Consequently, similar lower bounds hold for the problems STCONN-DAG, ACYC, and FAS.

This paper is focused on directed graph problems. However, it is worth noting that a by-product of our generalization of the Guruswami–Onak bound to randomly ordered streams is that we also obtain the first random-order super-linear (in \(n\)) lower bounds for two undirected graph problems.

**Corollary 2.2.** For each constant \(p\), \(n^{1+\Omega(1/p)}\) space is required to solve either of the following problems in \(p\) passes, erring with probability at most \(1/p^{\Omega(p)}\), over a randomly ordered edge stream of an \(n\)-vertex directed graph \(G\):

- decide whether \(G\) contains a perfect matching;
- decide whether the distance between prespecified vertices \(v_s\) and \(v_t\) is at most \(2p+2\).

**A Barrier Result.** Notably, Theorem 2.3 applies only to algorithms with a rather small error probability. This is inherent: allowing just a slightly larger error probability renders the problem solvable in semi-streaming space and \(\Omega(p)\) passes, erring with probability at most \(\ell(p)\). This is shown in the result below, which should be read as a barrier result rather than a compelling algorithm.

**Proposition 2.3.** Given a randomly ordered edge stream of a digraph \(G\), the SHORTPATH-DAG problem on \(G\) can be solved using \(\tilde{O}(n)\) space and \(p\) passes, with error probability at most \(2/p!\).

**Proof.** Recall that we’re trying to decide whether or not \(G\) has a path of length at most \((2p+2)\) from \(v_s\) to \(v_t\). The high-level idea is that thanks to the random ordering, a “Bellman–Ford” style algorithm that grows a forward path out of \(v_s\) and a backward path out of \(v_t\) is very likely to make more than one step of progress during some pass.

To be precise, we maintain arrays \(d_s\) and \(d_t\), each indexed by \(V\). Initialize the arrays to \(\infty\), except that \(d_s[v_s] = d_t[v_t] = 0\). During each pass, we use the following logic:

for each edge \((x, y)\) in the stream:
- if \(d_s[x] + d_t[y] \leq 2p + 1\): output TRUE and halt
  - \(d_s[y] \leftarrow \min(d_s[y], 1 + d_s[x])\)
  - \(d_t[x] \leftarrow \min(d_t[x], 1 + d_t[y])\)

If we complete \(p\) passes without any output, then we output FALSE.

If \(G\) has no short enough path from \(v_s\) to \(v_t\), this algorithm will always output FALSE. So let’s consider the other case, when there is a \(v_s-v_t\) path \(\pi\) of length at most \(2p+2\). Let vertex \(z\) be the midpoint of \(\pi\), breaking ties arbitrarily if needed. The subpaths \([v_s, z]_{\pi}\) and \([z, v_t]_{\pi}\) have lengths \(q\) and \(r\), respectively, with \(q \leq p+1\) and \(r \leq p+1\). Notice that if our algorithm is allowed to run for \(q\) (resp. \(r\)) passes, then \(d_s[z]\) (resp. \(d_t[z]\)) will settle to its correct value. If both of them settle, then the algorithm correctly outputs TRUE. So, the only nontrivial case is when \(q, r \in \{p, p+1\}\).

Let \(E_s\) be the event that the random ordering of the edges in the stream places the edges of \([v_s, z]_{\pi}\) in the exact reverse order of \(\pi\). Let \(E_t\) be the event that the random ordering places the edges of \([z, v_t]_{\pi}\) in the exact same order as \(\pi\). If \(E_s\) does not occur, then for some two consecutive edges \((w, x), (x, y)\) on \([v_s, z]_{\pi}\), the stream puts \((w, x)\) before \((x, y)\). Therefore, once \(d_s[w]\) settles to its correct value, the following pass will settle not just \(d_s[x]\), but also \(d_t[y]\): therefore, after \(q - 1 \leq p\) passes, \(d_s[z]\) is settled. Similarly, if \(E_t\) does not occur, then after \(r - 1 \leq p\) passes, \(d_t[z]\) is settled. As noted above, if both of them settle, the algorithm correctly outputs TRUE.

Thus, the error probability \(\leq \Pr[E_s \lor E_t] \leq \Pr[E_s] + \Pr[E_t] = 1/q! + 1/r! \leq 2/p!\), as required.

3 Feedback Arc Set in Tournaments

3.1 Accurate, One Pass, but Slow Algorithm for FAS-T

We shall now design an algorithm for FAS-T (that also solves FAS-SIZE-T) based on linear sketches for \(\ell_1\)-norm estimation. Recall that the \(\ell_1\)-norm of a vector \(x \in \mathbb{R}^N\) is \(\|x\|_1 = \sum_{i \in [N]} |x_i|\). A \(d\)-dimensional \(\ell_1\)-sketch with accuracy parameter \(\varepsilon\) and error parameter \(\delta\) is a distribution \(S\) over \(d \times N\) matrices, together with an estimation procedure \(\text{Est}: \mathbb{R}^d \rightarrow \mathbb{R}\) such that

\[
\Pr_{S \leftarrow S}[\left(1 - \varepsilon\right)\|x\|_1 \leq \text{Est}(Sx) \leq \left(1 + \varepsilon\right)\|x\|_1] \geq 1 - \delta.
\]

Such a sketch is “stream friendly” if there is an efficient procedure to generate a given column of \(S\) and further, \(\text{Est}\) is efficient. Obviously, a stream friendly sketch leads to a space and time efficient algorithm for estimating \(\|x\|_1\) given a stream of entrywise updates to \(x\). We shall use the following specialization of a result of Kane et al. [17].

**Fact 3.1. (Kane et al. [17])** There is a stream friendly \(d\)-dimensional \(\ell_1\)-sketch with accuracy \(\varepsilon\) and error \(\delta\) that can handle \(N^{O(1)}\) many \(\pm 1\)-updates to \(x \in \mathbb{R}^N\), with each update taking \(O(\varepsilon^{-2} \log \varepsilon^{-1} \log \delta^{-1} \log N)\) time, with \(d = O(\varepsilon^{-2} \log \log \delta^{-1})\), and with entries of the sketched vector fitting in \(O(\log N)\) bits.
There is a one-pass algorithm for FAS-T that uses $O(\varepsilon^{-2}n\log^2 n)$ space and returns a $(1 + \varepsilon)$-approximation with probability at least $\frac{2}{3}$, but requires exponential post-processing time.

Proof. Identify the vertex set of the input graph $G = (V, E)$ with $|\mathcal{V}|$ and put $N = \binom{\mathcal{V}}{2}$. We index vectors $z$ in $\mathbb{R}^N$ as $z_{uv}$, where $1 \leq u < v \leq n$. Define a vector $x \in \{0, 1\}^N$ based on $G$ and vectors $y^\pi \in \{0, 1\}^N$ for each permutation $\pi$: $[n] \rightarrow [n]$ using indicator variables as follows.

$$x_{uv} = 1\{(u, v) \in E\}, \quad y^\pi_{uv} = 1\{\pi(u) < \pi(v)\}.$$ 

A key observation is that the uv-entry of $x - y^\pi$ is nonzero iff the edge between $u$ and $v$ is a back edge of $G$ according to the ordering $\pi$. Thus, $\lVert B_G(\pi) \rVert = \lVert x - y^\pi \rVert_1$. By Fact 3.1, this takes $O\left(\varepsilon^{-2}n\log^2 n\right)$ space and $O\left(\varepsilon^{-2}\log\varepsilon^{-1}n\log^2 n\right)$ time per edge.

In post-processing, the algorithm considers all $n!$ permutations $\pi$ and, for each of them, computes $S(x - y^\pi) = Sx - Sy^\pi$. It thereby recovers an estimate for $\lVert x - y^\pi \rVert_1$ and finally outputs the ordering $\pi$ that minimizes this estimate. By a union bound, the probability that every estimate is $(1 \pm \varepsilon/3)$-accurate is at least $1 - n!\cdot\delta = 2/3$. When this happens, the output ordering provides a $(1 + \varepsilon)$-approximation to FAS-T by our key observation above.

Despite its “brute force” feel, the above algorithm is essentially optimal, both in its space usage (unconditionally) and its post-processing time (in a sense we shall make precise later). We address these issues in Section 3.4.

3.2 Multiple Passes: FAS-T in Polynomial Time

For a more time-efficient streaming algorithm, we design one based on the KWIKSORT algorithm of Ailon et al. [4]. This (non-streaming) algorithm operates as follows on a tournament $G = (V, E)$.

- Choose a random ordering of the vertices: $v_1, v_2, \ldots, v_n$.
- Vertex $v_1$ partitions $V$ into two sub-problems $\{u : (u, v_1) \in E\}$ and $\{w : (v_1, w) \in E\}$. At this point we know the exact place of $v_1$ in the ordering.
- Vertex $v_2$ further partitions one of these sub-problems. Proceeding in this manner, after $v_1, v_2, \ldots, v_i$ are considered, there are $i + 1$ sub-problems.
- Continue until all $n$ vertices are ordered.

When $v_i$ is being used to divide a sub-problem we refer to it as a pivot.

Emulating KWIKSORT in the Data Stream Model. We will emulate KWIKSORT in $p$ passes over the data stream. In each pass, we will consider the action of multiple pivots. Partition $v_1, \ldots, v_n$ into $p$ groups $V_1, \ldots, V_p$, where $V_1 = \{v_1, \ldots, v_{cn^{1/p} \log n}\}$. $V_2$ consists of the next $cn^{2/p} \log n$ vertices in the sequence, and $V_j$ contains $cn^{j/p} \log n$ vertices coming after $V_{j-1}$. Here $c$ is a sufficiently large constant. At the end of pass $j + 1$, we want to emulate the effect of pivots in $V_{j+1}$ on the sub-problems resulting from considering pivots in $V_1$ through $V_j$. In order to do that, in pass $j + 1$ for each vertex $v \in V_{j+1}$ we store all edges between $v$ and vertices in the same sub-problem as $v$, where the sub-problems are defined at the end of pass $j$.

The following combinatorial lemma plays a key role in analyzing this algorithm’s space usage.

Lemma 3.1. (Mediocrity Lemma) In an $n$-vertex tournament, if we pick a vertex $v$ uniformly at random, then $\Pr[|\mathcal{V}| < d_{in}(v) < (1 - \varepsilon)|\mathcal{V}| \geq 1 - 4\varepsilon$. Similarly, $\Pr[|\mathcal{V}| < d_{out}(v) < (1 - \varepsilon)|\mathcal{V}| \geq 1 - 4\varepsilon$. In particular, with probability at least $1/3$, $v$ has in/out-degree between $n/6$ and $5n/6$.

Proof. Let $H$ be a set of vertices of in-degree at least $(1 - \varepsilon)n$. Let $h = |H|$. On the one hand, $\sum_{v \in H} d_{in}(v) \geq (1 - \varepsilon)n$. On the other hand, the edges that contribute to the in-degrees of vertices in $H$ have both endpoints in $H$ or one endpoint in $H$ and one in $V \setminus H$. The number of such edges is

$$\sum_{v \in H} d_{in}(v) \leq \binom{h}{2} + h(n - h) = \frac{1}{2}(2nh - h^2 - h).$$

Therefore, $(2nh - h^2 - h)/2 \geq (1 - \varepsilon)n$. This implies $h < 2\varepsilon n$. Thus, the number of vertices with in-degree at least $(1 - \varepsilon)n$ (and out-degree at most $\varepsilon n$) is $h < 2\varepsilon n$. By symmetry, the number of vertices with out-degree at least $(1 - \varepsilon)n$ (and in-degree at most $\varepsilon n$) is also less than $2\varepsilon n$. Thus, the probability a random vertex has...
in/out-degree between $\varepsilon n$ and $(1 - \varepsilon)n$ is $(n - 2h)/n > (n - 2 \cdot 2\varepsilon n)/n = 1 - 4\varepsilon$. □

**Space Analysis.** Let $M_j$ be the maximum size of a sub-problem after pass $j$. The number of edges collected in pass $j + 1$ is then at most $M_j|V_{j+1}|$. By Lemma 3.2 (below), this is at most $cn^{j+1/p} \log n$. Once the post-processing of pass $j + 1$ is done, the edges collected in that pass can be discarded.

**Lemma 3.2.** With high probability, $M_j \leq n^{1-j/p}$ for all $j$.

**Proof.** Let $M^v_j$ denote the size of the sub-problem that contains $v$, after the $j$th pass. We shall prove that, for each $v$, $\Pr[M^v_j > n^{1-j/p}] \leq 1/n^{10}$. The lemma will then follow by a union bound.

Take a particular vertex $v$. If, before the $j$th pass, we already have $M^v_{j-1} \leq n^{1-j/p}$, there is nothing to prove. So assume that $M^v_{j-1} > n^{1-j/p}$. Call a pivot “good” if it reduces the size of the sub-problem containing $v$ by a factor of at least 5/6. A random pivot fails in the same sub-problem as $v$ with probability at least $n^{-j/p}/n$; when this happens, by the Mediocrity Lemma, the probability that the pivot is good is at least $1/3$. Overall, the probability that the pivot is good is at least $n^{-j/p}/3$.

In the $j$th pass, we use $cn^{j/p} \log n$ pivots. If at least $\log_{6/5} n$ of them are good, we definitely have $M^v_j \leq n^{1-j/p}$. Thus, by a Chernoff bound, for a sufficiently large $c$, we have

$$
\Pr \left[ M^v_j > n^{1-j/p} \right] \\
\leq \Pr \left[ \text{Bin} \left( cn^{j/p} \log n, n^{-j/p}/3 \right) < \log_{6/5} n \right] \\
\leq 1/n^{10}.
$$

**Theorem 3.2.** There exists a polynomial time $p$-pass data stream algorithm using $O(n^{1+1/p})$ space that returns a 3-approximation (in expectation) for FAS-T.

**Proof.** The pass/space tradeoff follows from Lemma 3.2 and the discussion above it; the approximation factor follows directly from the analysis of Ailon et al. [4]. □

### 3.3 A Space Lower Bound

Both our one-pass algorithm and the $O(\log n)$-pass instantiation of our multi-pass algorithm use at least $\Omega(n)$ space. For FAS-SIZE-T, where the desired output is a just a number, it is reasonable to ask whether $o(n)$-space solutions exist. We now prove that they do not.

**Proposition 3.1.** Solving ACYC-T is possible in one pass and $O(n \log n)$ space. Meanwhile, any $p$-pass solution requires $\Omega(n/p)$ space.

**Proof.** For the upper bound, we maintain the in-degrees of all vertices in the input graph $G$. Since $G$ is a tournament, the set of in-degrees is exactly $\{0, 1, \ldots, n - 1\}$ if the input graph is acyclic.

For the lower bound, we reduce from $\text{DISJ}_N,N/3$. Alice and Bob construct a tournament $T$ on $n = 7N/3$ vertices, where the vertices are labeled $\{v_1, v_2, \ldots, v_{2N}, w_1, \ldots, w_{N/3}\}$. Alice, based on her input $x$, adds edges $(v_i, v_{i+1})$ for each $i \in x$. For each remaining pair $(i, j) \in [2N] \times [2N]$ with $i < j$, she adds the edge $(v_i, v_j)$. Let $a_1 < \ldots < a_{N/3}$ be the sorted order of the elements in Bob’s set $y$. For each $k = a_\ell \in y$, Bob defines the alias $v_{2N+k} = w_\ell$ and then adds the edges $E_k = \{(v_i, v_{2N+k}) : 1 \leq i \leq 2k - 1\} \cup \{(v_{2N+k}, v_j) : 2k \leq j \leq 2N\}$.

Finally, he adds the edges $\{(w_i, w_j) : 1 \leq i < j < N/3\}$. This completes the construction of $T$.

We claim that the tournament $T$ is acyclic iff $x \cap y = \emptyset$. The “only if” part is direct from construction, since if $x$ and $y$ intersect at some index $k \in [N]$, we have the directed cycle $(v_{2k}, v_{2k-1}, v_{2N+k}, v_{2k})$. For the “if” part, let $\sigma$ be the ordering $(v_1, \ldots, v_{2N})$ and let $T' = \text{Tou}(\sigma)$, as defined in Section 1.2. We show how to modify $\sigma$ into a topological ordering of $T$, proving that $T$ is acyclic. Observe that, by construction, the tournament $T \setminus \{w_1, \ldots, w_{N/3}\}$ can be obtained from $T'$ by flipping only the edges $(v_{2k-1}, v_{2k})$ for each $i \in x$. Each time we perform such an edge flip, we modify the topological ordering of $T'$ by swapping the associated vertices of the edge. The resultant ordering would still be topological as the vertices were consecutive in the ordering before the flip. Thus, after performing these swaps, we get a topological ordering of $T \setminus \{w_1, \ldots, w_{N/3}\}$. Now, consider some $k \in y$. Since $x \cap y = \emptyset$, $k \notin x$ and so, $v_{2k}$ succeeds $v_{2k-1}$ in this ordering, just as in $\sigma$, since we never touched these two vertices while performing the swaps. Thus, for each such $k$, we can now insert $v_{2N+k}$ between $v_{2k-1}$ and $v_{2k}$ in the ordering and obtain a topological ordering of $T$. This proves the claim.

Thus, given a $p$-pass solution to ACYC-T using $s$ bits of space, we obtain a protocol for $\text{DISJ}_N,N/3$ that communicates at most $(2p - 1)s$ bits. By Fact 1.1, $(2p - 1)s = \Omega(N) = \Omega(n)$, i.e., $s = \Omega(n/p)$. □

**Theorem 3.3.** A $p$-pass multiplicative approximation for FAS-SIZE-T requires $\Omega(n/p)$ space.

**Proof.** This is immediate from Observation 1.1 and Proposition 3.1. □
3.4 An Oracle Lower Bound

Let us now consider the nature of the post-processing performed by our one-pass FAS-T algorithm in Section 3.1. During its streaming pass, that algorithm builds an oracle based on $G$ that, when queried on an ordering $\sigma$, returns a fairly accurate estimate of $|B_G(\sigma)|$. It proceeds to query this oracle $n$ times to find a good ordering. This raises the question: is there a more efficient way to exploit the oracle that the algorithm has built? A similar question was asked in Bateni et al. [6] in the context of using sketches for the maximum coverage problem.

Were the oracle exact—i.e., on input $\sigma$ it returned $|B_G(\sigma)|$ exactly—then two queries to the oracle would determine which of $(i, j)$ and $(j, i)$ was an edge in $G$. It follows that $O(n \log n)$ queries to such an oracle suffice to solve FAS-T and FAS-SIZE-T. However, what we actually have is an $\epsilon$-oracle; defined as one that, on query $\sigma$, returns $\beta \in \mathbb{R}$ such that $(1 - \epsilon)|B_G(\sigma)| \leq \beta \leq (1 + \epsilon)|B_G(\sigma)|$. We shall show that an $\epsilon$-oracle cannot be exploited efficiently: a randomized algorithm will, when queried on an ordering $\sigma$, make two queries to the oracle and, after at most $q$ such queries, outputs its answer to $P(G)$. We require this answer to be correct with probability at least $\frac{3}{4}$.

Let $\sigma$ be an ordering of $[n]$. Let $T$ be sampled from either $D_{yes}$ or $D_{no}$.

$$
\mathbb{E}|B_T(\sigma)| = m := \frac{1}{2} \binom{n}{2}.
$$

In fact, we can say much more.

**Lemma 3.3.** There is a constant $c$ such that, for all $\epsilon > 0$, sufficiently large $n$, a fixed ordering $\sigma$ on $[n]$, and random $T$ drawn from either $D_{yes}$ or $D_{no}$,

$$
\Pr[(1 - \epsilon)m < |B_T(\sigma)| < (1 + \epsilon)m] \geq 1 - 2^{-c\epsilon^2 n}.
$$

**Proof.** When $T \leftarrow D_{no}$, the random variable $|B_T(\sigma)|$ has binomial distribution $\text{Bin}(2m, \frac{1}{2})$, so the claimed bound is immediate.

Let $T \leftarrow D_{yes}$. Partition the edges of the tournament into perfect matchings $M_1,\ldots,M_{n-1}$. For each $i \in [n-1]$, let $X_i$ be the number of back edges of $T$ involving $M_i$, i.e.,

$$X_i = |\{(u,v) \in M_i : (u,v) \in B_T(\sigma) \text{ or } (v,u) \in B_T(\sigma)\}|.$$

Notice that $X_i \sim \text{Bin}(n/2, \frac{1}{2})$, whence

$$
\Pr[(1 - \epsilon)n/4 < X_i < \frac{1}{2}(1 + \epsilon)n/4] \geq 1 - 2^{-c\epsilon^2 n},
$$

for a certain constant $b$. By a union bound, the probability that all of the $X_i$s are between these bounds is at least $1 - (n-1)2^{-bc^2 n} \geq 1 - 2^{-c\epsilon^2 n}$, for suitable $c$. When this latter event happens, we also have $(1 - \epsilon)m < |B_T(\sigma)| = \frac{1}{2} \sum_{i=1}^n X_i < (1 + \epsilon)m$.

We define a $(q, \epsilon)$-query algorithm for a problem $P$ to be one that accepts an input digraph $G$ solely through queries to an $\epsilon$-oracle and, after at most $q$ such queries, outputs its answer to $P(G)$. We require this answer to be correct with probability at least $\frac{3}{4}$.

Now consider the particular oracle $O_{T,\epsilon}$, describing an $n$-vertex tournament $T$, that behaves as follows when queried on an ordering $\sigma$.

- If $(1 - \epsilon/2)m < |B_T(\sigma)| < (1 + \epsilon/2)m$, then return $m$.
- Otherwise, return $|B_T(\sigma)|$.

Clearly, $O_{T,\epsilon}$ is an $\epsilon$-oracle. The intuition in the next two proofs is that this oracle makes life difficult by seldom providing useful information.

**Proposition 3.2.** Every $(q, \epsilon)$-query algorithm for $\text{TOPO-SORT-T}$ makes $\exp(\Omega(\epsilon^2 n))$ queries.

**Proof.** WLOG, consider a $(q, \epsilon)$-query algorithm, $A$, that makes exactly $q$ queries, the last of which is its output. Using Yao’s minimax principle, fix $A$’s random coins, obtaining a deterministic $(q, \epsilon)$-query algorithm $A'$ that succeeds with probability $\geq \frac{3}{4}$ on a random tournament $T \leftarrow D_{yes}$. Let $\sigma_1,\ldots,\sigma_q$ be the sequence of queries that $A'$ makes when the answer it receives from the oracle to each of $\sigma_1,\ldots,\sigma_{q-1}$ is $m$.

Suppose that the oracle supplied to $A'$ is $O_{T,\epsilon}$. Let $E$ be the event that $A'$’s query sequence is $\sigma_1,\ldots,\sigma_q$ and it receives the response $m$ to each of these queries. For a particular $\sigma_i$,

$$
\Pr[O_{T,\epsilon}(\sigma_i) = m] = \Pr[(1 - \epsilon/2)m < |B_T(\sigma_i)| < (1 + \epsilon/2)m] \geq 1 - 2^{-bc\epsilon^2 n},
$$

for a suitable constant $b$, by Lemma 3.3. Thus, by a union bound, $\Pr[E] \geq 1 - q2^{-bc\epsilon^2 n}$. Unauthorized reproduction of this article is prohibited
When $E$ occurs, $A'$ must output $q$, but $E$ itself implies that $|B_T(q)| \neq 0$, so $A'$ errs. Thus, the success probability of $A'$ is at most $1 - \Pr[E] \leq q^2 - b \epsilon^2 n$. Since this probability must be at least $\frac{1}{3}$, we need $q \geq \frac{2}{3} \cdot 2^b \epsilon^2 n = \exp(\Omega(\epsilon^2 n))$. $\square$

**Proposition 3.3.** Every $(q, \epsilon)$-query algorithm for $\text{ACYC-T}$ makes $\exp(\Omega(\epsilon^2 n))$ queries.

**Proof.** We proceed similarly to Proposition 3.2, except that we require the deterministic $(q, \epsilon)$-query algorithm $A'$ to succeed with probability at least $\frac{1}{3}$ on a random $T \leftarrow \frac{1}{2}(D_{\text{yes}} + D_{\text{no}})$. We view $T$ as being chosen in two stages: first, we pick $Z \in U \{\text{yes, no}\}$ uniformly at random, then we pick $T \leftarrow D_Z$.

Define $\sigma_1, \ldots, \sigma_q$ and $E$ as before. So $\Pr[E] \geq 1 - q^2 - b \epsilon^2 n$. When $E$ occurs, $A'$ must output some fixed answer, either “yes” or “no.” We consider these cases separately.

Suppose that $A'$ outputs “no,” declaring that $T$ is not acyclic. Then $A'$ errs whenever $Z = \text{yes}$ and $E$ occurs. The probability of this is at least $\frac{1}{3} - q^2 - b \epsilon^2 n$, but it must be at most $\frac{1}{3}$, requiring $q = \exp(\Omega(\epsilon^2 n))$.

Suppose that $A'$ outputs “yes” instead. Then it errs when $Z = \text{no}$, $T$ is cyclic, and $E$ occurs. Since

$$\Pr[T \text{ acyclic} \mid Z = \text{no}] = n!/(2^Z) = \exp(-\Omega(n^2)),$$

we have $\frac{1}{3} \geq \Pr[A' \text{ errs}] \geq \frac{1}{3} - \exp(-\Omega(n^2)) - q^2 - b \epsilon^2 n$, requiring $q = \exp(\Omega(\epsilon^2 n))$. $\square$

**Theorem 3.4.** A $(q, \epsilon)$-query algorithm that gives a multiplicative approximation for either $\text{FAST}$ or $\text{FASIZE}$ must make $q = \exp(\Omega(\epsilon^2 n))$ queries.

**Proof.** This is immediate from Observation 1.1 and propositions 3.2 and 3.3. $\square$

## 4 Sink Finding in Tournaments

A classical offline algorithm for $\text{TOPO-SORT}$ is to repeatedly find a sink $v$ in the input graph (which must exist in a DAG), prepend $v$ to a growing list, and recurse on $G \setminus v$. Thus, $\text{SINK-FIND}$ itself is a fundamental digraph problem. Obviously, $\text{SINK-FIND}$ can be solved in a single pass using $O(n)$ space by maintaining an “is-sink” flag for each vertex. Our results below show that for arbitrary order streams this is tight, even for tournament graphs.

In fact, we say much more. In $p$ passes, on the one hand, the space bound can be improved to roughly $O(n^{1/p})$. On the other hand, any $p$-pass algorithm requires about $\Omega(n^{1/p})$ space. While these bounds don’t quite match, they reveal the correct asymptotics for the number of passes required to achieve polylogarithmic space usage: namely, $\Theta(\log n / \log \log n)$.

In contrast, we show that if the stream is randomly ordered, then using $\text{polylog}(n)$ space and a single pass is sufficient. This is a significant separation between the adversarial and random order data streams.

### 4.1 Arbitrary Order Sink Finding

**Theorem 4.1.** (Multi-pass algorithm) For all $p$ with $1 \leq p \leq \log n$, there is a $(2p - 1)$-pass algorithm for $\text{SINK-FIND}$ that uses $O(n^{1/p} \log(3p))$ space and has failure probability at most $1/3$.

**Proof.** Let the input digraph be $G = (V, E)$. For a set $S \subseteq V$, let $\max(S)$ denote the vertex in $S$ that has maximum in-degree. This can also be seen as the maximum vertex within $S$ according to the total ordering defined by the edge directions.

Our algorithm proceeds as follows.

- **Initialization:** Set $s = \lfloor n^{1/p} \ln(3p) \rfloor$. Let $S_1$ be a set of $s$ vertices chosen randomly from $V$.

  - **For $i = 1$ to $p - 1$:**
    - **During pass $2i - 1$:** Find $v_i = \max(S_i)$ by computing the in-degree of each vertex in $S_i$.
    - **During pass $2i$:** Let $S_{i+1}$ be a set of vertices randomly chosen from $\{u: (v_i, u) \in E\}$.

  - **During pass $2p - 1$:** Find $v_p = \max(S_p)$ by computing the in-degree of each vertex in $S_p$.

  For the sake of analysis, consider the quantity $\ell_i = |\{u: (v_i, u) \in E\}|$. Note that, for each $i \in [p]$,

$$\Pr[\ell_i > \ell_{i-1}/n^{1/p}] = (1 - 1/n^{1/p}) \leq \frac{1}{3p}.$$

Thus, by the union bound, $\ell_p = 0$ with probability at least $1 - p/(3p) = 2/3$. Note that $\ell_p = 0$ implies that $v_p$ is a sink. $\square$

We turn to establishing a multi-pass lower bound. Our starting point for this is the tree pointer jumping problem $\text{TP}_{k, t}$, which is a communication game involving $k$ players. To set up the problem, consider a complete ordered $k$-level $t$-ary tree $T$; we consider its root $z$ to be at level 0, the children of $z$ to be at level 1, and so on. We denote the $i$-th child of $y \in V(T)$ by $y_i$, the $j$-th child of $y_i$ by $y_{i, j}$, and so on. Thus, each leaf of $T$ is of the form $z_{i_1, \ldots, i_k}$ for some integers $i_1, \ldots, i_k \in [t]$.

An instance of $\text{TP}_{k, t}$ is given by a function $\phi: V(T) \rightarrow [t]$ such that $\phi(y) \in \{0, 1\}$ for each leaf.
Given an instance of level, we encode an instance of continue using the notations defined above. At a high
Proof. \[ \Omega(\cdot) \]
Theorem 4.2. \[ R^{-1}(TPJ_{k,t}) = \Omega(t/k) \]
Based on this, we prove the following lower bound.

**Theorem 4.3. (Multi-pass lower bound)** Any streaming algorithm that solves sink-find-T in \( p \) passes must use \( \Omega(n^{1/p}/p^2) \) space.

**Proof.** We reduce from \( TPJ_{k,t} \), where \( k = p + 1 \). We continue using the notations defined above. At a high level, we encode an instance of \( TPJ \) in the directions of edges in a tournament digraph \( G \), where \( V(G) \) can be viewed as two copies of the set of leaves of \( T \). Formally,

\[
V(G) = \{(i_1, \ldots, i_{k-1}, a) : \text{each } i_j \in [t] \text{ and } a \in \{0,1\}\}
\]

We assign each pair of distinct vertices \( u, v \in V(G) \) to a level in \( \{0, \ldots, k-1\} \) as follows. Suppose that \( u = (i_1, \ldots, i_k) \) and \( v = (i_1', \ldots, i_k') \). We assign \( \{u, v\} \) to level \( j-1 \), where \( j \) is the smallest index such that \( i_j \neq i_j' \). Given an instance of \( TPJ_{k,t} \), the players jointly create an instance of sink-find-T as follows. For each \( j \) from \( k-1 \) to 0, in that order, Player \( j \) assigns directions for all pairs of vertices at level \( j \), obtaining a set \( E_j \) of directed edges, and then appends \( E_j \) to a stream. The combined stream \( E_{k-1} \circ \cdots \circ E_1 = E_0 \) defines the tournament \( G \). It remains to define each set \( E_j \) precisely.

The set \( E_{k-1} \) encodes the bits \( \phi(y) \) at the leaves \( y \) of \( T \) as follows.

\[
E_{k-1} = \{(1,1-a), (a,1) \in V(G)^2 : \phi(z) = a\},
\]

Notice that if we ignore edge directions, \( E_{k-1} \) is a perfect matching on \( V(G) \).

Now consider an arbitrary level \( j \in \{0, \ldots, k-2\} \).

\[
\begin{align*}
&\pi_{i_1, \ldots, i_{j-1}, j}(1), \ldots, \pi_{i_1, \ldots, i_{j-1}}(t) \\
&= (1, \ldots, \ell - 1, \ell + 1, \ldots, t),
\end{align*}
\]

where \( \ell = \phi(z_{i_1, \ldots, i_{j-1}}) \).

Using this, we define \( E_j \) so as to encode the pointers at level \( j \) as follows.

\[
E_j = \{(i_1, i_{j-1}, i_j, i_{k}), (i_1, i_{j-1}, i_j', i_{k}) \in V(G)^2 : \pi_{i_1, \ldots, i_{j-1}, j}(1) \neq \pi_{i_1, \ldots, i_{j-1}, j}(t)\}
\]

It should be clear that the digraph \( (V(G), E_0 \cup E_1 \cup \cdots \cup E_{k-1}) \) is a tournament. We argue that it is acyclic. Suppose, to the contrary, that \( G \) has a cycle \( \sigma \). Let \( j \in \{0, \ldots, k-2\} \) be the smallest-numbered level of an edge on \( \sigma \). Then there exist \( h_1, \ldots, h_{j-1} \) such that every vertex on \( \sigma \) is of the form \( (h_1, \ldots, h_{j-1}, i_j, i_{k}) \). Let \( v^{(1)}, v^{(r)} \) be the vertices on \( \sigma \) whose outgoing edges belong to level \( j \). For each \( q \in [r] \), let \( v^{(q)} = (h_1, \ldots, h_{j-1}, i^{(q)}_j, i_{k}) \). Let \( \pi = \pi_{h_1, \ldots, h_{j-1}} \). According to eq. (4.5),

\[
\hat{\pi}^{-1}(1^{(1)}_j) < \hat{\pi}^{-1}(1^{(2)}_j) < \cdots < \hat{\pi}^{-1}(r^{(r)}_j) < \hat{\pi}^{-1}(1^{(1)}_j),
\]

a contradiction.

It follows that \( G \) has a unique sink. Let \( v = (h_1, \ldots, h_{k-1}, a) \in V(G) \) be this sink. In particular, for each level \( j \in \{0, \ldots, k-2\} \), all edges in \( E_j \) involving \( v \) must be directed towards \( v \). According to eq. (4.5), we must have \( \pi_{h_1, \ldots, h_{j-1}}(i_j) = t \), i.e., \( z_{h_1, \ldots, h_{j-1}}(t) = h_j \).

By eq. (4.4), this gives \( \phi(z_{h_1, \ldots, h_{j-1}}) = h_j \). Next, by eq. (4.2), this gives \( g(z_{h_1, \ldots, h_{j-1}}) = g(z_{h_1, \ldots, h_{j}}) \). Instantiating this observation for \( j = 0, \ldots, k-2 \), we have

\[
\begin{align*}
z_{h_1} &= g(z), \\
z_{h_1, h_2} &= g(z_{h_1}), \\
&\quad \ldots, \\
z_{h_1, \ldots, h_{k-1}} &= g(z_{h_1, \ldots, h_{k-2}}); \notag
\end{align*}
\]

i.e., \( z_{h_1, \ldots, h_{k-1}} = g^{(k-1)}(z) \).

At this point \( h_1, \ldots, h_{k-1} \) have been determined, leaving only two possibilities for \( v \). We now use the fact that the sole edge in \( E_{k-1} \) involving \( v \) must be directed towards \( v \). According to eq. (4.3), \( \phi(z_{h_1, \ldots, h_{k-1}}) = a \). Invoking eq. (4.2) again, \( a = \phi(g^{(k-1)}(z)) = g^{(k)}(z) = TPJ_{k,t}(\phi) \).

Thus, the players can read off the desired output \( TPJ_{k,t}(\phi) \) from the identity of the unique sink of the constructed digraph \( G \). Notice that \( n := |V(G)| = 2^{k-1} \). It follows that a \((k-1)\)-pass streaming algorithm...
for SINK-FIND-T that uses S bits of space solves TPJ$k,t$ in $k − 1$ rounds at a communication cost of $kS$. By Theorem 4.2, we have $S = \Omega((t/k^2) = \Omega(n^{1/(k−1)}/k^2)$.

### 4.2 Random Order Sink Finding

In this section we show that it is possible to find the sink of an acyclic tournament in one pass over a randomly order stream while using only polylog($n$) space. The algorithm we consider is as follows:

- **Initialization**: Let $S$ be a random set of $s = 200\log n$ nodes.
- For $i = 1$ to $k := \log_2 \left(\frac{m}{200000 n \log n}\right)$:
  - Ingest the next $c_i := 100 \cdot 2^i(n−1) \log n$ elements of the stream: For each $v \in S_i$ collect the set of edges $S_v$ consisting of all outgoing edges; throw away $S_v$ if it exceeds size $220 \log n$.
  - Pick any $v \in S_i$ such that $|S_v| = (200 \pm 20) \log n$ and let $S$ be the endpoints (other than $v$) of the edges in $S_v$.
- Ingest the next $m/1000$ elements: find $P$ the set of vertices $w$ such that there exists an edge $uw$ for some $u \in S$.
- Ingest the remaining $499m/500$ elements: Output any vertex in $P$ with no outgoing edges.

**Theorem 4.4.** There is a single pass algorithm for SINK-FIND-T that uses $O(\text{polylog } n)$ space and has failure probability at most $1/3$ under the assumption that the data stream is randomly ordered.

**Proof.** We refer to the $c_i$ elements used in the iteration $i$ as the $i$th segment of the stream. For a node $u$, let $X_{u,i}$ be the number of outgoing edges from $u$ amongst the $i$th segment. The following claim follows from the Chernoff bound:

**Claim 4.1.** With high probability, for all $u$ with $\left|rk(u) - n/2^i\right| \geq 0.2 \cdot n/2^i$ then

$$|X_{u,i} - 200 \log n| > 0.1 \cdot 200 \log n .$$

With high probability, for all $u$ with $\left|rk(u) - n/2^i\right| \leq 0.05 \cdot n/2^i$, then

$$|X_{u,i} - 200 \log n| < 0.1 \cdot 200 \log n .$$

If follows from the claim that if after processing the $i$th segment of the stream there exists a $v$ such that $|S_v| = (200 \pm 20) \log n$ then with high probability $rk(u) = (1 \pm 0.2) \cdot n/2^i$. We next need to argue that there exists such a $v$.

**Claim 4.2.** With high probability, for every node $u$ with $rk(u) = (1 \pm 0.2) \cdot n/2^i$, there exists an edge $uv$ in the $i$th segment such that $|rk(v) - n/2^i| \leq 0.05 \cdot n/2^i$.

**Proof.** There are at least $0.01/2^i$ such edges. The probability that none of them exists in the $i$th segment is at most $(1 − c_i/m)^{0.01 \cdot n/2^i} \leq 1/\text{poly}(n) .

The above two claims allow us to argue by induction that we will have an element $u$ with $rk(u) = (1\pm 0.2) \cdot n/2^i$ after the $i$th segment. At the end of the $k$th segment we have identified at least $(200 − 20) \log n$ vertices where every rank is at most $(1 + 0.2) \cdot n/2^k = O(\log n)$. With probability at least $1 − 1/\text{poly}(n)$ one of these vertices includes an edge to the sink amongst the $(k + 1)$ segment and hence the sink is in $P$ with high probability. There may be other vertices in $P$ but the following claim shows that we will identify any false positives while processing the final $499m/500$ elements of the stream.

**Claim 4.3.** With probability at least $1 − 1/499$, there exists at least once outgoing edge from every node except the sink amongst the last $499m/500$ elements of the stream.

**Proof.** [Proof of Claim] The probability no outgoing edge from the an element of rank $r > 0$ appears in the suffix of the stream is at most $(1 - 499/500)^r$. Hence, by the union bound the probability that there exists an element of rank $r > 0$ without an outgoing edge is at most $\sum_{r \geq 1}(1 - 499/500)^r = 1/499 .

This concludes the proof of Theorem 4.4. □

### 5 Topological Ordering in Random Graphs

We present results for computing a topological ordering of $G \sim \text{PlantDAG}_{n,q}$ (see Definition 1.1). We first present an $O(\log n)$-pass algorithm using $O(n^{4/3})$ space. We then present a one-pass algorithm that uses $O(n^{3/2})$ space and requires the assumption that the stream is in random order.

#### 5.1 Arbitrary Order Algorithm

In this section, we present two different algorithms. The first is appropriate when $q$ is large whereas the second is appropriate when $q$ is small. Combining these algorithms and considering the worst case value of $q$ yields the algorithm using $O(n^{4/3})$ space.

**Algorithm for large $q$.** The basic approach is to emulate QuickSort. We claim that we can find the relationship between any vertex $u$ among $n$ vertices and a predetermined vertex $v$ using three passes and
$O(n+q^{-3}\log n)$ space. Assuming this claim, we can sort in $O(\log(q^2n))$ passes and $O(n/q)$ space: we recursively partition the vertices and suppose at the end of a phase we have sub-problems of sizes $n_1, n_2, n_3, \ldots$. Any sub-problem with at least $1/q^2$ vertices is then sub-divided by picking $\Theta(\log n)$ random pivots (with replacement) within the sub-problems using the aforementioned three pass algorithm. There are at most $q^2n$ such sub-problems. Hence, the total space required partition all the sub-problems in this way is at most

$$O\left(\log n \sum_{i=1}^{q^2n} (n_i + q^{-3}\log n)\right) = O(nq^{-1}\log^2 n).$$

Note that the size of every sub-problem decreases by a factor at least 2 at each step with high probability and hence after $\log(q^2n)$ iterations, all sub-problems have at most $1/q^2$ vertices. Furthermore, each vertex degree is $O(1/q \cdot \log n)$ in each sub-problem. Hence, the entire remaining instance can be stored using $O(n/q \cdot \log n)$ space.

It remains to prove our three-pass claim. For this, we define the following families of sets:

$L_i = \{u : \exists \text{ u-to-v path of length } \leq i\},$

$R_i = \{u : \exists \text{ v-to-u path of length } \leq i\}.$

Using two passes and $O(n\log n)$ space we can identify $L_2$ and $R_2$ using $O(n\log n)$ space. Let $U$ be the set of vertices not contained in $L_2 \cup R_2$. The following lemma (which can be proved via Chernoff bounds) establishes that $L_2 \cup R_2$ includes most of the vertices of the graph with high probability.

**Lemma 5.1.** With high probability, $|U| = O(q^{-2}\log n)$.

In a third pass, we store every edge between vertices in $U$ and also compute $L_3$ and $R_3$. Computing $L_3$ and $R_3$ requires only $O(n\log n)$ space. There is an edge between each pair of vertices in $U$ with probability $q$ and hence, the expected number of edges between vertices in $U$ is at most $q|U|^2 = O(q^{-3}\log^2 n)$. By an application of the Chernoff Bound, this bound also holds w.h.p. Note that $L_3, R_3$, and the edges within $U$ suffice to determine whether $u \in L_\infty$ or $u \in R_\infty$ for all $u$. To see this first suppose $u \in L_\infty$ and that $(u, w)$ is the critical edge on the directed path from $u$ to $v$. Either $w \in L_2$ and therefore we deduce $u \in L_3$; or $u \in L_2$; or $u \notin L_2$ and $w \notin L_2$ and we therefore store the edge $(u, w)$.

This establishes the following lemma.

**Lemma 5.2.** There is an $O(\log n)$-pass, $O(n/q)$-space algorithm for topo-sort on a random input graph $G \sim \text{PlantDAG}_{n, q}$.  

### Algorithm for small $q$

We use only two passes. In the first pass, we compute the in-degree of every vertex. In the second, we store all edges between vertices where the in-degrees differ by at most $3\sqrt{cnq \cdot \ln n}$ where $c > 0$ is a sufficiently large constant.

**Lemma 5.3.** There is a two-pass, $\tilde{O}(n^{3/2}/\sqrt{q})$-space algorithm for topo-sort on a random input graph $G \sim \text{PlantDAG}_{n, q}$.

**Proof.** We show that, with high probability, the above algorithm collects all critical edges and furthermore only collects $\tilde{O}(n^{3/2}/\sqrt{q})$ edges in total. Let $u$ be the element of rank $r_u$. Note that $d_{\text{in}}(u)$ has distribution $1+\text{Bin}(r_u-2, q)$. Let $X_u = d_{\text{in}}(u)-1$. By an application of the Chernoff Bound,

$$\Pr\left[|X_u - (r_u - 2)q| \geq \sqrt{c(r_u - 2)q \ln n}\right] \leq 1/\text{poly}(n).$$

Hence, w.h.p., $r_u = 2 + X_u/q \pm \sqrt{c n/q \cdot \ln n}$ for all vertices $u$. Therefore, if $(u, v)$ is critical, then

$$|X_u - X_v| \leq |X_u - (r_u - 2)q| + \sqrt{c n/q \cdot \ln n}$$

$$\leq 3\sqrt{c nq \cdot \ln n}.$$ 

This ensures that the algorithm collects all critical edges. For the space bound, we first observe that for an arbitrary pair of vertices $u$ and $v$, if $|X_u - X_v| \leq 3\sqrt{c n q \cdot \ln n}$ then

$$|r_u - r_v| \leq |X_u - X_v|/q + 2\sqrt{c n/q \cdot \ln n} \leq 8\sqrt{c n/q \cdot \ln n}.$$ 

Hence, we only store an edge between vertex $u$ and $v$ whose rank differs by at most $8\sqrt{c n/q \cdot \ln n}$. Since edges between such vertices are present with probability $q$, the expected number of edges stored incident to $u$ is $8\sqrt{c n q \cdot \ln n}$ and is $O(\sqrt{nq \cdot \ln n})$ by an application of the Chernoff bounds. Across all vertices this means the number of edges stored is $O(n^{3/2}/\sqrt{q} \cdot \ln n)$ as claimed.

Combining Lemma 5.2 and Lemma 5.3 yields the main theorem of this section.

**Theorem 5.1.** There is an $O(\log n)$-pass algorithm for topo-sort on a random input $G \sim \text{PlantDAG}_{n, q}$ that uses $\tilde{O}\left(\min\{n/q, n^{3/2}/\sqrt{q}\}\right)$ space. For the worst-case over $q$, this is $\tilde{O}(n^{4/3})$.  

#### 5.2 Random Order Algorithm

The transitive reduction of a DAG $G = (V, E)$ is the minimal subgraph $G^\text{red} = (V, E')$ such that, for all
of the prefix.

The one-pass algorithm assuming a random ordering of the edges is simply to maintain \( G^{\text{red}} \) as \( G \) is streamed in, as follows. Let \( S \) be initially empty. For each edge \((u,v)\) in the stream, we add \((u,v) \) to \( S \) and then remove all edges \((u',v')\) where there is a \( u' \)-to-\( v' \) path among the stored edges.

**Theorem 5.2.** There is a one-pass algorithm that uses \( \tilde{O}(\max_q n \min\{n/q, n^2 q\}) \) space and solves \textsc{topo-sort} on an input \( G \sim \text{PlantDAG}_{n,q} \) presented in random order. In the worst case this space bound is \( O(n^3/2) \).

**Proof.** Consider the length-\( T \) prefix of the stream where the edges of \( G \) are presented in random order. It will be convenient to write \( T = n^2 q \). We will argue that the number of edges in the transitive reduction of this prefix is \( O(\min\{n/q, n^2 q\}) \) with high probability; note the bound \( n^2 q \) follows trivially because the transitive reduction has at most \( T \) edges. The result then follows by taking the maximum over all prefixes.

We say an edge \((u,v)\) of \( G \) is short if the difference between the ranks is \( r_v - r_u \leq \tau := cq^{-2} \log n \) where \( c \) is some sufficiently large constant. An edge that is not short is defined to be long. Let \( S \) be the number of short edges in \( G \) and let \( M \) be the total number of edges in \( G \). Note that \( \mathbb{E}[S] \leq (n - 1) + q r n \) and \( \mathbb{E}[M] = (n - 1) + q \binom{n}{2} \). By the Chernoff bound, \( S \leq 2 q r n \) and \( n^2 q/4 \leq M \leq n^2 q \) with high probability. Furthermore, the number of short edges in the prefix is expected to be \( T \cdot S/M \) and, with high probability, is at most

\[
2T \cdot S/M \leq \frac{4 T q r n}{n^2 q/4} = 16 cn/q \cdot \log n.
\]

Now consider how many long edges are in the transitive reduction of the prefix. For any long edge \((u,v)\), let \( X \) denote the event that \((u, v), (w, v)\) are both in the prefix. Note that the variables \( \{X_w\}_{w: r_u + 1 \leq r_w \leq r_v - 1} \) are negatively correlated and that

\[
\Pr[X_w = 1] \geq \left(\frac{q T / M}{2}\right)^2/2 \geq \frac{q^2}{2}.
\]

Hence, if \( X = \sum_{w: r_u + 1 \leq r_w \leq r_v - 1} X_w \) then

\[
\mathbb{E}[X] \geq c q^{-2} \log n \cdot q^2/2 = c/2 \cdot \log n
\]

and so, by the Chernoff bound, \( X > 0 \) with high probability and if this is the case, even if \((u,v)\) is in the prefix, it will not be in the transitive reduction of the prefix. Hence, by the union bound, with high probability no long edges exist in the transitive closure of the prefix.

6 **Rank Aggregation**

Recall the \textsc{rank-aggr} problem and the distance \( d \) between permutations, defined in Section 1.2. To recap, the distance between two orderings is the number of pairs of objects which are ranked differently by them, i.e.,

\[
d(\pi, \sigma) := \sum_{a,b \in [n]} \mathbb{I}\{\pi(a) < \pi(b), \sigma(b) < \sigma(a)\}.
\]

Note that \textsc{rank-aggr} is equivalent to finding the median of a set of \( k \) points under this distance function, which can be shown to be metric. It follows that picking a random ordering from the \( k \) input orderings provides a 2-approximation for \textsc{rank-aggr}.

A different approach is to reduce \textsc{rank-aggr} to the weighted feedback arc set problem on a tournament. This idea leads to a \((1 + \varepsilon)\)-approximation via \( \ell_1 \)-norm estimation in a way similar to the algorithm in Section 3.1. Define a vector \( x \) of length \( \binom{n}{2} \) indexed by pairs of vertices \( \{a,b\} \) where

\[
x_{a,b} = \sum_{i=1}^{k} \mathbb{I}\{\sigma_i(a) < \sigma_i(b)\},
\]

i.e., the number of input orderings that have \( a < b \). Then for any ordering \( \pi \) define a vector \( y^\pi \), where for each pair of vertices \( \{a,b\} \),

\[
y^\pi_{a,b} = k \cdot \mathbb{I}\{\pi(a) < \pi(b)\}.
\]

It is easy to see that \( \|x - y^\pi\|_1 = \text{cost}(\pi) \).

As in Section 3.1, our algorithm maintains an \( \ell_1 \)-sketch \( S x \) with accuracy \( \varepsilon/3 \) and error \( \delta = 1/(3 \cdot n!) \). By Fact 3.1, this requires at most \( O(\varepsilon^{-2} n \log^2 n) \) space. In post-processing, the algorithm considers all \( n! \) permutations \( \pi \) and, for each of them, computes \( S(x - y^\pi) = S x - S y^\pi \). It thereby recovers an estimate for \( \|x - y^\pi\|_1 \) and finally outputs the ordering \( \pi \) that minimizes this estimate.

The analysis of this algorithm is essentially the same as in Theorem 3.1. Overall, we obtain the following result.

**Theorem 6.1.** There is a one-pass algorithm for rank aggregation that uses \( O(\varepsilon^{-2} n \log^2 n) \) space, returns a \((1 + \varepsilon)\)-approximation with probability at least \( \frac{2}{3} \), but requires exponential post-processing time.

**Acknowledgements**

We thank Riko Jacob for a helpful discussion about the sink finding problem. We thank the anonymous SODA 2020 reviewers for several helpful comments that improved the presentation of the paper.
References


