Universally Optimal Streaming Algorithm for Random Walks in Dense Graphs

Klim Efremenko[*]
Ben-Gurion University

Gillat Kol[†]
Princeton University

Raghuvansh R. Saxena[‡]
Tata Institute of Fundamental Research

Zhijun Zhang[§]
INSAIT, Sofia University "St. Kliment Ohridski"

Abstract

Sampling a random walk is a fundamental primitive in many graph applications. In the streaming model, it is known that sampling an L-step random walk on an n-vertex directed graph requires $\Omega(nL)$ space, implying that no sublinear-space streaming algorithm exists for general graphs.

We show that sublinear algorithms are possible for the case of dense graphs, where every vertex has out-degree at least $\Omega(n)$. In particular, we give a one-pass turnstile streaming algorithm that uses only $\tilde{\mathcal{O}}(L)$ memory for such graphs. More broadly, for graphs with minimum out-degree at least d, our streaming algorithm samples a random walk using $\tilde{\mathcal{O}}(\frac{n}{d} \cdot L)$ memory.

We show that our algorithm is optimal in a strong "beyond worst-case" sense. To formalize this, we introduce the notion of *universal optimality* for graph streaming algorithms. Informally, a streaming algorithm is universally optimal if it performs (almost) as well as possible on *every* graph, assuming a worst-case choice of the streaming order. This notion of universal optimality is a key conceptual contribution of our work.

Contents

1 Introduction

1.1 Background and Motivation

Sampling a random-walk in a graph is a fundamental problem with broad algorithmic applications. For instance, it underlies techniques for connectivity testing [Rei08], clustering [ACL07, AP09, COP03, ST13], sampling [JVV86], constructing random spanning trees [Sch18], and approximate counting [JS89]. Another well-known application of random walk sampling is for *PageRank* estimation [SGP11], where the rank of a webpage corresponds to the likelihood that a user, randomly following hyperlinks, arrives at that page[1]. Since these applications often involve extremely large, real-world networks, an important challenge is to develop space-efficient *graph-streaming algorithms* that process the graph edge-by-edge rather than storing it explicitly.

The folklore algorithm. A well-known folklore streaming algorithm for simulating an L-step random walk on an n-vertex directed graph from a given start vertex v_0 uses $\tilde{\mathcal{O}}(nL)$ memory[2]. The algorithm works as follows: As the graph's edges arrive in a stream, maintain for each vertex v a list of L outgoing edges selected independently and uniformly at randomly. To generate the walk, begin at v_0 and, at step $i \in [L]$, move to a new vertex v_i by taking the first sampled edge that haven't been taken before from the precomputed list of outgoing edges at v_{i-1}.[3]

The folklore algorithm was later shown to be optimal [Jin19]; however, the hard instances used by the lower-bound proof crucially involve vertices of very low out-degree. This naturally raises the question of whether better algorithms might exist for *dense graphs*, which we define as graphs in which every vertex has out-degree at least d for some large $d \geq 1$.[4] We use the *minimum out-degree d* as our measure of density rather than, say, the average degree (or equivalently, the total number of edges), as, for instance, the graph formed by the union of a clique and a sparse subgraph may still have a high average out-degree. Yet, if v_0 is in the sparse subgraph, then sampling a random walk starting from v_0 requires sampling a random walk in the sparse subgraph.

---

[1]Formally, in a web graph $G = (V, E)$, PageRank values satisfy the system $\mathsf{PageRank}(u) = \sum_{(v,u) \in E} \mathsf{PageRank}(v)/d_{\mathsf{out}}(v)$ for every vertex u, where $d_{\mathsf{out}}(v)$ denotes the out-degree of v [BP98].

[2]Throughout, the notation $\tilde{\mathcal{O}}(\cdot)$ hides polylogarithmic factors in $\max(n, L)$. Algorithms are assumed to know n in advance, but not the length of the stream (the number of edges).

[3]For example, if the first two sampled edges from v_{i-1} have already been used, take the third. A single edge (v, u) may appear and be traversed multiple times if it occurs repeatedly in v's stored list of outgoing edges.

[4]We assume $d \geq 1$, since if a vertex (e.g., v_0) has out-degree 0, there may not be any paths of length L starting from v_0.

1.2 Our Results

1.2.1 Random Walk Sampling for Dense Graphs

The algorithm $\pi_{d,L}^{\mathsf{sparse}}$. We begin by noting that if the input graph has minimum out-degree at least d and $\frac{L}{d}$ is large, the folklore algorithm can be improved to store only $\tilde{\mathcal{O}}\big(\lceil\frac{L}{d}\rceil \cdot n\big)$ edges. Specifically, instead of storing L outgoing edges per vertex, $\pi_{d,L}^{\mathsf{sparse}}$ stores $\tilde{\mathcal{O}}\big(\lceil\frac{L}{d}\rceil\big)$ outgoing edges for each vertex. We show that this is enough because, with high probability, no vertex is visited more than that many times during an L-step random walk on a graph with minimum out-degree at least d. We call this algorithm $\pi_{d,L}^{\mathsf{sparse}}$.

The *distortion* of a random walk sampling algorithm is the maximum, over all graphs G, of the statistical distance between the true distribution of random walk on G and the distribution over outputs produced by the algorithm on input G. The distortion of $\pi_{d,L}^{\mathsf{sparse}}$ is small but nonzero, since, for example, it never outputs the rare walks in which a vertex is visited more than $\lceil\frac{L}{d}\rceil$ times.

The algorithm $\pi_{d,L}^{\mathsf{dense}}$. Since $\pi_{d,L}^{\mathsf{sparse}}$ must store at least one sampled edge per vertex (and we assume $\frac{L}{d}$ is large), it never achieves sublinear space. We therefore propose a second, fundamentally different, streaming algorithm, $\pi_{d,L}^{\mathsf{dense}}$, which, when d is sufficiently large (relative to L), only uses sublinear memory.

The algorithm $\pi_{d,L}^{\mathsf{dense}}$ proceeds as follows: Set $V_0 = \{v_0\}$. Sample L sets of vertices (layers of a graph) $V_1, \ldots, V_L$, each of size $\tilde{\mathcal{O}}(\lceil\frac{n}{d}\rceil)$ (if a vertex is sampled in multiple layers, we treat each occurrence as a distinct copy of the original vertex). Intuitively, V_i represents the possible choices for vertex v_i. Then, for each $i \in \{0, \ldots, L-1\}$ and $v \in V_i$, the algorithm samples a random outgoing edge from v into V_{i+1}, thereby constructing an L-layer graph. The output is the unique path of length L starting from v_0 in this layered graph.

It is not hard to verify the correctness of $\pi_{d,L}^{\mathsf{dense}}$: $\pi_{d,L}^{\mathsf{dense}}$ fails in the event that some vertex in a layer has no outgoing edge to the next layer. However, the sizes of the layers are selected such that this event happens with low probability. If the algorithm does not fail and the path $(v_0, v_1, \ldots, v_L)$ is sampled, we claim that each v_{i+1} is a random (outgoing) neighbor of v_i, and therefore that the output is a random walk. Indeed, since V_{i+1} is a random subset of vertices (of a specific size), choosing a random neighbor of v_i within V_{i+1} is equivalent to selecting a random neighbor of v_i in V. Additionally, the algorithm only stores $\tilde{\mathcal{O}}(\lceil\frac{n}{d}\rceil \cdot L)$ edges, and, in particular, on graphs with minimum out-degree $d = \Omega(n)$, its space usage reduces to $\tilde{\mathcal{O}}(L)$, which can be sublinear in n.

Note that $\pi_{d,L}^{\mathsf{dense}}$ enjoys the following property: It fails with a small probability, but, conditioned on non-failure, it samples from the true distribution of random walks. $\pi_{d,L}^{\mathsf{sparse}}$ does not satisfy this property.

1.2.2 Streaming Universal Optimality

The notion of streaming universal optimality. $\pi_{d,L}^{\mathsf{dense}}$ is essentially optimal for very dense graphs with $d = \Omega(n)$ as it uses only $\tilde{\mathcal{O}}(L)$ memory and the output consists of L edges. But is $\pi_{d,L}^{\mathsf{dense}}$ optimal for all values of d?

A worst-case optimality result for a graph streaming algorithm would identify a family of hard graph instances (on which the algorithm uses the most memory) and show that no other correct algorithm can use less memory on these instances. However, for easier instances, the algorithm may be highly suboptimal. This notion of optimality can be viewed as *existential* optimality: There exist inputs within the considered class for which the algorithm is optimal. A much more ambitious goal is to seek *universal* optimality, where the algorithm achieves optimal performance on *every* instance.

Universal optimality [GKP98], along with other "beyond worst-case" notions, has been studied for decades in various computational models, see the great book [Rou20]. In this work, we introduce the notion of universal optimality for streaming algorithms. Informally, a streaming algorithm is *universally optimal* if it performs (nearly) as well as possible on *every* graph under *worst-case* choice of streaming order. Specifically, for any graph G, if the algorithm uses S space on some streaming order of G's edges, then *every* other correct algorithm must also use at least (approximately) S space on some (possibly different) streaming order of G. Further discussion of the notion of universal optimality for streaming algorithms appears in Section 1.2.3.

Universally optimal random walk samplers? A priori, it is not clear that universally optimal algorithms should even exist for the task T of sampling an L-step random walk with distortion, say, 0.1, over the set of graphs $\mathcal{G}$ consisting of directed graphs with minimum out-degree at least d. Indeed, it is not hard to see that neither $\pi_{d,L}^{\mathsf{sparse}}$ nor $\pi_{d,L}^{\mathsf{dense}}$ satisfies such universal optimality for all values of d and L.[5]

Nevertheless, we show that with a careful implementation, $\pi_{d,L}^{\mathsf{sparse}}$ and $\pi_{d,L}^{\mathsf{dense}}$ can be combined into a universally optimal algorithm $\pi_{d,L}$ as follows: If $\frac{L}{d}$ is greater than (roughly) $\log n$, run $\pi_{d,L}^{\mathsf{sparse}}$; otherwise, run $\pi_{d,L}^{\mathsf{dense}}$.

Theorem 1.1. *For $d, L \in \mathbb{N}$, $\pi_{d,L}$ solves the task of sampling an L-step random walk, with distortion 0.1, from a given vertex in any directed graph with minimum out-degree d. It uses at most $\tilde{\mathcal{O}}\big(\sum_v \min(d_{\mathsf{out}}(v), \frac{L}{d})\big) \leq \tilde{\mathcal{O}}\big(\frac{n}{d} \cdot L\big)$ space.*

Furthermore, it is universally optimal over the set of directed graphs with minimum out degree at least d, for any $d, L \geq \log^{10} n$.

We note that $\pi_{d,L}$ does not work for dynamic streams, since it relies on reservoir sampling [Vit85] to select the outgoing edges to be stored. Nevertheless, by replacing reservoir

---

[5]For $d = \Omega(n)$ and $L = \sqrt{n}$, $\pi_{d,L}^{\mathsf{sparse}}$ uses $\Omega(n)$ memory, while $\pi_{d,L}^{\mathsf{dense}}$ uses $\tilde{\mathcal{O}}(\sqrt{n})$, so $\pi_{d,L}^{\mathsf{sparse}}$ is not universally optimal. If $L = \sqrt{n}$ and $\sqrt{n}$ vertices have out-degree $\Omega(n)$ while the rest have constant out-degrees, d is constant. In this case, $\pi_{d,L}^{\mathsf{dense}}$ uses $\Omega(n^{1.5})$ memory, whereas $\pi_{d,L}^{\mathsf{sparse}}$ uses only $\tilde{\mathcal{O}}(n)$ (as each of $n - \sqrt{n}$ constant out-degree vertices only stores a constant number of edges), showing that $\pi_{d,L}^{\mathsf{dense}}$ is not universally optimal.

sampling with ℓ_0-samplers [JST11], we can obtain a new algorithm that works for dynamic streams and uses $\tilde{\mathcal{O}}(\frac{n}{d} \cdot L)$ space. However, this algorithm is not (insertion-only) universally optimal with a straightforward implementation[6]. In Section 1.2.3, we will show that this turns out to be the best we can do.

As pointed out in [GKP98], an interesting "side-effect" of universal optimality is that it precisely identifies the parameters of the problem that are inherently responsible for its complexity. In our case, the minimum out-degree d of the graph dictates the optimal space complexity (at least when d is known to the algorithm and within certain parameter regimes). In particular, once d is specified, the optimal space requirement no longer depends on other parameters (such as the average out-degree, the minimum total degree, or the length of the longest directed path in the graph).

Do we need to know d? The algorithm $\pi_{d,L}$ requires advance knowledge of d. A natural question is whether one can design a one-pass streaming algorithm with comparable space usage that does not rely on knowing d beforehand[7].

Specifically, let d be sufficiently large for Theorem 1.1 to hold, and let π_L be an algorithm for sampling an L-step random walk in any graph with minimum out-degree at least d (but unknown to π_L). Let G be a graph with minimum out-degree $d' = \Omega(n)$. Running $\pi_{d,L}$ on G requires $\Omega(\frac{n}{d} \cdot L)$ space. By the universal optimality of $\pi_{d,L}$ guaranteed by Theorem 1.1, it follows that any algorithm that works for all graphs with minimum out-degree at least d, including π_L, must also use $\tilde{\Omega}(\frac{n}{d} \cdot L)$ space on some streaming order of G. In contrast, $\pi_{d',L}$ requires only $\tilde{\mathcal{O}}(L)$ space on every streaming order of G. Hence, algorithms that do not know d in advance cannot achieve the same space as $\pi_{d',L}$.

1.2.3 Further Discussion of the Notion of Universal Optimality

Universal optimality for dynamic streaming algorithms. Our definition of universal optimality considers an algorithm τ that works for *insertion-only streams*, and, for every graph, compares its space usage to that of any other algorithm τ' that works for *insertion-only* streams. One can consider the same definition where τ and τ' work for *dynamic* streams. However, we observe that notion of universal optimality for dynamic streaming algorithms coincides with standard worst-case optimality.

To see this, suppose τ is a worst-case optimal algorithm for some task T on dynamic streams. Fix n. Let S denote the maximum memory used by τ on graphs with n vertices. By (worst-case) optimality of τ, for every other algorithm τ' that solves T, there exists a stream of edge updates $E_{\tau'}$ on n vertices such that τ' requires at least S memory on $E_{\tau'}$. Now take any graph G' on n vertices and consider an algorithm τ' for T. Construct the

---

[6]Intuitively, it is possible that a vertex v has $\Omega(L/d)$ outgoing edges during the stream while its actual out-degree is very small at the end. For $\pi_{d,L}^{\mathsf{sparse}}$, this means $\Omega(L/d)$ ℓ_0 samplers are needed. In contrast, if only insertion-only streams are considered, $\pi_{d,L}^{\mathsf{sparse}}$ can simply store all outgoing edges if $d_{\mathsf{out}}(v) < L/d$.

[7]This can be done in just two passes: the first pass computes the minimum out-degree d, and the second pass runs $\pi_{d,L}$ using the degree d computed in the first pass.

following stream for G': First insert/delete edges according to $E_{\tau'}$, then reverse all of them, and finally insert the edges of G'. Since τ' already uses at least S memory on the prefix $E_{\tau'}$, it uses at least S memory on this streaming order of G'. In contrast, τ never exceeds S memory on any stream. Hence, τ is also universally optimal.

Universal optimality vs. instance optimality. Our definition of universal optimality requires the algorithm to performs (nearly) as well as any competitor on every graph G, when comparing their worst-case memory usage across all possible streaming orders of G. However, this leaves open the possibility that on specific streaming orders of G, a competing algorithm could do substantially better. A stronger requirement would be to guarantee optimality on every *stream*, not just every graph. Concretely, for any edge stream I, the memory used by any competing algorithm on I must be at least as large as that used by our algorithm on the same I. This stronger guarantee corresponds to adapting the notion of *instance optimality*, where an algorithm must be optimal on each individual instance, to the context of graph streaming algorithms [FLN01] (see also [ABC17, VV17]).

However, it turns out that instance optimality makes little sense. Suppose $d = \sqrt{n}$ and $L = n$. Consider an n-vertex graph G which is the disjoint union of $\sqrt{n}$ cliques of size $\sqrt{n}$ each, along with the input stream where edges are given in lexicographical order. We can construct an algorithm τ which checks if the input stream is exactly the edges of G given in lexicographical order. This can be done in only $\tilde{\mathcal{O}}(1)$ additional space because the edges of G can be easily generated in lexicographical order in such space. As soon as the check fails, τ simulates $\pi_{d,L}$ from the beginning of the stream. This is possible because the prefix of the stream can be regenerated. If the check passes at the end, then an L-step random walk can be easily generated in $\tilde{\mathcal{O}}(L)$ space as we know all the edges (implicitly). Clearly, τ always performs as well as $\pi_{d,L}$. Furthermore, τ is strictly better than $\pi_{d,L}$ on the specific instance of G and a lexicographical order of the edges of G.

More generally, the above "hard-coding" method can apply to any graph problem, if we can design a more space-efficient algorithm tailored to specific instances that have succinct representations and can be checked in small space. However, note that such things cannot happen with universal optimality as there is no way to check, using small space, if the input graph falls into certain categories.

1.3 Additional Related Work

In an influential work, [SGP11] give an $\tilde{\mathcal{O}}(\sqrt{L})$-pass, $\tilde{\mathcal{O}}(n)$-space algorithm for simulating L-step random walks on directed graphs, and used it to derive space-efficient algorithms for estimating *PageRank* on graph streams. For one-pass streaming, [Jin19] proved an $\Omega(nL)$ space lower bound, thereby proving the optimality of the folklore algorithm. For undirected graphs, [Jin19] also gave an $\mathcal{O}(n \cdot \sqrt{L})$-space algorithm and a matching lower bound for sufficiently large L. Recently, [CKP+21] showed that the space complexity of random walk sampling on directed graphs is $\tilde{\Theta}(n\sqrt{L})$ when *two passes* over the stream are allowed. Addi-

tionally, [KKP22, CKKP22] studied the random walk sampling problem in the *random-order* streaming model.

[BDV18] studies sublinear algorithms for MAXCUT and correlation clustering. For dense graphs, constant-space algorithms are known, whereas for sparse graphs, $\Omega(n)$ space lower bounds hold in the streaming model. Their work bridges these two extremes by providing algorithms whose performance is parameterized by the average degree.

2 Proof Overview

We overview the proof of Theorem 1.1 in this section. As the algorithm mentioned was already described in Section 1, we mostly focus on the proof of universal optimality. However, we do give some extra intuition for the algorithm $\pi_{d,L}^{\mathsf{dense}}$.

2.1 The Sampling Algorithm $\pi_{d,L}^{\mathsf{dense}}$

Just like $\pi_{d,L}^{\mathsf{sparse}}$, our algorithm $\pi_{d,L}^{\mathsf{dense}}$ is based on the folklore algorithm but develops it from a different perspective than $\pi_{d,L}^{\mathsf{sparse}}$. Recall that the standard view of the folklore algorithm interprets its memory as a set $\{L_v\}_{v \in V}$ of n lists, where each L_v stores L sampled outgoing edges from v. The algorithm $\pi_{d,L}^{\mathsf{sparse}}$ is obtained by simply shortening these lists by a factor of d and a proof that such a shortening will not affect the correctness of the algorithm.

For the algorithm $\pi_{d,L}^{\mathsf{dense}}$, we reinterpret the memory of the folklore algorithm as maintaining an L-layered graph rather than a collection of lists. Each layer contains a copy of every vertex v, and the (only) outgoing edge of v in layer i points to some vertex in layer $i + 1$. Specifically, if the i-th edge in L_v is (v, u), then the copy of v in layer i has a single outgoing edge to the copy of u in layer $i + 1$. This perspective is equivalent to viewing the folklore algorithm's memory as an $n \times L$ matrix of edges, where the row corresponding to vertex $v \in V$ encodes L_v. In this representation, the i-th column specifies the edges connecting layer i to layer $i + 1$ in the layered graph.

In this matrix (or layered-graph) view, $\pi_{d,L}^{\mathsf{sparse}}$ corresponds to restricting the number of columns (equivalently, the number of layers) from L to $\frac{L}{d}$. By contrast, our algorithm $\pi_{d,L}^{\mathsf{dense}}$ reduces the number of *rows* (equivalently, the size of each layer) from n to $\frac{n}{d}$, using the guarantee on the minimum out-degree to show that such a reduction does not affect the correctness of the algorithm. It is easy to see that this memory restriction of the folklore algorithm yields exactly the algorithm $\pi_{d,L}^{\mathsf{dense}}$ described in Section 1.2.1.

The property testing approach. Prior work has sometimes been motivated by a query model perspective to design algorithms for dense graphs. As such a perspective also yields an algorithm for our problem, we outline it here. However, the algorithm so obtained turns out to be suboptimal.

In the query model, a random walk can be efficiently simulated if adaptivity is allowed. Starting from v_0, we select a random neighbor v_1 by querying random entries in the adjacency row of v_0 until a 1 is found, then proceed similarly to select a neighbor v_2 of v_1, and so on. Consider, for instance, the case where minimum degree at least $d = \Omega(n)$. In this case, the expected number of queries needed to find the next vertex is constant. Consequently, the entire process requires roughly $\Theta(L)$ queries in expectation.

To obtain a one-pass streaming algorithm, we aim to transform the above adaptive query algorithm into a non-adaptive one, which are easily simulated by a streaming algortihm. The beautiful technique of [GT03] shows how to convert an adaptive query algorithm with q queries into a non-adaptive one by sampling an induced subgraph on $\mathcal{O}(q)$ vertices, which requires $\mathcal{O}(q^2)$ queries.[8] However, applying this idea directly to our adaptive random walk sampling algorithm yields a memory complexity of $\tilde{\mathcal{O}}(L^2)$ when $d = \Omega(n)$, which is not optimal as it is worse than the bound we get.

2.2   Universal Optimality

We now overview the proof that our algorithm is universally optimal. For this, we first describe a lower bound approach designed for directed bipartite graphs that has been helpful in prior works as well. Consider a directed bipartite graph $G = (V_1 \cup V_2, E)$ and an integer $d > 0$ such that every vertex $v \in V_1$ has out-edges to an independent uniformly random subset of size d_{out} of vertices in V_2. The vertices in V_2 have only one out-edge and all these edges lead to the same vertex $z \in V_1$ that is chosen uniformly at random.

From a "sampling" problem to a "search" problem.   Because all the vertices in V_2 lead to the same vertex z, a random walk starting from any vertex in V_2 would first go to z, then take a uniformly random edge going out of z, then go to z again, and so on for L steps. This means that any such walk contains within it, a sequence of $L/2$ edges going out of z chosen independently and uniformly at random. As the number of distinct out-edges of z is d_{out}, we get that with high probability at least $\Omega(\min(d_{\mathsf{out}}, L))$ of these edges are distinct. We conclude that any algorithm that samples a random walk in this graph can also be used to solve a search problem, where the goal is to identify a subset of edges of size $\Omega(\min(d_{\mathsf{out}}, L))$ coming out of z, with high probability.

Lower bound for the search problem.   However, it can be shown that the search problem is hard for streaming algorithms. Indeed, consider what happens when the edges in the graph are presented in a stream with all the edges going out of the vertices in V_1 appearing first in the stream followed by the edges going out of the vertices in V_2. In this case, when the algorithm is processing the edges going out of the vertices in V_1, it has no information

---

[8]While [GT03] state their result in the context of converting adaptive graph property testers into non-adaptive ones, their technique can be applied to convert the adaptive random walk algorithm above into a non-adaptive version.

about the vertex z for which it has to remember the out-edges. Thus, unless the algorithm records enough out-edges from a lot of vertices in V_1, it is highly likely that when z is chosen uniformly at random from the vertices in V_1, the algorithm would not have stored enough edges coming out of z.

In other words, in order to output $\Omega(\min(d_{\mathsf{out}}, L))$, the algorithm actually needs to remember this number of edges from all the vertices in V_1, thereby needing at least $\Omega(|V_1| \cdot \min(d_{\mathsf{out}}, L))$ memory. Observe that this proof is an adaptation of the lower bound proof for the well-known INDEX problem.

Further discussion and extensions of this approach. Note that it is important in the above proof that every vertex in V_1 has the same number d_{out} of edges going out of it. In case the number of out-edges going out of different vertices in V_1 is vastly different, the algorithm can focus on remembering only the edges from vertices that have few out-edges, and hope that when z is sampled it happens to be one of these vertices. This could mean a worse space lower bound.

Also, observe that the instances considered in this lower bound approach are such that all vertices in V_2 have out-degree 1. However, for all $d \leq \min(d_{\mathsf{out}}, |V_1|)$, it can be extended to the case where the out-degree of every vertex is at least d by letting all the vertices in V_2 have out-edges to the same subset $Z \subseteq V_1$ of size d of vertices in V_1 instead of having edges to exactly one vertex $z \in V_1$.

With this change, a random walk would not always return to the same vertex in V_1 and instead returns to a uniformly random vertex in Z. Thus, for any given vertex in Z, a random walk will only return to it (roughly) $L/(2d)$ many times. In turn, we have that the algorithm only needs to store around $L/(2d)$ many edges coming out of any vertex. As this number is now smaller, this would lead to a reduced space lower bound of $\Omega\big(|V_1| \cdot \min\big(d_{\mathsf{out}}, \frac{L}{d}\big)\big)$. This is exactly where the expression in Theorem 1.1 comes from, but its not the end of the proof, as the input graph is not guaranteed to be bipartite or promised to have the same number of edges from all vertices in V_1.

Extending to general graphs G. We now turn to show how the above approach can be extended to work for all graphs and give a lower bound of $\tilde{\Omega}\big(\sum_v \min\big(d_{\mathsf{out}}(v), \frac{L}{d}\big)\big)$ by embedding a hard instance of the form above in any input graph G. For this, we "clean" any input graph G in a sequence of steps, ultimately leading to a hard instance like those above.

Our first step is to identify a "dominant" degree in the graph. For this, we partition the vertices into $\log n$ levels based on which two consecutive powers of 2 its out-degree lies in. This also partitions the sum we want in our lower bound and we focus on the part that is the largest. This gives a subset of vertices $V' \subseteq V$ of vertices that have similar degrees and suffices for the lower bound up to logarithmic factors.

Having decided on a set V' to focus on, we now identify a cut in the graph G such that a constant fraction of the vertices in V' lie on the left side of the cut, and a constant fraction

8

of the neighbors of all these vertices lie on the right side of the cut. To show that such a cut exists, note that a random cut in the graph will have the property such that in expectation, a constant fraction of the neighbors of any vertex lie on a different side of the cut as that vertex. Using probabilistic arguments, we show that a cut exists for which this is true for a constant fraction of the vertices in V'.

We are almost there, except that the vertices in V' don't have *exactly* the same degree which was assumed in our proof approach. This can be easily fixed by removing some of the edges from all vertices in V' till they have the same degree. As they had degrees within a factor of 2 to being with, this loses only a constant fraction of edges and only costs us another constant factor in our lower bound. Another difference is that we started with a graph where every vertex is promised to have out-degree at least d. However, as we removed the edges not in the cut, we may have shrunk the out-degrees of vertices. As this may make some out-degrees smaller than d, it may take the resulting graph outside the promise, effectively making it useless. To fix this, we identify another set of $\mathcal{O}(d)$ randomly chosen vertices in the graph and connect them with each other in a clique and also connect all the other vertices to them. This increases the degrees of all vertices back to at least d, satisfying the promise. Moreover, as[9] $d = o(n)$, this only loses us another constant in the lower bound.

A hard distribution. Finally, note that the above construction only gave us a single graph while our lower bound approach required a distribution of graphs. This is a major difference, as there are algorithms that will solve the search problem on this particular graph (one can simply "hardwire" the answer for this graph in the algorithm) making a lower bound impossible. We get around this by converting our hard graph into a distribution over graphs as follows: For every vertex in V', we only include a randomly chosen subset of half its edges in the graph and the other edges are discarded. This again only costs us a constant factor but creates for us a distribution just like we need for the lower bound approach.

We finish this section by arguing why a lower bound obtained from such a hard distribution after making all these changes would also apply to the original graph G, as required for universal optimality. For this, we observe that our lower bound approach shows a lower bound on the memory needed at a *specific* point in the stream, when it has processed all the edges coming out from V_1 but before processing any edges that come out from V_2. In our constructed distribution, the set V_1 corresponds to the set V' and thus, the inputs streams we generate will start with the edges coming out of the vertices in V' and the memory lower bound will hold for the memory state of the algorithm immediately after processing the edges. Crucially, (unlike the edges going into V' or the edges added to raise the degree) all these edges were also part of the original graph G. Thus, for all the algorithm knows, the graph might as well have been G with these edges presented first in the stream, and the lower bound we show for our constructed distribution will also apply to the original graph

---

[9]When d and n are within a constant factor of each other, our bound becomes $\tilde{\mathcal{O}}(L)$ which is tight as it is the size of the output.

G, finishing the proof.

3   Preliminaries

Notation.   For integer $n > 0$, we write $[n]$ as a shorthand for $\{1, \ldots, n\}$. $\mathbb{N}$ denotes the set of positive integers. We often omit ceiling symbols, and by assigning a real number x to an integer y, we mean that y gets $\lceil x \rceil$.

Graphs.   For a directed graph $G = (V, E)$ and any vertex $v \in V$, we define $N_{\mathsf{out}}^G(v)$ to be the set of all out-edges of v and define $d_{\mathsf{out}}^G(v) = \left| N_{\mathsf{out}}^G(v) \right|$ to be the out-degree of v. We omit the superscript G when it is clear from context. We use $n = |V|$ throughout.

Random walks.   An L-step *random walk* in G with starting vertex $v_0 \in V$ is a (directed) path $(v_0, \ldots, v_L)$ of length L in G such that v_i is an independent and uniformly random out-neighbor of v_{i-1} for all $i \in [L]$. We say an algorithm samples an L-step random walk starting from v_0 with *distortion* $\delta \geq 0$, if the sampled distribution of $(v_0, \ldots, v_L)$ is δ-close in statistical distance to the true distribution of an L-step random walk starting from v_0.

Graph streaming.   An (insertion-only) graph streaming algorithm takes as input a sequence of edges of an underlying graph, and is required to process the edges one at a time using limited space. At the end of the stream, the algorithm needs to return an output for certain graph problems. In this paper, we consider the task of sampling random walks.

Universal optimality.   A graph streaming algorithm τ is *universally optimal* for a task T over a set of graphs $\mathcal{G}$ if:

1. τ solves T.

2. For *any* streaming algorithm τ' that also solves T, the following holds: Let G be *any* graph in $\mathcal{G}$ and let S be the maximum memory size used by τ' over all streaming orders of G. Then, the maximum memory size used by τ over any streaming order of G is at most $S \cdot \operatorname{poly} \log n$.

Chernoff bound.   We use the following version of the Chernoff bound.

Lemma 3.1 (Multiplicative Chernoff bound). *Suppose* $\mathsf{X}_1, \cdots, \mathsf{X}_n$ *are independent random variables taking values in* $[0, 1]$. *Let* X *denote their sum. Then,*

$$\Pr(\mathsf{X} \geq (1 + \delta) \cdot \mathbb{E}[\mathsf{X}]) \leq \mathrm{e}^{-\frac{\delta^2 \cdot \mathbb{E}[\mathsf{X}]}{2 + \delta}}, \qquad \forall 0 \leq \delta,$$

$$\Pr(\mathsf{X} \leq (1 - \delta) \cdot \mathbb{E}[\mathsf{X}]) \leq \mathrm{e}^{-\frac{\delta^2 \cdot \mathbb{E}[\mathsf{X}]}{2}}, \qquad \forall 0 \leq \delta \leq 1.$$

Information theory. We use sans-serif letters to denote random variables. We reserve $\mathcal{E}$ to denote an arbitrary event. All random variables will be assumed to be discrete and we shall adopt the convention $0 \log \frac{1}{0} = 0$. When it is clear from context, we may abbreviate an event $\mathsf{X} = x$ as just x. All logarithms are taken with base 2.

Definition 3.2 (Entropy). *The (binary) entropy of* X *is defined as:*

$$\mathbb{H}(\mathsf{X}) = \sum_{x \in \mathsf{supp}(\mathsf{X})} \Pr(x) \cdot \log \frac{1}{\Pr(x)}.$$

The entropy of X *conditioned on* $\mathcal{E}$ *is defined as:*

$$\mathbb{H}(\mathsf{X} \mid \mathcal{E}) = \sum_{x \in \mathsf{supp}(\mathsf{X})} \Pr(x \mid \mathcal{E}) \cdot \log \frac{1}{\Pr(x \mid \mathcal{E})}.$$

Definition 3.3 (Conditional entropy). *We define the conditional entropy of* X *given* Y *and* $\mathcal{E}$ *as:*

$$\mathbb{H}(\mathsf{X} \mid \mathsf{Y}, \mathcal{E}) = \sum_{y \in \mathsf{supp}(\mathsf{Y})} \Pr(y \mid \mathcal{E}) \cdot \mathbb{H}(\mathsf{X} \mid y, \mathcal{E}).$$

Henceforth, we shall omit writing the $\mathsf{supp}(\cdot)$ when it is clear from context.

Lemma 3.4 (Chain rule for entropy). *It holds for all* X, Y, Z *and* $\mathcal{E}$ *that:*

$$\mathbb{H}(\mathsf{X}, \mathsf{Y} \mid \mathsf{Z}, \mathcal{E}) = \mathbb{H}(\mathsf{X} \mid \mathsf{Z}, \mathcal{E}) + \mathbb{H}(\mathsf{Y} \mid \mathsf{X}, \mathsf{Z}, \mathcal{E}).$$

Lemma 3.5 (Conditioning reduces entropy). *It holds for all* X, Y, Z *and* $\mathcal{E}$ *that:*

$$\mathbb{H}(\mathsf{X} \mid \mathsf{Y}, \mathsf{Z}, \mathcal{E}) \leq \mathbb{H}(\mathsf{X} \mid \mathsf{Z}, \mathcal{E}).$$

Equality holds if and only if X *and* Y *are independent conditioned on* Z, $\mathcal{E}$.

As a corollary of Lemmas 3.4 and 3.5, we get that entropy is subadditive.

Corollary 3.6 (Subadditivity of entropy). *It holds for all* X, Y, Z *and* $\mathcal{E}$ *that:*

$$\mathbb{H}(\mathsf{X}, \mathsf{Y} \mid \mathsf{Z}, \mathcal{E}) \leq \mathbb{H}(\mathsf{X} \mid \mathsf{Z}, \mathcal{E}) + \mathbb{H}(\mathsf{Y} \mid \mathsf{Z}, \mathcal{E}).$$

Lemma 3.7. *It holds for all* X *and* $\mathcal{E}$ *that:*

$$0 \leq \mathbb{H}(\mathsf{X} \mid \mathcal{E}) \leq \log(|\mathsf{supp}(\mathsf{X})|).$$

The second inequality is tight if and only if X *conditioned on* $\mathcal{E}$ *is the uniform distribution over* $\mathsf{supp}(\mathsf{X})$.

As a corollary of Lemmas 3.4 and 3.7, we also have the following bound.

Corollary 3.8. *It holds for all* X, Y, Z *and* $\mathcal{E}$ *that:*

$$\mathbb{H}(\mathsf{X} \mid \mathsf{Y}, \mathsf{Z}, \mathcal{E}) \geq \mathbb{H}(\mathsf{X} \mid \mathsf{Z}, \mathcal{E}) - \mathbb{H}(\mathsf{Y} \mid \mathsf{Z}, \mathcal{E}).$$

Lemma 3.9 (Shearer's inequality). *Let* $n > 0$ *and* $\mathsf{X}_1, \ldots, \mathsf{X}_n$ *be random variables. Let* $\mathsf{X} = (\mathsf{X}_1, \ldots, \mathsf{X}_n)$ *and* $\mathsf{Z} \subseteq [n]$ *be a set-valued random variable that is independent of* X. *For any event* $\mathcal{E}$ *determined by* X, *it holds that:*

$$\mathbb{H}(\mathsf{X} \mid \mathcal{E}) \cdot \min_{i \in [n]} \Pr(i \in \mathsf{Z}) \leq \mathbb{H}(\mathsf{X}_{\mathsf{Z}} \mid \mathsf{Z}, \mathcal{E}).$$

Proof. We have:

$$
\begin{aligned}
\mathbb{H}(\mathsf{X}_{\mathsf{Z}} \mid \mathsf{Z}, \mathcal{E}) &= \sum_Z \Pr(Z \mid \mathcal{E}) \cdot \mathbb{H}(\mathsf{X}_{\mathsf{Z}} \mid Z, \mathcal{E}) && \text{(Definition 3.3)} \\
&= \sum_Z \Pr(Z \mid \mathcal{E}) \cdot \mathbb{H}(\mathsf{X}_Z \mid Z, \mathcal{E}) && \\
&= \sum_Z \Pr(Z) \cdot \mathbb{H}(\mathsf{X}_Z \mid \mathcal{E}) && \text{(as } \mathsf{Z} \text{ is independent of } \mathsf{X}, \mathcal{E}) \\
&= \sum_Z \Pr(Z) \cdot \sum_{i \in Z} \mathbb{H}(\mathsf{X}_i \mid \mathsf{X}_{Z \cap [i-1]}, \mathcal{E}) && \text{(Lemma 3.4)} \\
&\geq \sum_Z \Pr(Z) \cdot \sum_{i \in Z} \mathbb{H}(\mathsf{X}_i \mid \mathsf{X}_{[i-1]}, \mathcal{E}) && \text{(Lemma 3.5)} \\
&= \sum_{i=1}^n \sum_Z \Pr(Z) \cdot \mathbb{1}(i \in Z) \cdot \mathbb{H}(\mathsf{X}_i \mid \mathsf{X}_{[i-1]}, \mathcal{E}) && \\
&\geq \min_{i \in [n]} \Pr(i \in \mathsf{Z}) \cdot \sum_{i=1}^n \mathbb{H}(\mathsf{X}_i \mid \mathsf{X}_{[i-1]}, \mathcal{E}) && \\
&= \mathbb{H}(\mathsf{X} \mid \mathcal{E}) \cdot \min_{i \in [n]} \Pr(i \in \mathsf{Z}). && \text{(Lemma 3.4)}
\end{aligned}
$$

$\square$

Corollary 3.10. *Let* $n > 0$ *and* $\mathsf{X}_1, \ldots, \mathsf{X}_n$ *be random variables. Let* $\mathsf{X} = (\mathsf{X}_1, \ldots, \mathsf{X}_n)$ *and* $\mathsf{Z} \subseteq [n]$ *be a set-valued random variable that is independent of* X. *For any random variable* Y *determined by* X, *it holds that:*

$$\mathbb{H}(\mathsf{X} \mid \mathsf{Y}) \cdot \min_{i \in [n]} \Pr(i \in \mathsf{Z}) \leq \mathbb{H}(\mathsf{X}_{\mathsf{Z}} \mid \mathsf{Z}, \mathsf{Y}).$$

4 Streaming Algorithm

The goal of this section is to prove the following Theorem 4.1. Algorithms $\pi_{d,L}$, $\pi_{d,L}^{\mathsf{sparse}}$, and $\pi_{d,L}^{\mathsf{dense}}$, are given in Algorithms 1 to 3, respectively. By "output FAIL," we mean that the

---

Algorithm 1 The protocol $\pi_{d,L}(G, v_0)$.

1: If $\frac{L}{d} > 10^3 \log n$, run $\pi_{d,L}^{\mathsf{sparse}}(G, v_0)$.
2: Otherwise, run $\pi_{d,L}^{\mathsf{dense}}(G, v_0)$.

Algorithm 2 The protocol $\pi_{d,L}^{\mathsf{sparse}}(G, v_0)$.

1: For each vertex $v \in V$, if $d_{\mathsf{out}}(v) \leq \frac{2L}{d}$, store all its outgoing edges, and otherwise, sample $\frac{2L}{d}$ outgoing edges with replacement, uniformly at random, denoted $u_{v,1}, \ldots, u_{v,\frac{2L}{d}}$.
2: Output a path $(v_0, v_1, \ldots, v_L)$ such that for all $i \in [L]$, if $d_{\mathsf{out}}(v_{i-1}) \leq \frac{2L}{d}$, v_i is an independent and uniformly random out-neighbor of v_{i-1}, and otherwise, $v_i = u_{v_{i-1},t}$ where $t = |\{i' \in [i] \mid v_{i'-1} = v_{i-1}\}|$. If $t > \frac{2L}{d}$ for some $i \in [L]$, output FAIL.

---

algorithm outputs $(v_0, \ldots, v_0)$. We will elaborate on the details of their streaming implementation later on.

Theorem 4.1. *Let $d, L \in \mathbb{N}$. $\pi_{d,L}$ solves the task of sampling an L-step random walk with distortion 0.1. It uses $\tilde{\mathcal{O}}(\sum_v \min(d_{\mathsf{out}}(v), L/d))$ space.*

As Algorithm 1 is simply a combination of Algorithms 2 and 3, Theorem 4.1 is a direct corollary of the following Lemmas 4.2 and 4.3. Indeed, when $L/d \leq 10^3 \log n$, we know that L/d is larger than $\min(d_{\mathsf{out}}(v), L/d)$ by a multiplicative factor of at most $10^3 \log n$ as $d_{\mathsf{out}}(v) \geq d \geq 1$. Note that $\sum_v \min(d_{\mathsf{out}}(v), L/d)$ can be sublinear in n when $L/d \ll 1$.

We also emphasize that the proof of Lemma 4.3 does not require $L/d \leq 10^3 \log n$. In other words, Algorithm 3 alone is sufficient to achieve $\tilde{\mathcal{O}}(Ln/d)$ space in the worst case. However, as noted in Section 1, Algorithm 2 and Lemma 4.2 are necessary for achieving universal optimality.

Lemma 4.2. *Let $d, L \in \mathbb{N}$ such that $L/d > 10^3 \log n$. $\pi_{d,L}^{\mathsf{sparse}}$ solves the task of sampling an L-step random walk with distortion 0.1. It uses $\tilde{\mathcal{O}}(\sum_v \min(d_{\mathsf{out}}(v), L/d))$ space.*

Lemma 4.3. *Let $d, L \in \mathbb{N}$. $\pi_{d,L}^{\mathsf{dense}}$ outputs FAIL with probability at most 0.1 and otherwise samples an (unbiased) L-step random walk. It uses $\tilde{\mathcal{O}}(Ln/d)$ space.*

The following two sections constitute proofs of Lemmas 4.2 and 4.3, respectively.

4.1 Proof of Lemma 4.2

The correctness of Algorithm 2 follows from the following claim.

Claim 4.4. *Let $d, L \in \mathbb{N}$ such that $L/d > 10^3 \log n$. For any directed graph $G = (V, E)$ with minimum out-degree at least d and vertex $v_0 \in V$, with probability at least 0.9, an L-step random walk from v_0 in G visits any vertex at most $2L/d$ times.*

13

Algorithm 3 The protocol $\pi_{d,L}^{\mathsf{dense}}(G, v_0)$.
_____
1: Set $V_0 = \{v_0\}$.
2: Independently sample subsets $V_1, \ldots, V_L \subseteq V$, each of size $s = \min(\frac{n}{d} \cdot \ln(10Ln), n)$, uniformly at random.
3: For each $i \in [L]$ and vertex $v \in V_{i-1}$, independently sample an outgoing edge (v, u), uniformly at random, conditioned on $u \in V_i$.
4: Output the (unique) path $(v_0, v_1, \ldots, v_L)$ such that for all $i \in [L]$, (v_{i-1}, v_i) is sampled. If there is no edge from v_{i-1} to V_i for some $i \in [L]$, output FAIL.
_____

Proof. Fix $v \in V$. For $i \in [L]$, let V_i be the random variable for the i-th vertex on an L-step random walk from v_0 in G. For any sub-path $(v_0, \ldots, v_{i-1})$, $\Pr(V_i = v \mid V_0 = v_0, \ldots, V_{i-1} = v_{i-1}) \leq \frac{1}{d}$ because v_{i-1} has at least d outgoing edges, with v_i possibly being one of them.

Note that the events $V_i = v$ may not be independent. Nevertheless, as each step of a random walk is sampled independently, the probability of v being visited more than $2L/d$ times is always upper bounded by the probability that L independent biased coins, each with $1/d$ probability of getting a head, generate more than $2L/d$ heads in total. By Lemma 3.1 and the assumption $L/d > 10^3 \log n$, this happens with probability at most $1/n^2$. The claim then follows by union bounding over all vertices. $\qquad\square$

As to the space usage, Algorithm 2 can be implemented using reservoir sampling [Vit85]. Concretely, for each vertex $v \in V$, it takes $\mathcal{O}(\log n)$ bits to keep track of $d_{\mathsf{out}}(v)$. Meanwhile, it stores all its outgoing edges while $d_{\mathsf{out}}(v) \leq 2L/d$. Once $d_{\mathsf{out}}(v) > 2L/d$ during the stream, it can use $2L/d$ independent reservoir samplers, each taking $\mathcal{O}(\log n)$ bits. So each vertex v takes $\tilde{\mathcal{O}}(\min(d_{\mathsf{out}}(v), L/d))$ space. Since the reservoir samplers are unbiased, this concludes the proof of Lemma 4.2 by applying Claim 4.4.

4.2 Proof of Lemma 4.3

To implement Algorithm 3, for each $i \in L$ and vertex $v \in V_{i-1}$, one reservoir sampler is enough since all V_i's can be sampled in advance. The space bound follows as L is at most poly(n).

We now show that conditioned on no failure, Algorithm 3 samples an (unbiased) L-step random walk. In other words, for every $i \in [L]$, v_i has to be a uniformly random out-neighbor of v_{i-1} in G. To this end, consider any two out-neighbors u, u' of v_{i-1} in G, and subset $U \subseteq V$ of size s. Suppose $V_i = U$. If $u, u' \in U$, each of them has the same probability of being sampled as v_i. If $u, u' \notin U$, none of them would be sampled as v_i. If exactly one of u, u' is in U, assume $u \in U$ without loss of generality. Since V_i is a uniformly random subset of size s, $U' = (U \cup \{u'\}) \setminus \{u\}$ has the same probability as U of being sampled as V_i. Furthermore, the number of out-neighbors of v_{i-1} in V_i is the same in the two cases. As a result, the probability of u' being sampled as v_i in the case $V_i = U'$ is the same as the probability of u being sampled as v_i in the case $V_i = U$. Altogether, this means that the overall probability

of being sampled as v_i is the same for u, u'.

It then remains to show that Algorithm 3 outputs FAIL with probability at most 0.1. Note that the only possibility of failure is when there exists $i \in [L]$ and vertex $v \in V_{i-1}$ such that v has no out-neighbor in V_i. This happens only if $s = n/d \cdot \ln(10Ln)$. For any $i \in [L]$ and $v \in V_{i-1}$, the probability of v not having out-neighbors in V_i is at most

$$\frac{\binom{n-d}{s}}{\binom{n}{s}} = \prod_{j=1}^{s} \frac{n-d-j+1}{n-j+1} \leq \left(\frac{n-d}{n}\right)^s \leq \exp\left(-\frac{ds}{n}\right) = \frac{1}{10Ln}.$$

This concludes the proof of Lemma 4.3 by union bounding over all $1 + s(L-1) \leq Ls \leq Ln$ options for i, v.

5 Universal Optimality

The goal of this section is to show that the algorithm described in Algorithm 1 is universally optimal upto logarithmic factors.

We define a triple of positive integers $L, d, n \in \mathbb{N}$ to be *significant* if $n > 0$ is large enough for asymptotic inequalities to holds and $L \geq (\log n)^{10}$ and $(\log n)^{10} \leq d \leq \frac{n}{(\log n)^{10}}$.[10] For $d, n > 0$, we define the set $\mathcal{G}_{n,d}$ to be the set of all directed simple graphs where there are n vertices and every vertex has out-degree at least d. We say that a streaming algorithm Alg is a *random-walk algorithm* if given significant integers L, d, n, a graph $G = (V, E) \in \mathcal{G}_{n,d}$ and a start vertex $\in V$, the algorithm Alg samples an L-step random walk from the start vertex with distortion at most 0.1. The theorem we show in this section is the following Theorem 5.1. Theorem 1.1 then follows directly from Theorem 5.1 and Theorem 4.1.

Theorem 5.1. *Let $L, d, n \in \mathbb{N}$ be significant, Alg be a random-walk algorithm, and $G = (V, E) \in \mathcal{G}_{n,d}$ with a designated start vertex. There exists a streaming order for the edges of G on which Alg needs memory at least $\frac{1}{(\log n)^4} \cdot \sum_{v \in V} \min\left(d_{\mathsf{out}}(v), \frac{L}{d}\right)$.*

The proof of Theorem 5.1 spans this entire section. Fix L, D, n, Alg, and G be as in the theorem statement. As $G \in \mathcal{G}_{n,d}$, we have that for all $v \in V$, it holds that $d \leq d_{\mathsf{out}}(v) < n$. This means that there exists $\log d < k \leq \log n$ such that:

$$\frac{1}{\log n} \cdot \sum_{v \in V} \min\left(d_{\mathsf{out}}(v), \frac{L}{d}\right) \leq \sum_{\substack{v \in V \\ 2^{k-1} \leq d_{\mathsf{out}}(v) < 2^k}} \min\left(d_{\mathsf{out}}(v), \frac{L}{d}\right). \tag{1}$$

We fix this k for the rest of this proof and define V' to be the set of vertices in the sum on the right. Namely,

$$V' = \left\{v \in V \mid 2^{k-1} \leq d_{\mathsf{out}}(v) < 2^k\right\}. \tag{2}$$

---

[10] If $d \geq n/(\log n)^{10}$, $\pi_{d,L}$ (or simply $\pi_{d,L}^{\mathsf{dense}}$) uses at most $\tilde{\mathcal{O}}(L)$ space. This must be universally optimal as outputting a length-L path requires $\Omega(L)$ space.

Our first lemma shows that the set V' is large.

Lemma 5.2. *It holds that $|V'| \geq \frac{d}{\log n}$.*

Proof. We first prove this under the assumption that $L \leq d^2$. In this case, we have $\frac{L}{d} \leq d \leq d_{\text{out}}(v)$ for all $v \in V$. Plugging this in Equation (1), we get $\frac{1}{\log n} \cdot \frac{nL}{d} \leq |V'| \cdot \frac{L}{d}$ which gives the lemma as $d \leq n$. Now, assume that $L > d^2$. This means that $\frac{L}{d} > d$ implying that the minimum in the left hand side in Equation (1) can be lower bounded by d while the minimum on the right hand side can be upper bounded by n. Plugging in, we get $\frac{1}{\log n} \cdot nd \leq n \cdot |V'|$. Rearranging gives the result. $\qquad \square$

We use $\text{Pr}_{U \subseteq V}(\cdot)$ and $\mathbb{E}_{U \subseteq V}[\cdot]$ to denote the probability and expectation when U is a subset of vertices chosen uniformly at random. For any subset $U \subseteq V$, we define $\text{cut}(U) = E \cap \big((U \times \overline{U}) \cup (\overline{U} \times U)\big)$ to be the set of edges in the cut formed by U. Also, for all $U \subseteq V$, define:

$$\text{Big}(U) = \{v \in V \mid |N_{\text{out}}(v) \cap \text{cut}(U)| \geq 0.1 \cdot d_{\text{out}}(v)\}. \tag{3}$$

We show that there exists $U \subseteq V$ such that many of the vertices in V' lie in $\text{Big}(U) \cap U$.

Lemma 5.3. *There exists $U \subseteq V$ satisfying:*

$$|V' \cap \text{Big}(U) \cap U| \geq \frac{1}{5} \cdot |V'|.$$

Proof. For any $v \in V'$ such that $d_{\text{out}}(v) \leq 10$, we have that $v \notin \text{Big}(U)$ (for some $U \subseteq V$) if and only if all its out-neighbors are on the same side of the cut as v. As every vertex has at least one out-neighbor, we get that this event happens with probability at most $\frac{1}{2}$ for such v for a uniformly random $U \subseteq V$. For $v \in V'$ such that $d_{\text{out}}(v) \geq 10$, we have by Lemma 3.1 that:

$$\Pr_{U \subseteq V} (v \notin \text{Big}(U)) \leq \Pr_{U \subseteq V} (|N_{\text{out}}(v) \cap \text{cut}(U)| \leq 0.1 \cdot d_{\text{out}}(v)) \leq e^{-0.1 \cdot d_{\text{out}}(v)} < \frac{1}{2}.$$

Combining, we get that $\text{Pr}_{U \subseteq V}(v \notin \text{Big}(U)) \leq \frac{1}{2}$ for all $v \in V'$. It follows that:

$$\mathbb{E}_{U \subseteq V}[V' \setminus \text{Big}(U)] \leq \frac{1}{2} \cdot |V'|.$$

In particular, there exists a cut U for which $V' \setminus \text{Big}(U) \leq \frac{1}{2} \cdot |V'|$. From this, we get that at least one of the quantities $V' \cap \text{Big}(U) \cap U$ and $V' \cap \text{Big}(U) \cap \overline{U}$ is at least $\frac{1}{5} \cdot |V'|$. To finish the proof, observe from Equation (3) that $\text{Big}(U) = \text{Big}(\overline{U})$ and thus, at least one of U and $\overline{U}$ satisfies the lemma. $\qquad \square$

Next, we fix U to be the set promised by Lemma 5.3 and define $U' = V' \cap \text{Big}(U) \cap U$ and $d' = 0.1 \cdot \max(d, 2^{k-1})$. It follows that $|U'| \geq \frac{1}{5} \cdot |V'| \geq \frac{d}{5 \log n}$ by Lemma 5.2. Moreover,

for all $v \in U'$, we have that

$$
\begin{aligned}
\left| N_{\text{out}}(v) \cap \left(\{v\} \times \overline{U'} \right) \right| &\geq \left| N_{\text{out}}(v) \cap \left(\{v\} \times \overline{U} \right) \right| \\
&= \left| N_{\text{out}}(v) \cap \text{cut}(U) \right| \\
&\geq 0.1 \cdot d_{\text{out}}(v) \\
&\geq 0.1 \cdot \max\left(d, 2^{k-1} \right) \\
&= d'.
\end{aligned} \tag{4}
$$

Next, we isolate a suitably chosen set of vertices $W_1 \subseteq V$.

Lemma 5.4. *There exists a subset $W_1 \subseteq V$ such that all the following hold:*

$$
2d \leq |W_1| \leq 4d,
$$

$$
|U' \setminus W_1| \geq \frac{1}{2} \cdot |U'|,
$$

$$
\forall v \in U' : \qquad \left| N_{\text{out}}(v) \cap \left(\{v\} \times \left(\overline{U'} \cap \overline{W_1} \right) \right) \right| \geq \frac{1}{2} \cdot \left| N_{\text{out}}(v) \cap \left(\{v\} \times \overline{U'} \right) \right|.
$$

Proof. Proof by the probabilistic method. Consider the set-valued random variable W_1 obtained by including all $v \in V$ with probability $3d/n$. By Lemma 3.1 and using the fact that $d > (\log n)^{10}$, we get that the first inequality holds except with probability $\frac{1}{n^2}$. For the other inequalities, consider any set $S \subseteq V$ such that $|S| \geq (\log n)^5$. Note that because of our bound $|U'| \geq \frac{d}{5 \log n}$ and the bound in Equation (4), we have that U' and[11] $N_{\text{out}}(v) \cap \left(\{v\} \times \overline{U'} \right)$ for all $v \in U'$ are such sets. As $d \leq \frac{n}{(\log n)^{10}}$ implies that $3d/n \leq 0.01$, we have from Lemma 3.1 that $\Pr\left(|S \cap \mathsf{W}_1| \geq \frac{1}{2} \cdot |S| \right) \leq \frac{1}{n^2}$ for any such set S. A union bound over all sets S considered in the lemma statement finishes the proof. $\qquad \square$

Henceforth, we let W_1 be as promised by Lemma 5.4. We also define $W_2 = U' \setminus W_1$ and conclude from Lemma 5.4 that $|W_2| \geq \frac{1}{2} \cdot |U'| \geq \frac{d}{10 \log n}$. Finally, we define $W_3 = V \setminus (W_1 \cup W_2)$. For all $v \in W_2$, we have:

$$
\begin{aligned}
\left| N_{\text{out}}(v) \cap \left(\{v\} \times W_3 \right) \right| &= \left| N_{\text{out}}(v) \cap \left(\{v\} \times \overline{W_1 \cup W_2} \right) \right| \\
&= \left| N_{\text{out}}(v) \cap \left(\{v\} \times \overline{U' \cup W_1} \right) \right| \\
&\geq \frac{1}{2} \cdot \left| N_{\text{out}}(v) \cap \left(\{v\} \times \overline{U'} \right) \right| \\
&\geq d'/2.
\end{aligned} \tag{5}
$$

For all $v \in W_2$, let E'_v be an (arbitrary) subset of its out-edges going to vertices in W_3 of size $d'/2$. Observe from Equation (5) that at least one such subset exists and therefore, E'_v is well defined.

---

[11] Technically, as it needs to be a subset of V, the projection of $N_{\text{out}}(v) \cap \left(\{v\} \times \overline{U'} \right)$ on its second coordinate is such a set, but the same argument applies.

Next, for all $v \in W_2$, define the set-valued random variable $\mathsf{X}_v \subseteq E'_v$ to be an independent and uniformly random subseteq of E'_v of size $d'/4$. For notational convenience, for any subset $Z \subseteq W_2$, we will abbreviate $(\mathsf{X}_v)_{v \in Z}$ to X_Z and omit the subscript when $Z = W_2$. Also, define the set-valued random variable $\mathsf{Z} \subseteq W_2$ to be a uniformly random subset of W_2 of size $\frac{d}{20 \log n}$ chosen independently of X. As $|W_2| \geq \frac{d}{10 \log n}$, we have that the random variable Z is well defined. The following lemma is straightforward.

Lemma 5.5. *Let $v \in W_2$ and $S_v \subseteq E'_v$. It holds that:*

$$|\{X_v \in \mathsf{supp}(\mathsf{X}_v) \mid S_v \subseteq X_v\}| \leq \binom{d'/2}{d'/4} \cdot 2^{-|S_v|}.$$

Proof. The size of the set we have to bound equals $\binom{d'/2 - |S_v|}{d'/4 - |S_v|} \leq \binom{d'/2}{d'/4} \cdot 2^{-|S_v|}$. $\qquad\square$

5.1 A "Search" Lower Bound

Before showing the desired lower bound on the memory used by the algorithm Alg, we first define a related distributional search problem and show a lower bound for algorithms solving that. For this, let $\mathsf{X} = (\mathsf{X}_v)_{v \in W_2}$ and Z be as defined above. We use these random variables to define a graph G' with the vertex set $V = W_1 \cup W_2 \cup W_3$ and the following edges: (1) For all $v \in V$ and all $v' \neq v \in W_1$, there is an edge from v to v'. These edges are called the fixed edges. (2) For all $v \in W_2$, all the edges in X_v are present in G'. These edges are called the X-edges. (3) For all $v \in W_1$ and all $v' \in Z \subseteq W_2$, there is an edge from v to v'. These edges are called the Z-edges. We let G' be the random variable corresponding to G'. Note that the random variables G' and (X, Z) determine each other, and, more specifically, the X-edges and X determine each other and the Z-edges and Z determine each other. Also, note from the fact that $|W_1| \geq 2d$ (see Lemma 5.4) that the out-degree of any vertex in any graph G' in the support of G' is at least d. It follows that the support of G' is a subset of $\mathcal{G}_{n,d}$.

Having defined the random variable G', the search problem we consider receives the edges of a graph $G \sim \mathsf{G}'$, which we can equivalently view as a pair (X, Z), in the stream, with the X-edges arriving first (in lexicographic order), then the other edges (in lexicographic order). The goal of the algorithm is to make one-pass over the stream, and with probability at least 0.8, output at least $\frac{d}{(\log n)^3} \cdot \min(d', \frac{L}{d})$ distinct edges in $\bigcup_{v \in Z} X_v$.

Consider any streaming algorithm $\mathcal{A}$ that solves the above problem. As we study $\mathcal{A}$ on a specific distribution of inputs (specifically, the distribution G'), we can assume without loss of generality that $\mathcal{A}$ is deterministic. Let M the random variable denoting the memory state of the algorithm $\mathcal{A}$ after it has processed the X-edges but before processing the other edges and let O be the random variable denoting the output of $\mathcal{A}$. As $\mathcal{A}$ is deterministic, observe that M is a function of X and that O is a function of (M, Z). We abbreviate events of the form $\mathsf{M} = M$ to just M, $\mathsf{Z} = Z$ to just Z, and so on.

Lemma 5.6. *It holds that:*

$$\mathbb{H}(X_Z \mid Z, M) \le \frac{d}{20 \log n} \cdot \log \binom{d'/2}{d'/4} - \frac{1}{2} \cdot \frac{d}{(\log n)^3} \cdot \min\left(d', \frac{L}{d}\right).$$

Proof. Let E be an indicator random variable that is 1 if and only if the algorithm $\mathcal{A}$ is correct on input G' (which determines and is determined by (X, Z)). We have:

$$
\begin{aligned}
\mathbb{H}(X_Z \mid Z, M) &\le \mathbb{H}(X_Z \mid Z, M, E) + \mathbb{H}(E \mid Z, M) && \text{(Corollary 3.8)} \\
&\le 1 + \mathbb{H}(X_Z \mid Z, M, E) && \text{(Lemma 3.7)} \\
&\le 1 + \sum_{Z,M,E} \Pr(Z, M, E) \cdot \mathbb{H}(X_Z \mid Z, M, E) && \text{(Definition 3.3)} \\
&\le 1 + \sum_{Z,M,E} \Pr(Z, M, E) \cdot \mathbb{H}(X_Z \mid Z, M, E). &&
\end{aligned}
$$

We now use the upper bound for the entropy in Lemma 3.7. For the terms corresponding to $E = 0$, we use Lemma 5.5 with $S_v = \emptyset$ to get $|\mathsf{supp}(X_v)| \le \binom{d'/2}{d'/4}$ for all $v \in Z$. For the terms corresponding to $E = 1$, we use the fact that Z and M together determine the output of the algorithm, which when $E = 1$, determines $\frac{d}{(\log n)^3} \cdot \min(d', \frac{L}{d})$ of the edges in $\bigcup_{v \in Z} X_v$. By Lemma 5.5, this means that conditioned on Z, M, $E = 1$, the random variable X_Z is supported on a set of size at most $\left(\binom{d'/2}{d'/4}\right)^{\frac{d}{20 \log n}} \cdot 2^{-\frac{d}{(\log n)^3} \cdot \min(d', \frac{L}{d})}$. We get:

$$
\begin{aligned}
\mathbb{H}(X_Z \mid Z, M) &\le 1 + \frac{d}{20 \log n} \cdot \log \binom{d'/2}{d'/4} - \Pr(E = 1) \cdot \frac{d}{(\log n)^3} \cdot \min\left(d', \frac{L}{d}\right) \\
&\le 1 + \frac{d}{20 \log n} \cdot \log \binom{d'/2}{d'/4} - 0.8 \cdot \frac{d}{(\log n)^3} \cdot \min\left(d', \frac{L}{d}\right) \\
&&& \text{(As } \Pr(E = 1) \ge 0.8) \\
&\le \frac{d}{20 \log n} \cdot \log \binom{d'/2}{d'/4} - \frac{1}{2} \cdot \frac{d}{(\log n)^3} \cdot \min\left(d', \frac{L}{d}\right). \\
&&& \text{(As } L, d \ge (\log n)^{10} \text{ and } d' \ge 0.1d)
\end{aligned}
$$

$\qquad\square$

Combining Lemma 5.6 with Corollary 3.10, we get:

$$\mathbb{H}(X \mid M) \cdot \frac{d}{20 \log n} \cdot \frac{1}{|W_2|} \le \frac{d}{20 \log n} \cdot \log \binom{d'/2}{d'/4} - \frac{1}{2} \cdot \frac{d}{(\log n)^3} \cdot \min\left(d', \frac{L}{d}\right).$$

Using the fact that $\mathbb{H}(X) = |W_2| \cdot \log \binom{d'/2}{d'/4}$ and simplifying, we get:

$$\mathbb{H}(X \mid M) \le \mathbb{H}(X) - 10 \cdot \frac{|W_2|}{(\log n)^2} \cdot \min\left(d', \frac{L}{d}\right).$$

From Corollary 3.8 and the fact that $d' \geq 2^k/20$, this means that:

$$\mathbb{H}(\mathsf{M}) \geq \frac{1}{2} \cdot \frac{|W_2|}{(\log n)^2} \cdot \min\left(2^k, \frac{L}{d}\right).$$

Now, from Equation (1) and as $|W_2| \geq \frac{1}{2} \cdot |U'| \geq \frac{1}{10} \cdot |V'|$, we get:

$$\mathbb{H}(\mathsf{M}) \geq \frac{1}{(\log n)^4} \cdot \sum_{v \in V} \min\left(d_{\mathsf{out}}(v), \frac{L}{d}\right).$$

As $\mathbb{H}(\mathsf{M})$ is upper bounded by the logarithm of the size of the support of M, we have that the algorithm $\mathcal{A}$ must use at least $\frac{1}{(\log n)^4} \cdot \sum_{v \in V} \min\left(d_{\mathsf{out}}(v), \frac{L}{d}\right)$ bits of memory.

5.2 Finishing the Proof

To finish the argument, we prove that the streaming Alg (that is a random-walk algorithm) implies a streaming algorithm that solves our search problem from Section 5.1 with probability at least 0.8. For this, recall that any graph G' in the support of G' lies in $\mathcal{G}_{n,d}$. This means that we can run the algorithm Alg (with the start vertex being an arbitrary vertex $v_0 \in W_1$ and the order of edges being the same as in the search problem) and it will sample an L-step random walk from v_0 with distortion at most 0.1. Let $\mathsf{P}_{G'} = (v_0, \mathsf{v}_1, \ldots, \mathsf{v}_L)$ be the random variable denoting the path, written as a sequence of vertices, taken by an L-step random walk from v_0.

Lemma 5.7. *For all graphs G', which we can equivalently view as a pair (X, Z), it holds that:*

$$\Pr\left(\mathsf{P}_{G'} \text{ has } < \frac{d}{(\log n)^3} \cdot \min\left(d', \frac{L}{d}\right) \text{ distinct edges from } \bigcup_{v \in Z} X_v\right) < 0.1$$

Proof. Note that when the walk is at any vertex $v \in W_1$, the edge from v will either go to another vertex in W_1 (when it is a fixed edge added in Item 1 in Section 5.1) or it will go to a vertex in $Z \subseteq W_2$ (when it is a Z-edge added in Item 3). In the latter case, either the second edge will go to a vertex in W_1 (a fixed edge) or it will go to a vertex in W_3 (when it is an X-edge added in Item 2). If it does go to a vertex in W_3, the third edge has to be fixed and take it to a vertex in W_1. Thus, at least one of any three consecutive vertices in $\mathsf{P}_{G'}$ must lie in W_1.

In particular, if we break the random walk into chunks of length 10, then each such chunk $l \in [L/10]$ will contain at least two vertices in W_1 within it. For all $l \in [L/10]$, we define $\mathsf{u}_l^{(1)}, \mathsf{u}_l^{(2)}$ to be random variables denoting the positions (these are values in the interval $[(l-1) \cdot 10 + 1, (l-1) \cdot 10 + 6]$) of the first two vertices in W_1 in chunk l. As $|W_1| \leq 4d$ by Lemma 5.4, $|Z| = \frac{d}{20 \log n}$ and $|X_v| = d'/4 \geq d/40$ for all $v \in W_2$, we have by the above

20

discussion that, for all $l \in [L/10]$:

$$\Pr\left(\mathsf{u}_l^{(2)} - \mathsf{u}_l^{(1)} = 3\right) \geq \frac{1}{1000\log n}.$$

Because of the fact that $\mathsf{P}_{G'}$ is a random walk, the same bound holds even when conditioned on $\mathsf{u}_1^{(1)}, \mathsf{u}_1^{(2)}, \ldots \mathsf{u}_{l-1}^{(1)}, \mathsf{u}_{l-1}^{(2)}$. This means that we can find mutually independent indicator random variables $\mathsf{Y}_1, \ldots, \mathsf{Y}_{L/10}$, jointly distributed with $\mathsf{P}_{G'}$ such that for all $l \in [L/10]$, we have $\Pr(\mathsf{Y}_l = 1) = \frac{1}{1000\log n}$ and $\mathsf{Y}_l = 1 \implies \mathsf{u}_l^{(2)} - \mathsf{u}_l^{(1)} = 3$ with probability 1. From Lemma 3.1, it follows that:

$$\Pr\left(\sum_{l=1}^{L/10} \mathbb{1}\left(\mathsf{u}_l^{(2)} - \mathsf{u}_l^{(1)} = 3\right) \leq \frac{L}{(\log n)^2}\right) \leq \Pr\left(\sum_{l=1}^{L/10} \mathsf{Y}_l \leq \frac{L}{(\log n)^2}\right) \leq \frac{1}{n^2},$$

as $L \geq (\log n)^{10}$ and n is large enough. For the rest of this proof, we condition on any values $\left(u_l^{(1)}, u_l^{(2)}\right)_{l \in [L/10]}$ of $\left(\mathsf{u}_l^{(1)}, \mathsf{u}_l^{(2)}\right)_{l \in [L/10]}$ such that the above event does not occur, and also condition on all the vertices visited in the walk except the vertices $\left(\mathsf{v}_{u_l^{(1)}+1}, \mathsf{v}_{u_l^{(1)}+2}\right)_{l \in [L/10]: u_l^{(2)} - u_l^{(1)} = 3}$. Observe that, under this conditioning, for all $l \in [L/10]$ such that $u_l^{(2)} - u_l^{(1)} = 3$, the edge $\left(\mathsf{v}_{u_l^{(1)}+1}, \mathsf{v}_{u_l^{(1)}+2}\right)$ is an independent and uniformly random edge in $\bigcup_{v \in Z} X_v$. As the total number of edges in $\bigcup_{v \in Z} X_v$ is $\frac{dd'}{80\log n}$ and these edges constitute at least $\frac{L}{(\log n)^2}$ independent uniformly random samples from it, the number of distinct samples will be at least $\frac{1}{100} \cdot \min\left(\frac{dd'}{80\log n}, \frac{L}{(\log n)^2}\right) \geq \frac{d}{(\log n)^3} \cdot \min\left(d', \frac{L}{d}\right)$ except with probability at most 0.05. A union bound finishes the proof. $\qquad\square$

As for any graph $G' \sim \mathsf{G}'$, the algorithm Alg samples from $\mathsf{P}_{G'}$ with distortion at most 0.1, we have from Lemma 5.7 that the algorithm Alg solves the search problem with at least probability 0.8. From the result proved in Section 5.1, we get that the memory needed by the Alg after processing the X-edges is at least $\frac{1}{(\log n)^4} \cdot \sum_{v \in V} \min\left(d_{\mathsf{out}}(v), \frac{L}{d}\right)$. As the X-edges are always a subset of the edges in the graph G, there is a streaming order of the edges of G such that these are the edges that the algorithm Alg sees first when it is run on G. It follows that Alg needs memory at least $\frac{1}{(\log n)^4} \cdot \sum_{v \in V} \min\left(d_{\mathsf{out}}(v), \frac{L}{d}\right)$ when run on G and the proof of Theorem 5.1 is complete.

References

[ABC17] Peyman Afshani, Jérémy Barbay, and Timothy M. Chan. Instance-optimal geometric algorithms. *J. ACM*, 64(1):3:1–3:38, 2017. 5

[ACL07] Reid Andersen, Fan Chung, and Kevin Lang. Using pagerank to locally partition a graph. *Internet Mathematics*, 4(1):35–64, 2007. 1

[AP09]   Reid Andersen and Yuval Peres. Finding sparse cuts locally using evolving sets. In *Symposium on Theory of computing (STOC)*, pages 235–244, 2009. 1

[BDV18]   Aditya Bhaskara, Samira Daruki, and Suresh Venkatasubramanian. Sublinear algorithms for MAXCUT and correlation clustering. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 107 of *LIPIcs*, pages 16:1–16:14, 2018. 6

[BP98]   Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998. 1

[CKKP22]   Ashish Chiplunkar, John Kallaugher, Michael Kapralov, and Eric Price. Factorial lower bounds for (almost) random order streams. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 486–497. IEEE, 2022. 6

[CKP+21]   Lijie Chen, Gillat Kol, Dmitry Paramonov, Raghuvansh R. Saxena, Zhao Song, and Huacheng Yu. Near-optimal two-pass streaming algorithm for sampling random walks over directed graphs. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 198, pages 52:1–52:19, 2021. 5

[COP03]   Moses Charikar, Liadan O'Callaghan, and Rina Panigrahy. Better streaming algorithms for clustering problems. In *Symposium on Theory of computing (STOC)*, pages 30–39, 2003. 1

[FLN01]   Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. In *Symposium on Principles of Database Systems (PODC)*, 2001. 5

[GKP98]   Juan A. Garay, Shay Kutten, and David Peleg. A sublinear time distributed algorithm for minimum-weight spanning trees. *SIAM journal on computing*, 27(1):302–316, 1998. 3, 4

[GT03]   Oded Goldreich and Luca Trevisan. Three theorems regarding testing graph properties. *Random Struct. Algorithms*, 23(1):23–57, 2003. 7

[Jin19]   Ce Jin. Simulating random walks on graphs in the streaming model. In Avrim Blum, editor, *Innovations in Theoretical Computer Science Conference (ITCS)*, volume 124 of *LIPIcs*, pages 46:1–46:15, 2019. 1, 5

[JS89]   Mark Jerrum and Alistair Sinclair. Approximating the permanent. *SIAM journal on computing*, 18(6):1149–1178, 1989. 1

[JST11]   Hossein Jowhari, Mert Saglam, and Gábor Tardos. Tight bounds for lp samplers, finding duplicates in streams, and related problems. In *Symposium on Principles of Database Systems (PODS)*, pages 49–58, 2011. 4

[JVV86]   Mark R Jerrum, Leslie G Valiant, and Vijay V Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theoretical computer science*, 43:169–188, 1986. 1

[KKP22]   John Kallaugher, Michael Kapralov, and Eric Price. Simulating random walks in random streams. In *Symposium on Discrete Algorithms (SODA)*, pages 3091–3126, 2022. 6

[Rei08]   Omer Reingold. Undirected connectivity in log-space. *Journal of the ACM (JACM)*, 55(4):1–24, 2008. 1

[Rou20]   Tim Roughgarden, editor. *Beyond the Worst-Case Analysis of Algorithms*. Cambridge University Press, 2020. 3

[Sch18]   Aaron Schild. An almost-linear time algorithm for uniform random spanning tree generation. In *Symposium on Theory of Computing (STOC)*, pages 214–227, 2018. 1

[SGP11]   Atish Das Sarma, Sreenivas Gollapudi, and Rina Panigrahy. Estimating pagerank on graph streams. *J. ACM*, 58(3):13:1–13:19, 2011. 1, 5

[ST13]   Daniel A Spielman and Shang-Hua Teng. A local clustering algorithm for massive graphs and its application to nearly linear time graph partitioning. *SIAM Journal on computing*, 42(1):1–26, 2013. 1

[Vit85]   Jeffrey Scott Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11(1):37–57, 1985. 3, 14

[VV17]   Gregory Valiant and Paul Valiant. An automatic inequality prover and instance optimal identity testing. *SIAM J. Comput.*, 46(1):429–455, 2017. 5