# Almost Optimal Super-Constant-Pass Streaming Lower Bounds for Reachability

Lijie Chen
MIT
USA

Gillat Kol
Princeton University
USA

Dmitry Paramonov
Princeton University
USA

Raghuvansh R. Saxena
Princeton University
USA

Zhao Song
Institute for Advanced Study
USA

Huacheng Yu
Princeton University
USA

## ABSTRACT

We give an *almost quadratic* $n^{2-o(1)}$ lower bound on the space consumption of any $o(\sqrt{\log n})$*-pass* streaming algorithm solving the (directed) *s-t reachability* problem. This means that any such algorithm must essentially store the entire graph. As corollaries, we obtain almost quadratic space lower bounds for additional fundamental problems, including maximum matching, shortest path, matrix rank, and linear programming.

Our main technical contribution is the definition and construction of *set hiding graphs*, that may be of independent interest: we give a general way of encoding a set $S \subseteq [k]$ as a directed graph with $n = k^{1+o(1)}$ vertices, such that deciding whether $i \in S$ boils down to deciding if $t_i$ is reachable from $s_i$, for a specific pair of vertices $(s_i, t_i)$ in the graph. Furthermore, we prove that our graph "hides" $S$, in the sense that no low-space streaming algorithm with a small number of passes can learn (almost) anything about $S$.

## CCS CONCEPTS

• **Theory of computation** → **Streaming models**; *Communication complexity*.

## KEYWORDS

Graph Streaming, Communication Complexity, Lower Bounds

**ACM Reference Format:**
Lijie Chen, Gillat Kol, Dmitry Paramonov, Raghuvansh R. Saxena, Zhao Song, and Huacheng Yu. 2021. Almost Optimal Super-Constant-Pass Streaming Lower Bounds for Reachability. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC '21), June 21–25, 2021, Virtual, Italy*. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3406325.3451038

## 1 INTRODUCTION

*Graph streaming algorithms* are designed to process massive graphs and have been studied extensively over the last two decades. This study is timely as huge graphs naturally arise in many modern applications, particularly in those with structured data representing the relationships between a set of entities (*e.g.*, friendships in a social network). A graph streaming algorithm is typically presented with a sequence of graph edges in an arbitrary order and it can read them one-by-one in the order in which they appear in the sequence. We want the algorithm to only make one or few passes through the edge sequence and use limited memory, ideally much smaller than the size of the graph.

Much of the streaming literature was devoted to the study of *one-pass* algorithms, and for many basic graph problems $\Omega(n^2)$ lower bounds were shown, where $n$ is the number of vertices. This implies that the trivial algorithm that stores the entire graph and then uses an offline algorithm to compute the output is essentially optimal. Such quadratic lower bounds were shown for maximum matching and minimum vertex cover [26, 31], *s-t* reachability and topological sorting [15, 26, 36], shortest path and diameter [26, 27], minimum or maximum cut [49], maximal independent set [4, 22], dominating set [6, 25], and many others.

Recently, the *multi-pass* streaming setting received quite a bit of attention. For some graph problems, it was shown that going from a single pass to even a few passes can reduce the memory consumption of a streaming algorithm dramatically. For example, *semi-streaming* algorithms (which are algorithms that only use $\widetilde{O}(n)$ space and are often considered "tractable") with few passes were designed for various graph problems previously shown to admit quadratic lower bounds for single pass streaming. These include a two-pass algorithm for minimum cut in undirected graphs [46], an $O(1)$-pass algorithm for approximate matching [29, 31, 38, 42], an $O(\log \log n)$-pass algorithm for maximal independent set [4, 22, 30], and $O(\log n)$-pass algorithms for approximate dominating set [6, 16, 35] and weighted minimum cut [44].

### 1.1 Our Main Result: Lower Bound for *s-t* Reachability

Our main result is a *near-quadratic* lower bound on the space complexity of any streaming algorithm that solves *s-t reachability* (a.k.a, *directed connectivity*) and uses $o(\sqrt{\log n})$ passes:

Theorem 1.1 (Reachability). *Any randomized $o(\sqrt{\log n})$-pass streaming algorithm that, given an $n$-vertex directed graph $G = (V, E)$ with two designated vertices $s, t \in V$, can determine whether there is a directed path from $s$ to $t$ in $G$ requires $n^{2-o(1)}$ space.*


*STOC '21, June 21–25, 2021, Virtual, Italy*

ACM ISBN 978-1-4503-8053-9/21/06.
https://doi.org/10.1145/3406325.3451038

The $s$-$t$ reachability problem is amongst the most basic graph problems[1] and was also one of the first to be studied in the context of streaming [36]. Prior to our work, an almost-quadratic lower bound was only known for *two*-pass streaming algorithms by the very recent breakthrough of [9]. Prior to that, a quadratic lower bound was shown for *one*-pass streaming [27, 36]. For $p$-pass streaming algorithms with $p \geq 3$, the best space lower bound was $\Omega(n^{1+1/(2p+2)})$ [34]. We mention that the hard instance constructed and analyzed by [9] is easy (admits a semi-streaming algorithm), even with only three passes. Additionally, the hard instance used by [34] to prove their lower bound against $p$-pass streaming algorithms can be solved in $n^{1+1/\Omega(p)}$ space with a single pass and with $\widetilde{O}(n)$ space with $p+1$ passes. Thus, Theorem 1.1 cannot be shown using the hard instances constructed by previous papers, and we indeed design a very different instance.

Using a slightly different instance, the techniques used to prove Theorem 1.1 also give a non-trivial lower bound for more than $o(\sqrt{\log n})$ passes. Specifically, we obtain a lower bound of $n^{1+1/O(\log\log n)}$ on the space used by any streaming algorithm with $o(\log n/(\log\log n)^2)$ passes (see Remark 5.2). For $p$ satisfying $p = \omega(\log\log n)$ and $p = o(\log n/(\log\log n)^2)$, this improves over the $\Omega(n^{1+1/(2p+2)})$ lower bound of [34]. Still, proving super-linear $n^{1+\varepsilon}$ space lower bounds for $n^{\varepsilon}$-pass streaming algorithms solving $s$-$t$ reachability with $\varepsilon > 0$ is a great problem that we leave open. (Note that with $O(n)$-passes, semi-streaming is possible by implementing a BFS search).

Since the $s$-$t$ reachability problem is a special case of the $s$-$t$ *minimum cut* problem in directed graphs, the lower bound in Theorem 1.1 can also be applied to minimum cut. We note that space efficient algorithms are known for the *undirected* versions of both these problems: $s$-$t$ connectivity (the undirected version of $s$-$t$ reachability) has a one-pass semi-streaming algorithm (*e.g.*, by maintaining a spanning forest [26]) and there is also a two-pass streaming algorithm for $s$-$t$ minimum cut in undirected graphs that only requires $O(n^{5/3})$ space ([46], see also [3]).

*Technique: Set Hiding.* We derive Theorem 1.1 as a special case of a more general framework: given a set $S \subseteq [k]$, we are able to construct a random graph $G_S$ that "*hides*" $S$, in the sense that for any two different sets $S$ and $S'$, no small-space streaming algorithm with a small number of passes can distinguish between $G_S$ and $G_{S'}$ with any reasonable advantage. The graph $G_S$ we construct has only $n = k^{1+o(1)}$ vertices, out of which $k$ are "designated source vertices" $U = \{u_1, \cdots, u_k\}$ and $k$ are "designated sink vertices" $V = \{v_1, \cdots, v_k\}$. There is a directed path from the source $u_i$ to the sink $v_i$ in $G_S$ if and only if $i \in S$. See Section 2 for a detailed sketch of this construction.

Theorem 1.1 now follows by the following argument: let $s := u_1$ and $t := v_1$ and observe that $G_{[k]}$ has a directed path from $s$ to $t$, while $G_\emptyset$ does not. This suggests that any algorithm for $s$-$t$ reachability can also distinguish between $G_{[k]}$ and $G_\emptyset$, violating the hiding property, which is impossible for a small-space algorithm with a small number of passes. In fact, this argument proves a stronger statement: the $s$-$t$ reachability problem remains hard even under the promise that in the $n$-vertex input graph, either there are no paths from $s$ to $t$ or there are at least $k = n^{1-o(1)}$ such paths, that are vertex disjoint[2]. This stronger statement allows us to obtain lower bounds for approximate versions of related graph problems (with modest, sub-constant approximation factors), as detailed below.

## 1.2 Lower Bounds for Other Streaming Problems

*Matching, shortest path, and rank.* As in the case of the two-pass lower bound for $s$-$t$ reachability proved by [9], Theorem 1.1 also implies multi-pass lower bounds for the *shortest path length*, *maximum bipartite matching size*, and *matrix rank* problems. This can be shown by (by now standard) reductions: $s$-$t$ reachability $\preceq$ shortest path, and $s$-$t$ reachability $\preceq$ maximum matching $\preceq$ matrix rank.

Theorem 1.2 (Shortest path). *Any randomized $o(\sqrt{\log n})$-pass streaming algorithm that, given an $n$-vertex* undirected *graph $G = (V, E)$ and two designated vertices $s, t \in V$, can output the length of the shortest path connecting $s$ and $t$ in $G$ requires $n^{2-o(1)}$ space.*

Theorem 1.3 (Matching). *Any randomized $o(\sqrt{\log n})$-pass streaming algorithm that, given an $n$-vertex* undirected *bipartite graph $G = (L \sqcup R, E)$ can determine whether $G$ has a perfect matching requires $n^{2-o(1)}$ space.*

Theorem 1.4 (Matrix rank). *Any randomized $o(\sqrt{\log n})$-pass streaming algorithm that, given the rows of a matrix $M \in \mathbb{F}_q^{n\times n}$ where $q = \omega(n)$, can determine whether the matrix has full rank requires $n^{2-o(1)}$ space.*

When it comes to lower bounds, the state of affairs for (exact) shortest path, maximum matching, and matrix rank is similar to that of $s$-$t$ reachability: $\Omega(n^2)$ for one-pass streaming [19, 26], $\Omega(n^{2-o(1)})$ for two passes [9], and $\Omega(n^{1+1/(2p+2)})$ for any $p \geq 3$ [34]. On the upper bound front, semi-streaming algorithms with $O(\sqrt{|E|})$ passes are known for (weighted) maximum matching [40], and with $O(\sqrt{n})$ passes for shortest path [18]. Understanding the pass-space tradeoffs for these problems is a great goal.

*Lower bounds for approximation algorithms.* Our proofs of the above theorems also give some non-trivial results in the approximation setting. Specifically, for constant $p$, our almost quadratic lower bounds continue to hold even for $p$-pass algorithms that only give a $(1+\omega(\log n)^{-2p})$-approximation to the length of the shortest $s$-$t$ path (see Theorem 5.1), or a $(1+2^{-\Omega_p(\sqrt{\log n})})$-approximation to the size of the maximum matching or to the rank of a given matrix (see Theorem 5.3 and Corollary 5.6). These lower bounds for approximate maximum matching and approximate matrix rank are possible because our lower bound for $s$-$t$ reachability holds even when there are many (vertex) disjoint paths from $s$ to $t$ (see Section 1.1). In the reduction from $s$-$t$ reachability to maximum matching, the number of such paths translates into the difference in the size of the maximum matchings that the streaming algorithm is unable to distinguish between.

[1] Time-space tradeoffs for $s$-$t$ reachability in various graph automata models were shown in [10, 23, 24].

[2] To get this, start with the graph $G_{[k]}$ and add two vertices, a global source $s$ and a global sink $t$. Add directed edges from $s$ to every $u_i$ and from every $v_i$ to $t$.

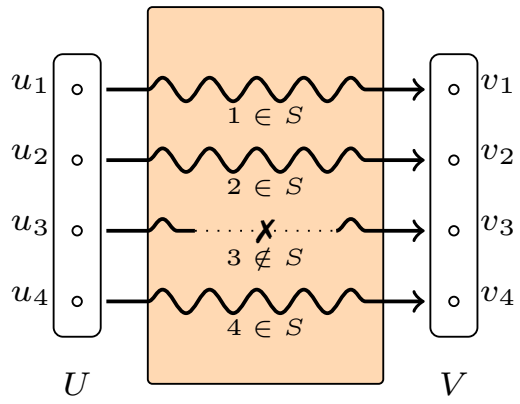



**Figure 1: A graph that encodes the set $\{1, 2, 4\}$ using the reachability from $U$ to $V$. Vertex $u_i$ cannot reach $v_j$ for $i \neq j$.**

Since $O_\varepsilon(1)$-pass semi-streaming algorithms for $(1+\varepsilon)$-approximations to the size of the maximum matching and single-source shortest path are known [11, 42], our above lower bounds for these problems cannot be strengthened to deal with constant $\varepsilon$. A polynomial (but sub-linear) lower bound of $n^{1-\varepsilon^{\Omega(1/p)}}$ on the space complexity of $p$-pass streaming algorithms that obtain a $(1+\varepsilon)$-approximation of maximum matching and of matrix rank was very recently proved by [8].

*Lower bounds for additional problems.* Via other known reductions, the lower bound in Theorem 1.1 can be shown to imply lower bounds for additional streaming problems, such as estimating the number of *reachable vertices* from a given source [36] and approximating the *minimum feedback arc set* [15].

We also consider the *linear programming feasibility* (LP feasibility) problem, where given a set of $n$ linear constraints (inequalities) over $d$ variables, one needs to decide if all constraints can be satisfied simultaneously. We prove that Theorem 1.1 implies a similar lower bound for the LP feasibility problem with $d \approx n$, see Theorem 5.7 (for the low dimension $d \ll n$ regime, see [5, 17]). To this end, we devise a reduction from $s$-$t$ reachability to LP feasibility that exploits the fact that our hard $s$-$t$ reachability instance is a layered graph[3].

## 2 TECHNICAL OVERVIEW

Our proof proceeds by designing a carefully structured hard instance for $s$-$t$ reachability. The key component in our lower bound proof is a construction that *hides* a set in a random (directed) graph from streaming algorithms. Specifically, let $S \subseteq [n]$ be a set, and $U, V$ be two sets of $n$ vertices. We will construct a random graph, possibly adding more vertices, such that $u_i$ (the $i$-th vertex in $U$) cannot reach $v_j$ (the $j$-th vertex in $V$) for any $i \neq j$; and $u_i$ can reach $v_i$ *if and only if* $i \in S$ (see Figure 1). That is, the graph encodes the set $S$ using the reachability from $U$ to $V$. The most important feature of this random graph construction is that (with a proper ordering of its edges in a stream) any $p$-pass (for some small $p$) low-space streaming algorithm $\mathcal{A}$ cannot "learn anything" about $S$, in the sense that for any $S_1$ and $S_2$, $\mathcal{A}$ cannot distinguish between the random graphs generated based on $S_1$ or $S_2$ except with probability at most $1/n$. We call such a random graph a Set-Hiding graph.[4]

Assuming such a graph construction, the $s$-$t$ reachability lower bound follows easily. To see this, we set $S_1 := \emptyset$ and $S_2 := \{1\}$, let the source $s$ be $u_1$ and the sink $t$ be $v_1$. Then in a Set-Hiding graph that hides $S_1$, $s$ cannot reach $t$; and in a Set-Hiding graph that hides $S_2$, $s$ can reach $t$. But any $p$-pass low-space streaming algorithm cannot distinguish between the two cases. In particular, it is impossible for such an algorithm to solve $s$-$t$ reachability. In the following, we will focus on the construction of such Set-Hiding graphs.

### 2.1 Set-Hiding Graphs Against One-Pass Algorithms

Let us for now set the goal to constructing graphs that hide a set $S$ from any low-space *one-pass* streaming algorithm, as a demonstration of the idea.

#### 2.1.1 *A New Communication Problem.*

*The problem.* It turns out that the indistinguishablility stems from the hardness of the following one-way communication problem:

- Alice gets $nK$ sets $(T_j^{(k)})_{(j,k)\in[n]\times[K]}$ (think of $K = \log n$), which are subsets of $[n]$;
- Bob gets $K$ indices $j_1, \ldots, j_K \in [n]$ and $K$ permutations $\pi_1, \ldots, \pi_K$ on $[n]$;
- Alice sends a single message to Bob, whose goal is to learn the set
$$S := \bigoplus_{k=1}^{K} \pi_k(T_{j_k}^{(k)}),$$
where $\oplus$ of sets is defined as the coordinate-wise XOR of their indicator vectors, and $\pi(T) := \{\pi(a) : a \in T\}$.

In other words, Alice gets $K$ collections of sets, each collection consists of $n$ sets, and each set is over $[n]$. Then Bob picks one set from each collection, permutes the sets according to his input, and he wishes to know the $\oplus$ of the $K$ permuted sets.

*Lower bound.* We prove that for *any* two sets $S_1$ and $S_2$, if Alice's message has only $n^{1.99}$ bits (note that her input has $n^2K$ bits), Bob is not able to distinguish between $S = S_1$ and $S = S_2$, except with probability exponentially small in $K$.[5] Note that we prove a much stronger form of lower bound than just a lower bound on the error probability of computing $S$, such an indistinguishability lower bound is crucial to obtain the Set-Hiding property of our graph construction.

[3] While there are known reductions from $s$-$t$ reachability to LP feasibility, to the best of our knowledge, our reduction is the only one that is both deterministic and generates $\Theta(n)$ dense constraints with super small coefficients ($\{0, 1, -1\}$ coefficients), as opposed to polynomially large ones.

[4] In the formal proof, the collection of $2^n$ such random graphs, one for each subset of $[n]$, is called a Set-Hiding generator, and a single (deterministic) graph encoding a set is called a Set-Encoding graph. We will not differentiate between the two when discussing the intuition in this section.

[5] Careful readers may have noticed that the statement as written here is technically false, as Alice could send the parity of the size for each set, taking $nK \ll n^{1.99}$ bits. In this case, Bob learns the parity of $S$, falsifying the statement for $S_1$, $S_2$ with different parities. In the actual proof, Alice's sets as well as Bob's permutations will be over $[4n]$, and the set $S$ is defined to be $\bigoplus_{k=1}^{K} \pi_k(T_{j_k}^{(k)})$ *restricted to the first* $n$ *elements* $[n]$. It resolves the above parity issue, and the indistinguishability holds in this case. The arguments below follow as well.

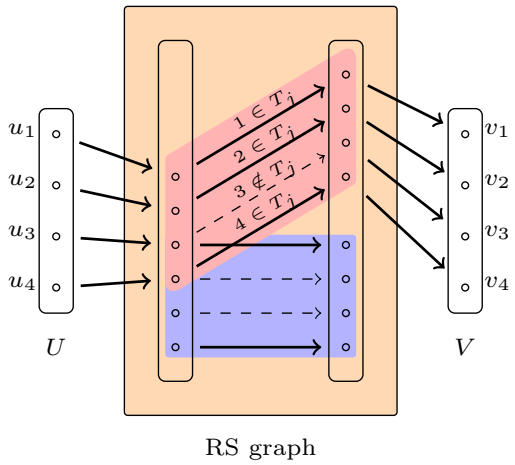
The communication lower bound proof uses an *XOR lemma* for the INDEX problem, which we prove in this paper. Suppose the players only focus on deciding whether $1 \in S$, then Alice's input can be viewed as $K$ arrays of length $n^2$, where the $k$-th array is the concatenation of the $n$ indicator vectors $\mathbb{1}(T_j^{(k)})$ for $j \in [n]$, and Bob's input chooses one entry from each array. The bit indicating $1 \in S$ is precisely the XOR of the $K$ chosen bits, i.e., the XOR of the $\pi_k^{-1}(1)$-th bit in $T_{j_k}^{(k)}$ for $k \in [K]$ (observe that both the index $j_k$ and the random permutation $\pi_k$ in the definition of the communication problem are needed to ensure that Alice doesn't know what entry is chosen by Bob in array $k$).

The standard INDEX lower bound shows that for one array, if the communication is less than $n^{1.99}$, from Bob's view the chosen bit is still close to uniform, with bias at most $n^{-\Omega(1)}$. If the players handle all $K$ arrays independently, then the $K$ chosen bits are independent from Bob's view. Therefore, their XOR has bias at most $n^{-\Omega(K)}$, by the standard result on the XOR of independently random bits. In general, an XOR lemma states that this bias bound holds even for generic protocols. We prove such an XOR lemma for INDEX by showing a discrepancy bound (a similar discrepancy bound was (implicitly) proved in [32, 33] using a different argument). Finally, we apply this XOR lemma and a hybrid argument to prove the indistinguishability of any two sets $S_1$ and $S_2$. See the full version for the XOR lemma for INDEX and the communication lower bound.

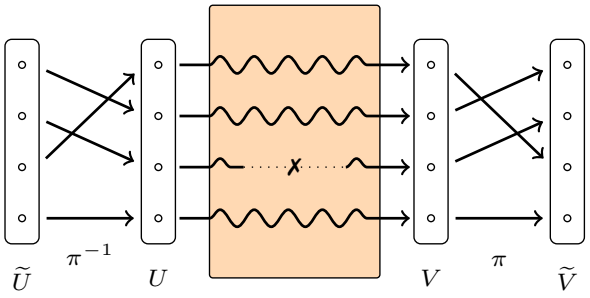
*2.1.2 Constructing the Set-Hiding Graph.* To construct the Set-Hiding graph that hides a set $S$, we first generate $(T_j^{(k)})_{(j,k)}, (j_k)_k, (\pi_k)_k$ according to the hard input distribution for the communication problem, *conditioned on* the final set being $S$, i.e., $S = \bigoplus_{k=1}^{K} \pi_k(T_{j_k}^{(k)})$. Then, we will construct a graph that mimics the computation of $\bigoplus_{k=1}^{K} \pi_k(T_{j_k}^{(k)})$, and use the hardness of this communication problem to argue that low-space streaming algorithms cannot learn anything about $S$.

*Representing index selection – graphs for $(T_1^{(k)}, \ldots, T_n^{(k)})$ and $T_{j_k}^{(k)}$.* To this end, we do this computation bottom-up, and let us first see how to "compute" the set $T_{j_k}^{(k)}$ for each $k$, i.e., *select* the $j_k$-th set from the collection $(T_1^{(k)}, \ldots, T_n^{(k)})$. This is done using a similar construction to [9], which uses the *Ruzsa-Szemerédi graphs* (RS graphs). The version we use is a bipartite graph on $m$ vertices, whose edges form a disjoint union of $t$ *induced* matchings of size $r$ (i.e., $r$ by $r$ bipartite subgraphs consisting of only $r$ matching edges). Such graphs were shown to exist for $r, t = m \cdot 2^{-\Theta(\sqrt{\log m})}$ [47].

For each $k$, we first fix such an RS graph with $r, t \geq n$. Then, we associate the $j$-th matching with set $T_j^{(k)}$, and keep the $i$-th edge in the matching if and only if $i \in T_j^{(k)}$. The RS graph encodes the collection $(T_1^{(k)}, \ldots, T_n^{(k)})$. Intuitively, we work with RS graphs as they allow us to "pack" many sets into a small graph. We select the $j_k$-th set by connecting two vertex sets $U$ and $V$ to the corresponding matching (see Figure 2a). In this way, $u_i \in U$ can reach $v_i \in V$ if and only if $i \in T_{j_k}^{(k)}$, i.e., the reachability from $U$ to $V$ *encodes* the selected set $T_{j_k}^{(k)}$ (in the same way as Set-Hiding graphs would encode $S$).




**(a)**




**(b)**

**Figure 2: In (a), the red and blue boxes correspond to two *induced* matchings in the RS graph, thin dashed edges exist in the original RS graph, but are removed according to $T_j$.**

*Representing permutations: graph for $\pi_k(T_{j_k}^{(k)})$.* Next, we implement the permutation by adding two more layers $\widetilde{U}$ and $\widetilde{V}$, and putting a matching corresponding to $\pi_k^{-1}$ from $\widetilde{U}$ to $U$, and a matching corresponding to $\pi_k$ from $V$ to $\widetilde{V}$. Then the reachability from $\widetilde{U}$ to $\widetilde{V}$ encodes $\pi_k(T_{j_k}^{(k)})$ (see Figure 2b).

*Representing XOR: graph for $\bigoplus_{k=1}^{K} \pi_k(T_{j_k}^{(k)})$.* The last step is to mimic the computation of $\oplus$ of $K$ sets. We have constructed $K$ graphs such that in $k$-th graph, the reachability from $U_k$ to $V_k$ encodes $\pi_k(T_{j_k}^{(k)})$. We wish to combine them into a single graph,

containing $U$ and $V$ as subsets of vertices, such that the reachability from $U$ to $V$ encodes $\oplus_{k=1}^{K} \pi_k(T_{j_k}^{(k)})$. The main idea is to use the fact that there exists an $\{\wedge, \vee, \neg\}$-*formula* of size $O(K^2)$ that computes the XOR of $K$-bit. For $x_1, \ldots, x_K \in \{\text{True}, \text{False}\}$, we can write $x_1 \oplus x_2 \oplus \cdots \oplus x_K$ as a small Boolean formula $F$ which only uses AND, OR and NOT gates. Moreover, we can assume that the NOTs are only applied on the $x_i$. $F$ will be applied to the sets $\pi_k(T_{j_k}^{(k)})$ *coordinate-wisely*, computing $\oplus_{k=1}^{K} \pi_k(T_{j_k}^{(k)})$. We are going to construct the graph recursively according to $F$.

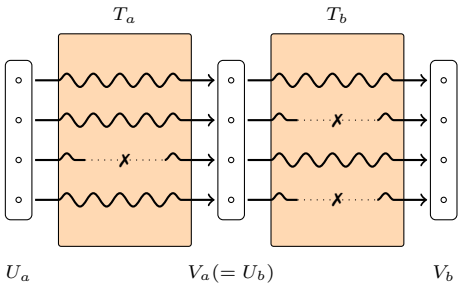
For an AND gate in $F$, suppose its two operands are $T_a$ and $T_b$, and we have constructed a graph containing $U_a$ and $V_a$ as subsets of vertices that encodes $T_a$ using the reachability from $U_a$ to $V_a$, and a graph containing $U_b$ and $V_b$ that encodes $T_b$ using the reachability from $U_b$ to $V_b$. Then the coordinate-wise AND of $T_a$ and $T_b$ (equivalently, the intersection) can be computed by merging $V_a$ and $U_b$ into one set (see Figure 3a). For an OR gate in $F$ with two operands $T_a$ and $T_b$, their coordinate-wise OR (equivalently, the union) can be computed by merging $U_a$ and $U_b$ into one set, and merging $V_a$ and $V_b$ into one set (see Figure 3b). Eventually, we either reach an input variable corresponding to a graph that encodes one $\pi_k(T_{j_k}^{(k)})$, which we have already constructed, or reach the *negation* of an input variable, which corresponds to the *complement* of one $\pi_k(T_{j_k}^{(k)})$. It suffices to also construct a graph that encodes $[n] \setminus \pi_k(T_{j_k}^{(k)})$ for each $k$. Note that $[n] \setminus \pi_k(T_{j_k}^{(k)}) = \pi_k([n] \setminus T_{j_k}^{(k)})$. Therefore, this can be done by applying the construction in the last paragraph on the complement of input sets $([n] \setminus T_j^{(k)})_{(j,k)}$ (and with the same indices $j_k$ and permutations $\pi_k$).

*The order of edges in the stream.* The above construction generates a graph that encodes the set $S = \bigoplus_{k=1}^{K} \pi_k(T_{j_k}^{(k)})$, which we wanted to hide. To determine the order of its edges in the stream, observe that the edges in all RS (sub)graphs only depend on the sets $(T_j^{(k)})_{(j,k)}$, and the rest of the graph only depends on the indices $(j_k)_k$ and permutations $(\pi_k)_k$. Hence, in the stream, we will first give all edges in the RS graphs, then all remaining edges. By the standard reduction from one-way communication to streaming algorithms and the hardness of the communication problem, we prove that $S$ is hidden from any $n^{1.99}$-space one-pass streaming algorithm.

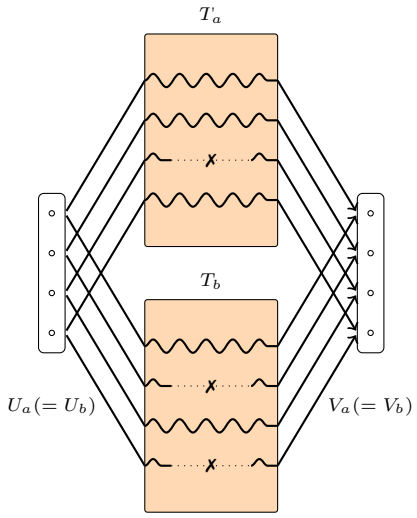
## 2.2 Generalizing to $p$ Passes

*Hiding the selected sets.* The lower bound for the one-way communication problem uses the fact that Alice does not know the indices and permutations. Equivalently, the streaming algorithm does not know the parts encoding $(j_k)_k$ and $(\pi_k)_k$ when it sees the RS graphs, which encode $(T_j^{(k)})_{(j,k)}$. However, this is not the case if the algorithm can read the stream even just twice, as it can remember the indices and permutations in the first pass so that the second time it sees the RS graphs, it already knows which sets are selected. To generalize our hard instance to $p$ passes, the main idea is to also *hide* the indices and permutations, from $(p-1)$-pass streaming algorithms.

More specifically, we wish to construct subgraphs (gadgets) that serve the same purposes as the parts encoding the indices and




**(a)**




**(b)**

**Figure 3: (a) shows a graph computing $T_a \wedge T_b$, and (b) shows a graph computing $T_a \vee T_b$.**

permutations (in terms of reachability), but additionally, for any $(j_k)_k, (\pi_k)_k$ and $(j'_k)_k, (\pi'_k)_k$, any low-space $(p-1)$-pass algorithm should not be able to distinguish between the subgraphs constructed based on them. Suppose we have such gadgets, then we may apply the one-way communication lower bound *to the $p$-th pass* (after replacing the edges from $U$ to the RS graph and from the RS graph to $V$ in Figure 2a by such gadgets, and replacing the edges from $\widetilde{U}$ and $U$ and the edges from $V$ to $\widetilde{V}$ in Figure 2b). This is because when the streaming algorithm sees $(T_j^{(k)})_{(j,k)}$ for the $p$-th time, it has only scanned the parts encoding the indices and permutations $p-1$ times (recall that $(T_j^{(k)})_{(j,k)}$ appears before all indices and

permutations in the stream), and is not able to learn anything about them. Therefore, the one-way communication lower bound still holds.

Perm-Hiding *graphs.* To construct such subgraphs, we construct a gadget that allows us to hide each permutation or index separately. To be more precise, given a permutation $\pi$ on $[n]$, we want to construct a (random) graph containing $X$ and $Y$ as subsets of vertices, such that for each $i \in [n]$, the only vertex in $Y$ that $u_i$ can reach is $v_{\pi(i)}$ (the "indices" are special cases of the "permutations", and they can also be hidden using such gadgets, see the full version). Moreover, for any $\pi_1, \pi_2$, any $(p-1)$-pass low-space streaming algorithm cannot distinguish between the graphs generated from $\pi_1$ and $\pi_2$. We call such random graphs the Perm-Hiding graphs.

*Hiding structured permutations via* Set-Hiding *graphs.* We first show that (assuming $n$ is even) if $\pi$ is structured such that for each $i$, either $\pi(2i) = 2i$ and $\pi(2i+1) = 2i+1$, or $\pi(2i) = 2i+1$ and $\pi(2i+1) = 2i$ (i.e., for each $i$, $\pi$ either swaps $2i$ and $2i+1$, or maps both to themselves), then we can construct a Perm-Hiding graph for $\pi$ using the Set-Hiding graphs against $(p-1)$-pass streaming algorithms. To see this, we add two extra layers $\widetilde{X}, \widetilde{Y}$ of sizes $2n$ between $X$ and $Y$. Denote the vertices in $\widetilde{X}$ and $\widetilde{Y}$ by $\widetilde{x}_{j,1}, \widetilde{x}_{j,2}$ and $\widetilde{v}_{j,1}, \widetilde{v}_{j,2}$ respectively for $j \in [n]$. For all $i$, we add the following edges from $X$ to $\widetilde{X}$ and from $\widetilde{Y}$ to $Y$:

- from $x_{2i}$ to $\widetilde{x}_{2i,1}, \widetilde{x}_{2i,2}$, from $x_{2i+1}$ to $\widetilde{x}_{2i+1,1}, \widetilde{x}_{2i+1,2}$, and
- from $\widetilde{y}_{2i,1}, \widetilde{y}_{2i+1,1}$ to $y_{2i}$, from $\widetilde{y}_{2i,2}, \widetilde{y}_{2i+1,2}$ to $y_{2i+1}$.

Now if we add an edge from $\widetilde{x}_{2i,1}$ to $\widetilde{y}_{2i,1}$ and an edge from $\widetilde{x}_{2i+1,2}$ to $\widetilde{y}_{2i+1,2}$, then $x_{2i}$ reaches $y_{2i}$ and $x_{2i+1}$ reaches $y_{2i+1}$ (see Figure 4a); if we add an edge from $\widetilde{x}_{2i,2}$ to $\widetilde{y}_{2i,2}$ and an edge from $\widetilde{x}_{2i+1,1}$ to $\widetilde{y}_{2i+1,1}$, then $x_{2i}$ reaches $y_{2i+1}$ and $x_{2i+1}$ reaches $y_{2i}$ (see Figure 4b). In the other words, such a permutation can always be implemented by placing a set of *parallel* edges from $\widetilde{X}$ to $\widetilde{Y}$. Therefore, to hide $\pi$, it suffices to put a Set-Hiding graph between $\widetilde{X}$ and $\widetilde{Y}$ to hide the corresponding set over $[2n]$. Since by the guarantee of Set-Hiding graphs, no low-space algorithm can distinguish between any two sets, we prove that any such two permutations cannot be distinguished.

Similarly, if there is a set of *fixed* $\leq n/2$ *disjoint* pairs of coordinates such that $\pi$ may only swap two coordinates in a pair, then the same argument from the last paragraph shows that such permutations can be hidden from $(p-1)$-pass streaming algorithms as well, assuming Set-Hiding graphs.

*Hiding general permutations.* Finally, we use the fact that there exists $d = O(\log n)$ *fixed* sets of $\leq n/2$ disjoint pairs,[6] such that *every* permutation $\pi$ can be decomposed into $\pi = \pi_d \circ \cdots \circ \pi_2 \circ \pi_1$, where $\pi_i$ may only swap (a subset of) the pairs in the $i$-th set (e.g., this is a corollary of the existence of $O(\log n)$-depth sorting networks [1]). The final Perm-Hiding graph consists of $d$ blocks concatenated with identity matchings, where the $i$-th block applies the construction from the last paragraph to swap pairs in the $i$-th set. For each $\pi_i$, we hide a set in the block using Set-Hiding. By a standard hybrid argument, we conclude that no two permutations can be distinguished.

[6] It is important that the sets do not depend on $\pi$.

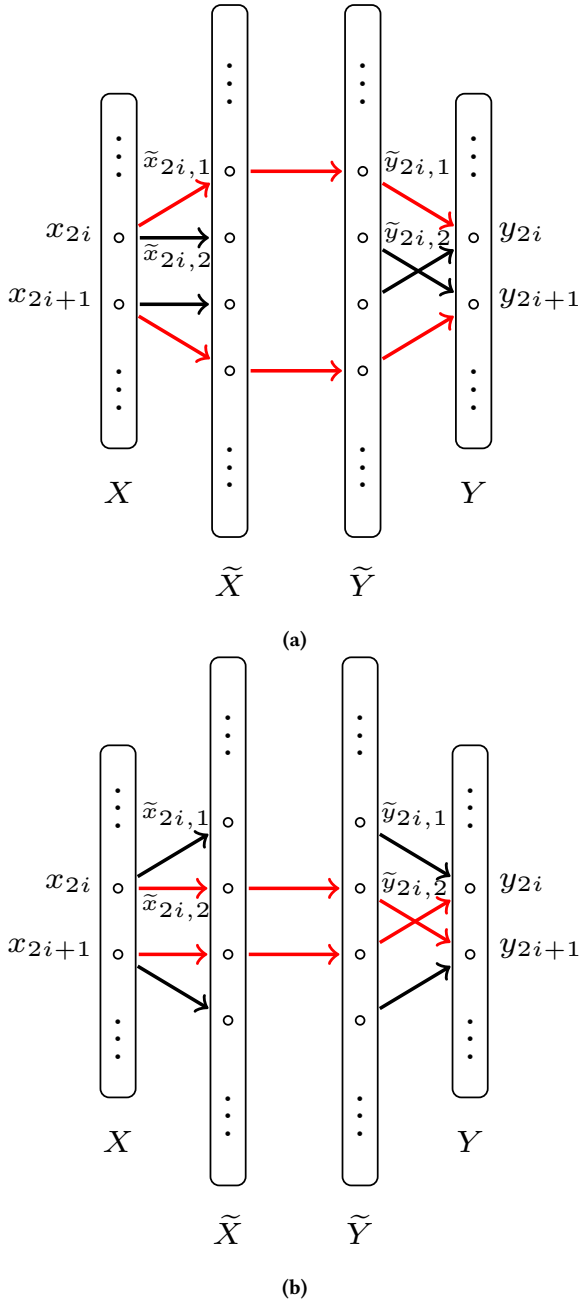



**Figure 4: (a) shows the graph that does not swap $2i$ and $2i+1$, (b) shows the graph that swaps $2i$ and $2i+1$. The edges from $X$ to $\widetilde{X}$ and the edges from $\widetilde{Y}$ to $Y$ are fixed.**

*Putting it together.* Overall, the $p$-pass Set-Hiding graphs use the structure from Section 2.1, together with $(p-1)$-pass Perm-Hiding graphs. The $(p-1)$-pass Perm-Hiding graphs, in turn, use $(p-1)$-pass Set-Hiding graphs, which are constructed *recursively*. One may verify that the size of the graph blows up by a factor of $2^{\Theta(\sqrt{\log n})}$ in each level of recursion, due to the parameters in RS graphs. Hence,

when $p = o(\sqrt{\log n})$, the final graph size $N$ is at most $n^{1+o(1)}$, implying the space lower bound of $N^{1.99}$. See the full version for the formal construction of Set-Hiding graphs, the construction of Perm-Hiding graphs, and the recursive construction that combines them.

# 3 PRELIMINARY

## 3.1 Notation

We often use bold font letters (e.g., $\boldsymbol{X}$) to denote random variables, and calligraphic font letters (e.g., $\mathcal{X}$) to denote distributions. For two random variables $\boldsymbol{X}$ and $\boldsymbol{Y}$, and for $Y \in \mathrm{supp}(\boldsymbol{Y})$, we use $(\boldsymbol{X}|\boldsymbol{Y} = Y)$ to denote $\boldsymbol{X}$ conditioned on $\boldsymbol{Y} = Y$. For two lists $a$ and $b$, we use $a \circ b$ to denote their concatenation.

For two distributions $\mathcal{D}_1$ and $\mathcal{D}_2$ on set $\mathcal{X}$ and $\mathcal{Y}$ respectively. We use $\mathcal{D}_1 \otimes \mathcal{D}_2$ to denote their product distribution over $\mathcal{X} \times \mathcal{Y}$, and $\|\mathcal{D}_1 - \mathcal{D}_2\|_{\mathrm{TV}}$ to denote the total variation distance between them.

Let $n \in \mathbb{N}$. We use $[n]$ to denote the set $\{1, \ldots, n\}$. For two sets $A, B \subseteq [n]$, we use $A \wedge B$ and $A \vee B$ to denote the intersection and the union of $A$ and $B$, respectively. We also use $\neg_n A$ to denote the set $[n] \setminus A$, and $A \oplus B$ to denote the set of elements appearing in exactly one of $A$ and $B$ (i.e., the symmetric difference of the sets $A$ and $B$). Note that

$$A \oplus B = (A \wedge \neg_n B) \vee (\neg_n A \wedge B).$$

When it is clear from the context, we drop the subscript in $\neg_n$ for simplicity.

We also use $\mathsf{Perm}([n])$ to denote the set of permutations on $[n]$. For a predicate $P$, we use $\mathbb{1}(P)$ to denote the corresponding Boolean value of $P$, that is, $\mathbb{1}(P) = 1$ if $P$ is true, and 0 otherwise.

## 3.2 Layered Graphs and Layer-Arrival Model

In this paper we will mostly consider directed layered graphs such that edges are always from one layer to its succeeding layer. We will also associate an **edge-layer ordering** to the layered graph $G$, which will be very convenient when we are working with graph streaming algorithms.

*Directed Layered Graphs.* Formally, a **directed layered graph** $G$ is a triple $(\vec{V}, \vec{E}, \vec{\ell})$, such that:

- $\vec{V} = (V_i)_{i=1}^{k}$ is the collection of $G$'s layers, where $k$ is the number of layers in $G$;
- $\vec{\ell} = (\ell_i)_{i=1}^{k-1}$ and $\vec{E} = (E_i)_{i=1}^{k-1}$ is a list of disjoint sets of edges on the vertex set $V = \bigcup_{i=1}^{k} V_i$. For each $i \in [k-1]$, $E_i$ is the set of all the edges in $G$ between $V_{\ell_i}$ and $V_{\ell_i+1}$. All the $\ell_i$ are distinct integers in $[k-1]$.

For each $i \in [k-1]$, we call the set of edges between $V_i$ and $V_{i+1}$ the $i$-th **edge-layer** of $G$. That is, $\vec{\ell}$ specifies an ordering of edge-layers of $G$, we will call it the edge-layer ordering of $G$. We remark that unless some edge-layers are empty, the edge list vector $\vec{E}$ always uniquely determine the ordering $\vec{\ell}$. In most cases we will just specify the edge list vector $\vec{E}$ and the $\vec{\ell}$ will be determined from the context.

We will use $E(G), \vec{V}(G), k(G)$ and $\vec{\ell}(G)$ to denote the set of edges, the list of layers of $G$, the number of layers in $G$ and the edge-layer ordering of $G$, respectively. For $i \in [k]$, we use $V_i(G)$ to denote the vertex set of the $i$-th layer of $G$. We also use $V(G)$ to denote $\bigcup_{i=1}^{k} V_i(G)$.

We say a layered graph $G$ is an $(N_G, k_G, \vec{\ell}_G)$ graph, if $G$ has $N_G$ vertices, $k_G$ layers and its edge-layer ordering is $\vec{\ell}_G$.

For a layered graph $G = ((V_i)_{i=1}^{k}, E)$, we use $\mathsf{First}(G)$ to denote $V_1$ and $\mathsf{Last}(G)$ to denote $V_k$ for convenience. For each layer, we index all the vertices by consecutive integers starting from 1. For a set $S$ of vertices from a single layer of $G$ (that is, $S \subseteq V_i$ for some $i \in [k]$), we use $S_{[i]}$ to denote the vertex with the $i$-th smallest index in $S$.

We note that a directed bipartite graph (all edges go from left side to the right side) is a directed layered graph with two layers (for which the list $\vec{E}$ only contains one set of all edges in the graph, and $\vec{\ell} = (1)$). Unless explicitly stated otherwise, we will always use layered graphs or bipartite graphs to refer to their directed versions. (The only place we study undirected graph is in Section 5.1 and Section 5.2.)

*Concatenation of two layered graphs.* For two layered graphs $G_1, G_2$ such that $|\mathsf{Last}(G_1)| = |\mathsf{First}(G_2)|$, we use $H = G_1 \odot G_2$ to denote their concatenation by identifying $\mathsf{Last}(G_1)$ and $\mathsf{First}(G_2)$. That is, for each $i \in [|\mathsf{Last}(G_1)|]$, we identify the vertex $\mathsf{Last}(G_1)_{[i]}$ and $\mathsf{First}(G_2)_{[i]}$. We also set $\vec{E}(H) = \vec{E}(G_1) \circ \vec{E}(G_2)$ to specify the edge-layer ordering of $H$.

*The layer-arrival model.* Our lower bounds actually holds for the *layer-arrival* setting, which is stronger than the usually studied edge-arrival or vertex-arrival models. In the following we formally define this model.

Definition 3.1 (Layer-arrival model). *Given a layered graph $G = (\vec{V}, \vec{E}, \vec{\ell})$ of $k$ layers, a randomized $p$-pass streaming algorithm $A$ with space $s$ in the layer-arrival setting works as follows:*

- *The algorithm makes $p$-pass over the graph, each pass has $(k-1)$ phases. Hence, there are $(k-1) \cdot p$ phases in total. The algorithm starts with memory state $w_0 = 0^s$. Additionally, at the beginning $A$ can draw an unbounded number of random bits, from a fixed distribution $\mathcal{D}_{\mathsf{rand}}$. These random bits are read-only and $A$ can always access them freely.*[7]
- *For $i \in [p]$ and $j \in [k-1]$, let $t = (i-1) \cdot p + j$. In the $t$-phase, $A$ can use unlimited computational resource to compute another state $w_t$ of $s$ bits, given the previous state $w_{t-1}$ together with the edge set $E_j$. (Note that $w_t$ is indeed a random variable depending on $w_{t-1}$ and $E_j$, since $A$ is randomized).*
- *Finally, $A$'s output only depends on the last state $w_{p(k-1)}$ and its random bits.*

In other words, the streaming algorithm is allowed to access the graph layer by layer, and can use unlimited computational resources to process each layer. The only constraint is that it can restore at most $s$ bits of information after processing one layer.

Clearly, lower bounds for graph streaming algorithms in the layer-arrival model immediately imply the same lower bounds for graph streaming algorithms in the edge-arrival model or vertex-arrival model.

[7] That is, randomness is free for $A$ and are not charged in the space complexity of $A$. This is very important for the hybrid argument used in this paper, see Section 3.4.

### 3.3 Ruzsa-Szemerédi Graphs

A bipartite graph $G^{\text{RS}} = (L \sqcup R, E)$ is a called an $(r,t)$-Ruzsa-Szemerédi graph (RS graph for short) iff its edge-set $E$ can be partitioned into $t$ *induced matchings* $M_1, \ldots, M_t$, each of size $r$.

We will use the original construction of RS graphs due to Ruzsa and Szemerédi [47] based on the existence of large sets of integers with no 3-term arithmetic progression, proven by Behrend [12].

Proposition 3.2 ([47]). *There is an absolute constant $c^{\text{RS}} \geq 1$ such that, for all sufficiently large $n$, there is an integer $N \leq n^{1+c^{\text{RS}}/\sqrt{\log n}}$ such that there are $(n,n)$-RS graphs with $N$ vertices on each side of the bipartition.*

For convenience, we define $N^{\text{RS}}(n) = n^{1+c^{\text{RS}}/\sqrt{\log n}}$. We also need the following construction of RS graphs with different parameters by [28].

Proposition 3.3 ([28]). *There is an absolute constant $c_2^{\text{RS}} > 0$ such that, for all sufficiently large $n$, there are $(n, n^{c_2^{\text{RS}}/\log\log n})$-RS graphs with $4n$ vertices on each side of the bipartition.*

### 3.4 Indistinguishability and The Hybrid Argument

We say two distributions on layered graphs $\mathcal{D}_1$ and $\mathcal{D}_2$ are $\varepsilon$-indistinguishable for $p$-pass streaming algorithms with space $s$ in the layer-arrival model. If for every $p$-pass streaming algorithm $A$ with space $s$ in the layer-arrival model

$$\|A(\mathcal{D}_1) - A(\mathcal{D}_2)\|_{\text{TV}} \leq \varepsilon,$$

where for each $i \in [2]$, $A(\mathcal{D}_i)$ is the output distribution of $A$ given an input graph drawn from $\mathcal{D}_i$.

Note that the above notation of indistinguishability also generalizes to streaming algorithms in the edge-arrival model or vertex-arrival model. But throughout this paper we will mostly study indistinguishability with respect to multi-pass streaming algorithms in the layer-arrival model. Hence, we will just omit this model name whenever it is clear from the context.

Given $t$ layered graphs $G_1, \ldots, G_t$. They can be treated as a single input to a $p$-pass streaming algorithm $A$ as follows: there are $p$ passes, in each pass $A$ process (the edges of) $G_1, \ldots, G_t$ in order. We use $(G_1, \ldots, G_k)_{\text{seq}}$ to denote this new input to $A$.

The following lemma shows that the standard hybrid argument also applies to the setting of multi-pass graph streaming algorithms. The hybrid argument will be used throughout our proofs, and we give a proof here for completeness.

Lemma 3.4 (Hybrid argument for multi-pass streaming algorithms). *Let $k$ be a positive integer. Let $\varepsilon \in \mathbb{R}^k_{\geq 0}$ denote $k$ parameters. Let $(\mathcal{D}_1, \mathcal{D}'_1), (\mathcal{D}_2, \mathcal{D}'_2), \ldots, (\mathcal{D}_k, \mathcal{D}'_k)$ be $k$ pairs of distributions over graphs. Suppose for each $i \in [k]$, $\mathcal{D}_i$ and $\mathcal{D}'_i$ is $\varepsilon_i$-indistinguishable for $p$-pass streaming algorithms with space $s$, then $(\mathcal{D}_1, \ldots, \mathcal{D}_k)_{\text{seq}}$ and $(\mathcal{D}'_1, \ldots, \mathcal{D}'_k)_{\text{seq}}$ are $\|\epsilon\|_1$-indistinguishable for $p$-pass streaming algorithms with space $s$.*[8]

Proof. Our proof is based on a standard hybrid argument. For each $j \in \{0, 1, \ldots, k\}$, let

$$\mathcal{H}_j = (\mathcal{D}_1, \ldots, \mathcal{D}_j, \mathcal{D}'_{j+1}, \ldots, \mathcal{D}'_k)_{\text{seq}}$$

Observe that $\mathcal{H}_0 = (\mathcal{D}'_1, \ldots, \mathcal{D}'_k)_{\text{seq}}$ and $\mathcal{H}_k = (\mathcal{D}_1, \ldots, \mathcal{D}_k)_{\text{seq}}$. Let $A$ be a $p$-pass streaming algorithms with space $s$.

We claim that for each $j \in [k]$,

$$\|A(\mathcal{H}_j) - A(\mathcal{H}_{j-1})\|_{\text{TV}} \leq \varepsilon_j.$$

Assuming the claim above holds, the lemma follows from the triangle inequality.

To prove the claim above, we show how to construct another streaming algorithm $B$ with the same pass and space complexity as $A$, such that $\|A(\mathcal{H}_j) - A(\mathcal{H}_{j-1})\|_{\text{TV}} = \|B(\mathcal{D}_j) - B(\mathcal{D}'_j)\|_{\text{TV}}$. Given an input graph $G$, $B$ first draws sample graphs $G_1 \sim \mathcal{D}_1, \ldots, G_{j-1} \sim \mathcal{D}_{j-1}$, and then draws sample graphs $G_{j+1} \sim \mathcal{D}'_{j+1}, \ldots, G_k \sim \mathcal{D}'_k$. $B$ then simulates $A$ on the input $(G_1, G_2, \ldots, G_k)$.

Recall that our definition of randomized streaming algorithms (see Definition 3.1) allows unbounded randomness from any fixed distribution (which are independent from the input distribution), and the random bits are not counted in space usage. The $k-1$ sample graphs of $B$ are then regarded as $B$'s randomness. Hence, $B$ has the same pass and space complexity of $A$. Moreover, one can see that $B(\mathcal{D}_j)$ distributes as $A(\mathcal{H}_j)$ and $B(\mathcal{D}'_j)$ distributes as $A(\mathcal{H}_{j-1})$. Since $\mathcal{D}_j$ and $\mathcal{D}'_j$ are $\varepsilon_j$-indistinguishable for $p$-pass streaming algorithms with space $s$, we have

$$\|A(\mathcal{H}_j) - A(\mathcal{H}_{j-1})\|_{\text{TV}} = \|B(\mathcal{D}_j) - B(\mathcal{D}'_j)\|_{\text{TV}} \leq \varepsilon_j,$$

which completes the proof of the claim. □

### 3.5 Set-Encoding/Perm-Encoding Graphs and Set-Hiding/Perm-Hiding Generators

*3.5.1* Set-Encoding *Graphs and* Perm-Encoding *Graphs.* The following two special layered graphs will be studied throughout the paper.

Definition 3.5 (Set-Encoding Graphs and Perm-Encoding Graphs).

*(1) (*Set-Encoding *Graphs) For a set $S \subseteq [n]$, we say a layered graph $G$ with first and last layer each having exactly $n$ vertices is a* Set-Enc$_n(S)$ *graph (i.e., a* Set-Encoding *graph for the set $S$). If for each $(i,j) \in [n] \times [n]$,* First$(G)_{[i]}$ *can reach* Last$(G)_{[j]}$ *if and only if $i = j$ and $i \in S$.*

*(2) (*Perm-Encoding *Graphs) For a permutation $\pi : [n] \to [n]$, we say a layered graph $G$ with first and last layer each having exactly $n$ vertices is a* Perm-Enc$_n(\pi)$ *graph (i.e., a* Perm-Encoding *graph for the permutation $\pi$). If for each $(i,j) \in [n] \times [n]$,* First$(G)_{[i]}$ *can reach* Last$(G)_{[j]}$ *if and only if $\pi(i) = j$.*

*3.5.2* Set-Hiding *Generators and* Perm-Hiding *Generators.* Note that a single Set-Encoding graph (resp. Perm-Encoding graph) just encodes a set, and does not hide it. Now we formally define Set-Hiding generators and Perm-Hiding generators, which generate *distributions* over Set-Enc/Perm-Enc graphs that hides the encoded set/permutation from multi-pass streaming algorithms.

[8] $(\mathcal{D}_1, \ldots, \mathcal{D}_k)_{\text{seq}}$ denotes the distribution obtained by for each $i \in [k]$, independently drawing $D_i \leftarrow \mathcal{D}_i$, and outputting $(D_1, \ldots, D_k)_{\text{seq}}$.

Definition 3.6 ($\varepsilon$-secure Set-Hiding generators). *Let $n \in \mathbb{N}$, and let $\mathcal{G}$ be a function from subsets of $[n]$ to distributions over layered graphs. We say $\mathcal{G}$ is $\varepsilon$-*Set-Hiding *for subsets of $[n]$ against $p$-pass algorithms with space $s$, if the following statements hold:*

*(1) For every $S \subseteq [n]$, $\mathcal{G}(S)$ is a distribution over* Set-Enc$_n(S)$ *graphs.*
*(2) For every two sets $S, T \subseteq [n]$, the distributions $\mathcal{G}(S)$ and $\mathcal{G}(T)$ are $\varepsilon$-indistinguishable for $p$-pass streaming algorithms with space $s$.*

Definition 3.7 ($\varepsilon$-secure Perm-Hiding generators). *Let $n \in \mathbb{N}$, and let $\mathcal{G}$ be a function from* Perm$([n])$ *to distributions over layered graphs. We say $\mathcal{G}$ is $\varepsilon$-*Perm-Hiding *for permutations in* Perm$([n])$ *against $p$-pass algorithms with space $s$, if the following statements hold:*

*(1) For every $\pi \in$ Perm$([n])$, $\mathcal{G}(\pi)$ is a distribution over* Perm-Enc$_n(\pi)$ *graphs.*
*(2) For every two permutations $\pi_1, \pi_2 \in$ Perm$([n])$, the distributions $\mathcal{G}(\pi_1)$ and $\mathcal{G}(\pi_2)$ are $\varepsilon$-indistinguishable for $p$-pass streaming algorithms with space $s$.*

For a generator $\mathcal{G}$ as in Definition 3.6 and Definition 3.7, we say $\mathcal{G}$ always outputs $(N_{\mathcal{G}}, k_{\mathcal{G}}, \vec{\ell}_{\mathcal{G}})$ graphs, if for all possible inputs $x$, the distribution $\mathcal{G}(x)$ is supported on $N_{\mathcal{G}}$-vertex layered graphs with $k_{\mathcal{G}}$ layers and edge-layer ordering $\vec{\ell}_{\mathcal{G}}$.

Remark 3.8. *The property that $\mathcal{G}$ always outputs $(N_{\mathcal{G}}, k_{\mathcal{G}}, \vec{\ell}_{\mathcal{G}})$ graphs for some triple $(N_{\mathcal{G}}, k_{\mathcal{G}}, \vec{\ell}_{\mathcal{G}})$ is pretty strong since it forces $\mathcal{G}$ to always output graphs with the same number of vertices, the same number of layers and the same edge-layer ordering. We remark here that all our constructions in this paper have this property.*

For simplicity, we will often use $\mathcal{G}^{\mathsf{SH}}_n$ (resp. $\mathcal{G}^{\mathsf{PH}}_n$) to denote an $\varepsilon$-Set-Hiding (resp. $\varepsilon$-Perm-Hiding) generator $\mathcal{G}$ for subsets of $[n]$ (resp. permutations in Perm$([n])$). We may also write $\mathcal{G}^{\mathsf{SH}}_{n,p}$ (resp. $\mathcal{G}^{\mathsf{PH}}_{n,p}$) to indicate that the generator is against $p$-pass streaming algorithms.

## 4 CONSTRUCTION OF Set-Hiding GENERATORS

In this section, we will give a construction overview of the Set-Hiding generators, which summarizes some key technical lemmas which will be proved in the later sections.

Now we are ready to state the main theorem of this section.

Theorem 4.1 (Main Theorem). *There is a constant $c > 1$ and an integer $N_0 \in \mathbb{N}$ such that for every $p \in \mathbb{N}$ and every integer $n$ satisfying $n \geq N_0$ and $p \leq c^{-1} \cdot \sqrt{\log n}$, there is a generator $\mathcal{G}^{\mathsf{SH}}_{n,p}$ such that: (1) it always outputs $(N_{\mathcal{G}^{\mathsf{SH}}}, k_{\mathcal{G}^{\mathsf{SH}}}, \vec{\ell}_{\mathcal{G}^{\mathsf{SH}}})$ graphs, where $N_{\mathcal{G}^{\mathsf{SH}}} \leq c \cdot n^{1+cp/\sqrt{\log n}}$ and $k_{\mathcal{G}^{\mathsf{SH}}} \leq c \cdot (c \log n)^{2p}$; (2) it is $(n^{-1})$-*Set-Hiding *against $p$-pass streaming algorithms with space $n^2$.*

We remark that Theorem 4.1 is all we need to prove the lower bounds for streaming algorithms in Section 5. The rest of this section is a proof of Theorem 4.1, with key technical lemmas proved in later sections.

*Overview of the construction.* Our construction works recursively. The base case will be generators against 0-pass streaming algorithms. Clearly, trivial constructions suffice for this base case since 0-pass streaming algorithms cannot read the input at all.

Next, for the case against $p$-pass streaming algorithms, Lemma 4.2 shows how to construct Set-Hiding generators against $p$-pass streaming algorithms from Perm-Hiding generators for $(p-1)$-pass streaming algorithms, and Lemma 4.3 shows how to construct Perm-Hiding generators against $p$-pass streaming algorithms from Set-Hiding generators for $p$-pass streaming algorithms. The formal proofs of Lemma 4.2 and Lemma 4.3 can be found in the full version.

Lemma 4.2 (From Perm-Hiding generators to Set-Hiding generators). *Let $n$ be a sufficiently large integer. Let $p, s \in \mathbb{N}$ such that $s \leq n^2$, and let $N = N^{\mathsf{RS}}(4n)$. Let $\varepsilon \in [0, 1)$ such that $\varepsilon \geq 1/n^{10}$. Suppose there is a generator $\mathcal{G}^{\mathsf{PH}}_{N,p-1}$ which always outputs $(N_{\mathcal{G}^{\mathsf{PH}}}, k_{\mathcal{G}^{\mathsf{PH}}}, \vec{\ell}_{\mathcal{G}^{\mathsf{PH}}})$ graphs and is $(\varepsilon/\log^2 n)$-*Perm-Hiding *against $(p-1)$-pass streaming algorithms with space $s$. Then there is a generator $\mathcal{G}^{\mathsf{SH}}_{n,p}$ such that: (1) it always outputs $(N_{\mathcal{G}^{\mathsf{SH}}}, k_{\mathcal{G}^{\mathsf{SH}}}, \vec{\ell}_{\mathcal{G}^{\mathsf{SH}}})$ graphs, where $N_{\mathcal{G}^{\mathsf{SH}}} = O(N_{\mathcal{G}^{\mathsf{PH}}} \cdot \log^2 n)$ and $k_{\mathcal{G}^{\mathsf{SH}}} = O(k_{\mathcal{G}^{\mathsf{PH}}} \cdot \log n)$; (2) it is $\varepsilon$-*Set-Hiding *against $p$-pass streaming algorithms with space $s$.*

Lemma 4.3 (From Set-Hiding generators to Perm-Hiding generators). *Let $n$ be a sufficiently large integer. Let $s \in \mathbb{N}$ and let $\varepsilon \in [0, 1)$. Suppose there is a generator $\mathcal{G}^{\mathsf{SH}}_{3n,p}$ which always outputs $(N_{\mathcal{G}^{\mathsf{SH}}}, k_{\mathcal{G}^{\mathsf{SH}}}, \vec{\ell}_{\mathcal{G}^{\mathsf{SH}}})$ graphs and is $(\varepsilon/\log^2 n)$-*Set-Hiding *against $p$-pass streaming algorithms with space $s$. Then there is a generator such that: (1) it always outputs $(N_{\mathcal{G}^{\mathsf{PH}}}, k_{\mathcal{G}^{\mathsf{PH}}}, \vec{\ell}_{\mathcal{G}^{\mathsf{PH}}})$ graphs where $N_{\mathcal{G}^{\mathsf{PH}}} = O(N_{\mathcal{G}^{\mathsf{SH}}} \cdot \log n)$ and $k_{\mathcal{G}^{\mathsf{PH}}} = O(k_{\mathcal{G}^{\mathsf{SH}}} \cdot \log n)$; (2) it is $\varepsilon$-*Perm-Hiding *against $p$-pass streaming algorithms with space $s$.*

Finally, Theorem 4.1 follows by applying Lemma 4.2 and Lemma 4.3 repeatedly.

Proof of Theorem 4.1. Let $N_0$ be a sufficiently large constant to be specified later. We will set $N_0$ so that Lemma 4.2 and Lemma 4.3 holds for all integers $n \geq N_0$. Let $c \geq 2$ be a sufficiently large constant to be specified later.

We will prove the theorem by induction on $p$. The theorem trivially holds when $p = 0$: since 0-pass streaming algorithm cannot read anything from the input, given an input subset $S$, one can simply output a bipartite graph of size $(n, n)$ such that the $i$-th vertex on the left side is connected to the $i$-th vertex on the right side if and only if $i \in S$. Clearly, this output is a Set-Enc$_n(S)$ graph.

Now, suppose the theorem holds for $p-1$, we show it holds for $p$ as well. We fix an $n \geq N_0$ such that $p \leq c^{-1} \cdot \sqrt{\log n}$, and we will show how to construct the desired generator $\mathcal{G}^{\mathsf{SH}}_{n,p}$. Let $n_2 = N^{\mathsf{RS}}(4n)$ and $n_1 = 3n_2$. We proceed as follows:

(1) Since $n_1 \geq n \geq N_0$, it follows that $(p-1) \leq c^{-1} \cdot \sqrt{\log n_1}$. Hence, by the induction hypothesis, there is a generator $\mathcal{G}^{\mathsf{SH}}_{n_1,p-1}$ such that: (1) it always outputs $(N_{(1)}, k_{(1)}, \vec{\ell}_{(1)})$ graphs, where $N_{(1)} \leq c \cdot n_1^{1+c(p-1)/\sqrt{\log n_1}}$ and $k_{(1)} \leq c \cdot (c \log n_1)^{2(p-1)}$; (2) it is $\varepsilon_{(1)}$-Set-Hiding against $(p-1)$-pass streaming algorithms with space $n_1^2$, where $\varepsilon_{(1)} = 1/n_1$.

(2) Since $n_2 \geq N_0$ and $n_1 = 3n_2$, combining Lemma 4.3 with the generator $\mathcal{G}^{\mathsf{SH}}_{n_1,p-1}$, there is a generator $\mathcal{G}^{\mathsf{PH}}_{n_2,p-1}$ such that: (1) it always outputs $(N_{(2)}, k_{(2)}, \vec{\ell}_{(2)})$ graphs where $N_{(2)} \leq O(N_{(1)} \cdot \log n_2)$ and $k_{(2)} \leq O(k_{(1)} \cdot \log n_2)$; (2) it is $\varepsilon_{(2)}$-Perm-Hiding against $(p-1)$-pass streaming algorithms with space $n_1^2$, where $\varepsilon_{(2)} = \varepsilon_{(1)} \cdot (\log n_2)^2$.

(3) Since $n \geq N_0$ and $n_2 = N^{\mathsf{RS}}(4n)$, combining Lemma 4.2 with the generator $\mathcal{G}^{\mathsf{PH}}_{n_2,p-1}$, there is a generator $\mathcal{G}^{\mathsf{SH}}_{n,p}$ such that: (1) it always outputs $(N_{(3)}, k_{(3)}, \vec{\ell}_{(3)})$ graphs, where $N_{(3)} \leq O(N_{(2)} \cdot \log^2 n)$ and $k_{(3)} \leq O(k_{(2)} \cdot \log n)$; (2) it is $\varepsilon_{(3)}$-Set-Hiding against $p$-pass streaming algorithms with space $n^2$, where $\varepsilon_{(3)} = \varepsilon_{(2)} \cdot (\log n)^2$.

Now we verify that the last generator $\mathcal{G}^{\mathsf{SH}}_{n,p}$ satisfies our requirements. Noting that $\log n_2 = O(\log n)$, it follows that

$$N_{(3)} \leq O\left(c \cdot \log^3 n \cdot n_1^{1+c(p-1)/\sqrt{\log n_1}}\right). \tag{1}$$

Setting $N_0$ to be sufficiently large, we have $n_1 = 3 \cdot N^{\mathsf{RS}}(4n) \leq n^{1+2c^{\mathsf{RS}}/\sqrt{\log n}}$, and hence

$$\log n_1 \leq \log n + 2c^{\mathsf{RS}} \cdot \sqrt{\log n}. \tag{2}$$

Taking log of both sides of Equation 1, it follows that

$$\log N_{(3)} \leq O(1) + 3\log\log n + \log n_1 + c(p-1) \cdot \sqrt{\log n_1}. \tag{3}$$

Setting $N_0$ to be sufficiently large and noting that $\sqrt{x + 2c^{\mathsf{RS}}\sqrt{x}} \leq \sqrt{x} + 2c^{\mathsf{RS}}$ for any $x > 0$, it follows from Equation 2 that

$$\sqrt{\log n_1} \leq \sqrt{\log n} + 2c^{\mathsf{RS}}. \tag{4}$$

Plugging Equation 2 and Equation 4 in Equation 3 and setting $c$ to be sufficiently large, we have

$$\begin{aligned}
&\log N_{(3)} && (5)\\
&\leq O(1) + 3\log\log n + \log n + 2c^{\mathsf{RS}} \cdot \sqrt{\log n} && (6)\\
&\quad + c(p-1) \cdot (\sqrt{\log n} + 2c^{\mathsf{RS}})\\
&\leq \log n + (3c^{\mathsf{RS}} + c(p-1)) \cdot \sqrt{\log n} + c(p-1) \cdot 2c^{\mathsf{RS}}\\
&\leq \log n + (5c^{\mathsf{RS}} + c(p-1)) \cdot \sqrt{\log n} && (cp \leq \sqrt{\log n})\\
&\leq \log n + cp \cdot \sqrt{\log n}. && (c \text{ is sufficiently large})
\end{aligned}$$

Noting that $\log n_2 = O(\log n)$ and setting $c$ to be sufficiently large, it follows that

$$\begin{aligned}
k_{(3)} &\leq O(\log^2 n \cdot k_{(1)})\\
&\leq (c/10\log^2 n) \cdot c \cdot (c \log n_1)^{2(p-1)}\\
&\leq (c/10\log^2 n) \cdot c \cdot (c \log n)^{2(p-1)} \cdot \left(1 + \frac{2c^{\mathsf{RS}}}{\sqrt{\log n}}\right)^{2(p-1)} && (\text{Equation 2})\\
&\leq c \cdot (c \log n)^{2p}. && (p \leq c^{-1}\sqrt{\log n} \text{ and } c \text{ is sufficiently large})
\end{aligned}$$

Finally, since $n_1 = 4N^{\mathsf{RS}}(4n) \geq n^{1+\Omega(1/\sqrt{\log n})}$, setting $N_0$ to be sufficiently large, we also have $\varepsilon_{(3)} = 1/n_1 \cdot (\log n_2)^2 \cdot (\log n)^2 \leq 1/n$. This completes the proof. □

In the full version, we also have the following different construction of Set-Hiding generators.

Remark 4.4. *For every $p(n) = o(\log n/\log\log n)$, for every sufficiently large integer $n$, there is a generator $\mathcal{G}^{\mathsf{SH}}_{n,p(n)}$ such that: (1) it always outputs $(N_{\mathcal{G}^{\mathsf{SH}}}, k_{\mathcal{G}^{\mathsf{SH}}}, \vec{\ell}_{\mathcal{G}^{\mathsf{SH}}})$ graphs, where $N_{\mathcal{G}^{\mathsf{SH}}} \leq n \cdot (\log n)^{O(p(n))}$; (2) it is $(n^{-1})$-Set-Hiding against $p$-pass streaming algorithms with space $n^{1+c_2^{\mathsf{RS}}/\log\log n}$.*

# 5 LOWER BOUNDS FOR MULTI-PASS STREAMING ALGORITHMS

In this section, we show that Theorem 4.1 implies our lower bounds for multi-pass streaming algorithms.

- In Section 5.1, we prove the lower bounds for $s$-$t$ reachability and $s$-$t$ undirected shortest-path.
- In Section 5.2, we prove our lower bounds for (approximate) bipartite perfect matching.
- In Section 5.3, we prove our lower bounds for estimating the rank of a matrix.
- In Section 5.4, we prove our lower bounds for linear programing in the row-streaming model.

## 5.1 $s$-$t$ Reachability and $s$-$t$ Undirected Shortest-Path

As already discussed in Section 2, Theorem 4.1 directly implies the following lower bounds for $s$-$t$ directed connectivity against multi-pass streaming algorithms.

Theorem 5.1 (Detailed version of Theorem 1.1 and Theorem 1.2). *The following statements hold.*

*(1) Given an $n$-vertex directed graph $G = (V, E)$ with two designated vertices $s, t \in V$, no randomized $o(\sqrt{\log n})$-pass streaming algorithm with space $n^{2-\varepsilon}$ for some constant $\varepsilon > 0$ can determine whether $s$ can reach $t$ in $G$ with probability at least $2/3$.*

*(2) Given an* undirected *graph $G = (V, E)$ and two designated vertices $s, t \in V$, no randomized $o(\sqrt{\log n})$-pass streaming algorithm with space $n^{2-\varepsilon}$ for some constant $\varepsilon > 0$ can output the length of the shortest $s$-$t$-path in $G$.*
*Moreover, for $p(n)$-pass streaming algorithms where $p(n) = o(\sqrt{\log n})$, the lower bound above for $s$-$t$ undirected shortest-path still holds if the algorithm is only required to compute an $(1 + \omega(\log n)^{-2p(n^2)})$-approximation to the length of the shortest path between $s$ and $t$.*

Proof. We first prove the theorem for $s$-$t$ reachability, and then show how to adapt the proof for $s$-$t$ undirected shortest-path.

*Lower bounds for $s$-$t$ reachability.* Suppose for the sake of contradiction that there is $p(n) \leq o(\sqrt{\log n})$ and a constant $\varepsilon > 0$ such that there is a $p(n)$-pass streaming algorithm $A_{\mathsf{stReach}}$ with $n^{2-\varepsilon}$ space, which solves $s$-$t$ reachability with probability at least

2/3. We further assume that $A_{\mathsf{stReach}}$ outputs 1 if it determines that $s$ can reach $t$, and 0 otherwise.

By Theorem 4.1 and noting that $p(n^2) \le o(\sqrt{\log n})$, there is $m(n) = n^{1+o(1)}$ such that for every sufficiently large $n$, there is a generator $\mathcal{G}^{\mathsf{SH}}_{n,p(n^2)}$ which always outputs $(m(n), k_{\mathcal{G}^{\mathsf{SH}}}, \vec{\ell}_{\mathcal{G}^{\mathsf{SH}}})$ graphs and is (1/10)-Set-Hiding for subsets of $[n]$ against $p(n^2)$-pass streaming algorithms with space $n^2$. For a layered graph $G$ in the support of $\mathcal{G}^{\mathsf{SH}}_{n,p}(\emptyset)$ or $\mathcal{G}^{\mathsf{SH}}_{n,p}(\{1\})$, we set $s = \mathsf{First}(G)_{[1]}$ and $t = \mathsf{Last}(G)_{[1]}$ ($s$ and $t$ do not depend on the choice of graph $G$).

Since $A_{\mathsf{stReach}}$ solves $s$-$t$ reachability with probability at least 2/3, it follows that

$$\Pr_{G \leftarrow \mathcal{G}^{\mathsf{SH}}_{n,p}(\{1\})}[A_{\mathsf{stReach}}(G) = 1] \ge 2/3,$$

$$\Pr_{G \leftarrow \mathcal{G}^{\mathsf{SH}}_{n,p}(\emptyset)}[A_{\mathsf{stReach}}(G) = 0] \ge 2/3.$$

The above means that $\|A_{\mathsf{stReach}}(\mathcal{G}^{\mathsf{SH}}_{n,p}(\emptyset)) - A_{\mathsf{stReach}}(\mathcal{G}^{\mathsf{SH}}_{n,p}(\{1\}))\|_{\mathrm{TV}} \ge 1/3$. This contradicts the fact that $\mathcal{G}^{\mathsf{SH}}_{n,p(n^2)}$ is (1/10)-Set-Hiding against $p(n^2)$-pass algorithms with space $n^2$, since $A_{\mathsf{stReach}}$ takes $p(m) \le p(n^2)$ passes and $m^{2-\varepsilon} \le n^2$ space.

*Lower bounds for $s$-$t$ undirected shortest-path.* We will use the same reduction in [9, Theorem 6]. Again suppose for the sake of contradiction that there is a $p(n)$-pass streaming algorithm $A_{\mathsf{stUpath}}$ with $n^{2-\varepsilon}$ space, which solves $s$-$t$ undirected shortest-path with probability at least 2/3.

Recall that $\mathcal{G}^{\mathsf{SH}}_{n,p(n^2)}$ always outputs graphs with exactly $k_{\mathcal{G}^{\mathsf{SH}}}$ layers. Let $\overline{G}$ be the undirected version of the layered graph $G$ in the support of $\mathcal{G}^{\mathsf{SH}}_{n,p}(\emptyset)$ or $\mathcal{G}^{\mathsf{SH}}_{n,p}(\{1\})$ (that is, $\overline{G}$ is obtained by removing the directions on all edges of $G$), we claim that $s$ can reach $t$ in $G$ if and only if the shortest path between $s$ and $t$ in $\overline{G}$ has length exactly $k_{\mathcal{G}^{\mathsf{SH}}} - 1$.

To see the claim above, note that (1) the shortest path between $s$ and $t$ in $\overline{G}$ has length at least $k_{\mathcal{G}^{\mathsf{SH}}} - 1$, since there are $k_{\mathcal{G}^{\mathsf{SH}}}$ layers in $\overline{G}$; (2) if $s$ can reach $t$ in $G$, then the same path gives us a $(k_{\mathcal{G}^{\mathsf{SH}}} - 1)$-length path from $s$ to $t$ in $\overline{G}$, and vice versa. Therefore, $A_{\mathsf{stUpath}}$ can be similarly used to distinguish the distributions $\mathcal{G}^{\mathsf{SH}}_{n,p}(\emptyset)$ and $\mathcal{G}^{\mathsf{SH}}_{n,p}(\{1\})$, a similar argument as in the case of $s$-$t$ reachability gives us the desired lower bound for $s$-$t$ undirected shortest-path.

Finally, $A_{\mathsf{stUpath}}$ is in fact only required to distinguish between (1) the shortest path between $s$ and $t$ in $\overline{G}$ has length exactly $k_{\mathcal{G}^{\mathsf{SH}}} - 1$ and (2) the shortest path between $s$ and $t$ in $\overline{G}$ has length at least $k_{\mathcal{G}^{\mathsf{SH}}}$. By Theorem 4.1 it holds that $k_{\mathcal{G}^{\mathsf{SH}}} \le O(\log n)^{2p(n^2)}$. Hence, it suffices for $A_{\mathsf{stUpath}}$ to compute a $(1 + (k_{\mathcal{G}^{\mathsf{SH}}})^{-1})$ approximation to the shortest path between $s$ and $t$ in $G$, and the theorem is proved by noting $\omega(\log m(n))^{2p(n^2)} \ge k^{\mathsf{SH}}_{\mathcal{G}}$ (recall that $G$ has $m(n)$ vertices). □

Remark 5.2. *If we apply Remark 4.4 instead of Theorem 4.1 in the proof of Theorem 5.1, then it follows that $s$-$t$ reachability or $s$-$t$ undirected shortest-path cannot be solved by $o(\log n/(\log\log n)^2)$-pass streaming algorithms with $n^{1+o(1/\log\log n)}$ space.*

Proof sketch. We will just sketch the proof for $s$-$t$ reachability here. The proof of $s$-$t$ undirected shortest-path is identical. Suppose for the sake of contradiction that there is $p(n) \le o(\log n/(\log\log n)^2)$ and $s(n) = n^{1+o(1/\log\log n)}$ such that there is a $p(n)$-pass streaming algorithm $A_{\mathsf{stReach}}$ with $s(n)$ space, which solves $s$-$t$ reachability with probability at least 2/3.

By Remark 4.4 and noting $p(n^2) \le o(\log n/\log\log n)$, there is $m(n) = n^{1+o(1/\log\log n)}$ such that for every sufficiently large $n$, there is a generator $\mathcal{G}^{\mathsf{SH}}_{n,p(n^2)}$ which always outputs $m(n)$-vertex graphs and is (1/10)-Set-Hiding against $p(n^2)$-pass streaming algorithms with space $n^{1+c_2^{\mathsf{RS}}/\log\log n}$. Noting that $p(m(n)) \le p(n^2)$ and $s(m(n)) \le n^{1+c_2^{\mathsf{RS}}/\log\log n}$ and applying the same argument as in Theorem 5.1, we can use $A_{\mathsf{stReach}}$ to break the generator $\mathcal{G}^{\mathsf{SH}}_{n,p(n^2)}$, which finishes the proof. □

## 5.2 Bipartite Perfect Matching

The goal of this section is to prove Theorem 5.3, which has two parts : the exact case and the approximate case.

Theorem 5.3 (Detailed version of Theorem 1.3). *No $o(\sqrt{\log n})$-pass streaming algorithm with $n^{2-\varepsilon}$ space for some $\varepsilon > 0$ can determine whether a bipartite graph $G = (L \sqcup R, E)$ with $|L| = |R| = n$ has a perfect matching with probability at least 2/3.*

*Moreover, for $p(n)$-pass streaming algorithms where $p(n) = o(\sqrt{\log n})$, the lower bound above still holds if the algorithm is only required to distinguish with probability at least 2/3 between (1) $G$ has a perfect matching of size $n$ and (2) $G$ has no matching of size at least $n \cdot (1 - \delta(n))$, for some $\delta(n) = 2^{-cp(n^2)/\sqrt{\log n}}$, where $c > 1$ is an absolute constant.*

Proof. We will adapt a folklore reduction from reachability to perfect matching, which is also used in [9, Theorem 5].

Suppose for the sake of contradiction that there is $p(n) \le o(\sqrt{\log n})$ and a constant $\varepsilon > 0$ such that there is a $p(n)$-pass streaming algorithm $A_{\mathsf{matching}}$ with $n^{2-\varepsilon}$ space, which determine whether a bipartite graph has perfect matching or not with probability at least 2/3. By Theorem 4.1 and noting that $p(n^2) \le o(\sqrt{\log n})$, there is $m(n) = n^{1+o(1)}$ and such that for every sufficiently large $n$, there is a generator $\mathcal{G}^{\mathsf{SH}}_{n,p(n^2)}$ which always outputs $(m(n), k_{\mathcal{G}^{\mathsf{SH}}}, \vec{\ell}_{\mathcal{G}^{\mathsf{SH}}})$ graphs and is (1/10)-Set-Hiding for subsets of $[n]$ against $p(n^2)$-pass streaming algorithms with space $n^2$.

For a layered graph $G$ in the support of $\mathcal{G}^{\mathsf{SH}}_{n,p(n^2)}(\emptyset)$ or $\mathcal{G}^{\mathsf{SH}}_{n,p(n^2)}([n])$, let $V_{\mathsf{mid}} = \bigcup_{i=2}^{k_{\mathcal{G}^{\mathsf{SH}}}-1} V_i(G)$. That is, $V_{\mathsf{mid}}$ is the set of vertices in the middle layers of $G$. We will construct a bipartite graph $H = (L \cup R, E_H)$[9] as follows:

[9] $E_H$ here is a list of edges, which specify the order that the streaming algorithm read the graph.

**Bipartite Perfect Matching from Set-Hiding Generators**

(1) For every vertex $v \in V_{\mathsf{mid}}$, we add a vertex $v^\ell$ to $L$ and a vertex $v^r$ to $R$. For every vertex $s \in \mathsf{First}(G)$, we add a vertex $s^\ell$ to $L$. For every vertex $t \in \mathsf{Last}(G)$, we add a vertex $t^r$ to $R$.
(2) Next we enumerate all the (directed) edges $(u, v)$ in $G$ according to their ordering in $\vec{E}(G)$, with ties broken according the lexicographically order[a]: for each edge $(u, v) \in E(G)$, we add an edge $(u^\ell, v^r)$ to $E_H$. (Note that vertices in $\mathsf{First}(G)$ has no incoming edges, and vertices in $\mathsf{Last}(G)$ has no outgoing ones.) For every vertex $v \in V_{\mathsf{mid}}$, we also add an edge $(v^\ell, v^r)$ to $E_H$.

[a]That is, we first enumerate edges in $E_1(G)$ in lexicographical order, then edges in $E_2(G)$ and so on

From the construction above, one can verify easily that $|L| = |R| = |V_{\mathsf{mid}}| + n = m(n) - n$. Let $n_H = |L|$. The following claim is crucial for the proof.

Claim 5.4.

*(1) If $G$ is a $\mathsf{Set\text{-}Enc}_n([n])$ graph, then $H$ has a perfect matching of size $n_H$.*
*(2) if $G$ is a $\mathsf{Set\text{-}Enc}_n(\emptyset)$ graph, then the maximum matching in $G$ has at most $|V_{\mathsf{mid}}| = n_H - n$ edges.*

To see Item (1) of Claim 5.4, consider the matching $M = \{(v^\ell, v^r) : v \in V_{\mathsf{mid}}\}$. Note that $|M| = |V_{\mathsf{mid}}| = n_H - n$, and the only unmatched vertices are $s^\ell$ and $t^r$ for $s \in \mathsf{First}(G)$ and $t \in \mathsf{Last}(G)$. For every $s \in \mathsf{First}(G)$ and $t \in \mathsf{Last}(G)$, any *augmenting path* of this matching $M$ in $H$ between $s^\ell$ and $t^r$ corresponds to a directed path from $s$ to $t$ in $G$. If $G$ is a $\mathsf{Set\text{-}Enc}_n([n])$ graph, we can find $|\mathsf{First}(G)|$ disjoint augmenting paths, in which the $i$-th path is from $\mathsf{First}(G)_{[i]}$ to $\mathsf{Last}(G)_{[i]}$.[10] This means that $H$ has a matching of size $|M| + n = n_H$, which is a perfect matching.

For Item (2) of Claim 5.4, if $G$ is a $\mathsf{Set\text{-}Enc}_n(\emptyset)$ graph, then no augmenting path between the unmatched vertices $s^\ell$ and $t^r$ can be found, since for every $(i, j) \in [n] \times [n]$, $\mathsf{First}(G)_{[i]}$ cannot reach $\mathsf{First}(G)_{[j]}$. Hence, $M$ is a maximum matching of $H$.

Note that $H$ can be generated "on the fly" in the streaming setting. Hence, since $A_{\mathsf{matching}}$ takes $p(|L|) \le p(n^2)$ passes and $(|L|)^{2-\varepsilon} \le m(n)^{2-\varepsilon} \le n^2$ space. By Claim 5.4, $A_{\mathsf{matching}}$ can be used to distinguish between the distributions $\mathcal{G}^{\mathsf{SH}}_{n,p(n^2)}(\emptyset)$ and $\mathcal{G}^{\mathsf{SH}}_{n,p(n^2)}([n])$, contradicts the security of the generator. This proves the first part of the theorem.

Finally, note that $A_{\mathsf{matching}}$ is indeed only required to distinguish between (1) $H$ has perfect matching of size $|L| = m(n) - n$ and (2) $H$ has no matching of size greater than $|V_{\mathsf{mid}}| = m(n) - 2n$. By Theorem 4.1, we have $m(n) \le c \cdot n^{1+cp(n^2)/\sqrt{\log n}}$ for some constant $c > 1$, the second part of the theorem then follows. □

[10]By the definition of a $\mathsf{Set\text{-}Enc}_n([n])$ graph, for each $i \in [n]$, there exists a directed path $P_i$ from $\mathsf{First}(G)_{[i]}$ to $\mathsf{Last}(G)_{[i]}$ in $G$. We further observe that these $n$ paths are vertex-disjoint. As otherwise, if $P_i$ share a vertex $v$ with $P_j$ for $i \neq j$, then it means $\mathsf{First}([G])_{[i]}$ can first reach $v$ can then reach $\mathsf{Last}(G)_{[j]}$, which contradicts the definition of $\mathsf{Set\text{-}Enc}_n([n])$ graphs.

Remark 5.5. *Consider the edge list $E_H$ constructed in the proof of Theorem 5.3, for every left vertex $u^\ell \in L$, its adjacent edges $(u^\ell, v^r)$ are listed consecutively in $E_H$. (Except for the edge $(u^\ell, v^\ell)$, which is an auxiliary edge that does not depend on the given graph $G$.)*

## 5.3 Matrix Rank

Estimating the rank of an $n \times n$ matrix is an interesting problem in the streaming. There has been several results studying this problem, we refer the readers to [7, 8, 14, 39]. The goal of this section is present a lower bound for the rank estimation problem.

The following corollary follows from Theorem 5.3 and the well-known reduction from computing the size of maximum matching for bipartite graphs to computing the rank of matrices (see, e.g., [43, Page 167] and [41]), we will consider the row streaming model in which the streaming algorithms get the rows of the matrix one by one in some arbitrary order.

Corollary 5.6 (Detailed version of Theorem 1.4). *No $o(\sqrt{\log n})$-pass streaming algorithm with $n^{2-\varepsilon}$ space for some $\varepsilon > 0$ can determine whether a given matrix $M \in \mathbb{F}_q^{n \times n}$ for some prime power $q = \omega(n)$ has full rank with probability at least 2/3.*

*Moreover, for $p(n)$-pass streaming algorithms where $p(n) = o(\sqrt{\log n})$, the lower bound above still holds if the algorithm is only required to distinguish with probability at least 2/3 between (1) $\mathsf{rank}(M) = n$ and (2) $\mathsf{rank}(M) \le n \cdot (1 - \delta(n))$, for some $\delta(n) = 2^{-cp(n^2)/\sqrt{\log n}}$, where $c > 1$ is an absolute constant.*

Proof. For a bipartite graph $G = (L \sqcup R, E)$ where $L = \{u_1, \ldots, u_n\}$ and $R = \{v_1, \ldots, v_n\}$, we consider the *Edmonds matrix* $M$ defined over the variables $\vec{x} = (x_{i,j})_{(i,j) \in [n] \times [n]}$:

$$M(i, j) := \begin{cases} x_{i,j} & \text{if } (u_i, v_j) \in E, \\ 0 & \text{otherwise.} \end{cases}$$

Let prime power $q = \omega(n)$ be a prime power. We know that $\mathsf{rank}(M)$ (rank of $M$ over the polynomial ring $Z[\vec{x}] = Z[x_{1,1}, \ldots, x_{n,n}]$) equals the size of the maximum matching in $H$. For a vector $\vec{r} = (r_{i,j})_{(i,j) \in [n] \times [n]} \in \mathbb{F}_q^{n \times n}$, we use $M(\vec{r})$ to denote the matrix over $\mathbb{F}_q$ obtained by substituting $x_{i,j}$ by $r_{i,j}$ for each $(i, j) \in [n] \times [n]$. Applying the Schwartz-Zippel lemma, it holds that if all the $r_{i,j}$ are i.i.d. uniform distributed over $\mathbb{F}_q$, then $\Pr_{\vec{r}}[\mathsf{rank}(M(\vec{r})) = \mathsf{rank}(M)] \ge 1 - o(1)$.

Therefore, computing the size of the maximum matching in $G$ can be reduced to computing the rank of $M(\vec{r})$ over $\mathbb{F}_q$. The proof is then finished by combing Theorem 5.3 and Remark 5.5. □

## 5.4 Linear Programming

Linear programming (LP) is a fundamental problem in optimization. The fastest LP solver for general dense matrix is dense to [37]. It takes $n^\omega$ time with $O(n^2)$ space, where $\omega \approx 2.37286$ is the exponent of current matrix multiplication [2]. For the situation where LP has roughly $n$ constraints/variables, it is not known how to extend the classical result [37] into streaming setting with $o(n)$ passes and $o(n^2)$ space. For matrix related problems such as linear

regression and low-rank approximation [13, 20, 21, 45, 48], there are two natural streaming models, one is row model and the other is entry model. Our linear programming focuses on the row model, which is discussed below.

*The row streaming model for LP..* Our lower bounds holds for the feasibility of LP in the row streaming model: in which the streaming algorithms get the constraints of the LP one by one in some arbitrary order, and is required to decide whether all the constraints can be simultaneously satisfied. Note that algorithms in the entry streaming model for LP also work in the row streaming model, hence our lower bounds hold for the entry streaming model as well.

Theorem 5.7. *No $o(\sqrt{\log n})$-pass algorithm with $n^{2-\varepsilon}$ space for some $\varepsilon > 0$ in the row streaming model can determine if a linear program of $n$ variables and $n$ constraints with coefficients in $\{0, 1\}$ is feasible or not.*

Proof. We will show a reduction from $s$-$t$ reachability over layered graphs to the feasibility to an LP problem. Consider a layered graph $G$ with $n$ vertices and $s$ and $t$ be two vertices in $G$. We construct the following linear program $P$ with $n$ variables:

**LP from $s$-$t$ Reachability**

(1) **Variables:** For each vertex $v$ of $G$, we add a variable $x_v$ to $P$. Note that as in the standard formulation of LP, all $x_v$ are non-negative.
(2) **Constraints:** For each vertex $v$ of $G$ such that $v \neq s$, we add a constraint
$$x_v \geq \sum_{u \,:\, \text{edge } (u,\, v) \,\in\, E(G))} x_u$$
to $P$
We also add two constraints $x_t \leq 0$ and $x_s \geq 1$ to $P$.
(3) **Constraints Ordering:** The constraints $x_t \leq 0$ and $x_s \geq 1$ come first. Note that each of the other constraints correspond to one vertex $v$ and all its incoming edges, which are all in the same edge-layer of $G$. We say this edge-layer is the corresponding edge-layer of that constraint. Then we list all the other constraints in the ordering of their corresponding edge-layers in $\vec{\ell}(G)$ (with ties broken arbitrarily).

Clearly, one can see that if $s$ can reach $t$ in $G$, then $x_s \geq 1$ implies $x_t \geq 1$ as well, and the LP instance $P$ is not feasible. On the other hand, if $s$ cannot reach $t$ in $G$, then one can construct an assignment to all variables $x$ so that for every vertex $v$ which is not reachable from $s$, $x_v$ is set to 0. Hence, $x_t = 0$ as well and $P$ is then feasible.

Finally, since the LP instance can be generated "on the fly", an algorithm deciding whether $P$ is feasible in the row streaming model also implies a streaming algorithm deciding whether $s$ can reach $t$ in $G$ in the layer-arrival model. This completes the proof. □

## ACKNOWLEDGMENTS

The authors would like to thank Sepehr Assadi for useful discussions.

Lijie Chen is supported by an IBM Fellowship. Zhao Song is supported in part by Ma Huateng Foundation, Schmidt Foundation, Simons Foundation, NSF, DARPA/SRC, Google and Amazon AWS.

## REFERENCES

[1] Miklós Ajtai, János Komlós, and Endre Szemerédi. 1983. An $O(n \log n)$ sorting network. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing (STOC)*. 1–9.
[2] Josh Alman and Virginia Vassilevska Williams. 2021. A Refined Laser Method and Faster Matrix Multiplication. In *SODA*. https://arxiv.org/pdf/2010.05846.pdf.
[3] Sepehr Assadi, Yu Chen, and Sanjeev Khanna. 2019. Polynomial pass lower bounds for graph streaming algorithms. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing (STOC)*. 265–276.
[4] Sepehr Assadi, Yu Chen, and Sanjeev Khanna. 2019. Sublinear algorithms for ($\Delta$ + 1) vertex coloring. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 767–786.
[5] Sepehr Assadi, Nikolai Karpov, and Qin Zhang. 2019. Distributed and Streaming Linear Programming in Low Dimensions. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS)*. ACM, 236–253.
[6] Sepehr Assadi, Sanjeev Khanna, and Yang Li. 2016. Tight bounds for single-pass streaming complexity of the set cover problem. In *48th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*. Association for Computing Machinery, 698–711.
[7] Sepehr Assadi, Sanjeev Khanna, and Yang Li. 2017. On estimating maximum matching size in graph streams. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 1723–1742.
[8] Sepehr Assadi, Gillat Kol, Raghuvansh R Saxena, and Huacheng Yu. 2020. Multi-Pass Graph Streaming Lower Bounds for Cycle Counting, MAX-CUT, Matching Size, and Other Problems. In *FOCS*. https://arxiv.org/pdf/2009.03038.pdf.
[9] Sepehr Assadi and Ran Raz. 2020. Near-Quadratic Lower Bounds for Two-Pass Graph Streaming Algorithms. In *FOCS*. https://arxiv.org/pdf/2009.01161.pdf.
[10] Greg Barnes and Jeff A Edmonds. 1998. Time–Space Lower Bounds for Directed st-Connectivity on Graph Automata Models. *SIAM J. Comput.* 27, 4 (1998), 1190–1202.
[11] Ruben Becker, Andreas Karrenbauer, Sebastian Krinninger, and Christoph Lenzen. 2017. Near-Optimal Approximate Shortest Paths and Transshipment in Distributed and Streaming Models. In *DISC*, Vol. 91. 7:1–7:16.
[12] Felix A. Behrend. 1946. On sets of integers which contain no three terms in arithmetical progression. *Proceedings of the National Academy of Sciences of the United States of America* 32, 12 (1946), 331.
[13] Christos Boutsidis, David P Woodruff, and Peilin Zhong. 2016. Optimal principal component analysis in distributed and streaming models. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing (STOC)*. 236–249.
[14] Marc Bury and Chris Schwiegelshohn. 2015. Sublinear estimation of weighted matchings in dynamic data streams. In *ESA*. 263–274.
[15] Amit Chakrabarti, Prantar Ghosh, Andrew McGregor, and Sofya Vorotnikova. 2020. Vertex ordering problems in directed graph streams. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 1786–1802.
[16] Amit Chakrabarti and Anthony Wirth. 2016. Incidence geometries and the pass complexity of semi-streaming set cover. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms (SODA)*. SIAM, 1365–1373.
[17] Timothy M. Chan and Eric Y. Chen. 2007. Multi-Pass Geometric Algorithms. *Discret. Comput. Geom.* 37, 1 (2007), 79–102.
[18] Yi-Jun Chang, Martin Farach-Colton, Tsan-Sheng Hsu, and Meng-Tsung Tsai. 2020. Streaming complexity of spanning tree computation, In STACS. *arXiv preprint arXiv:2001.07672*.
[19] Rajesh Chitnis, Graham Cormode, Mohammad Taghi Hajiaghayi, and Morteza Monemizadeh. 2014. Parameterized streaming: Maximal matching and vertex cover. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms (SODA)*. SIAM, 1234–1251.
[20] Kenneth L Clarkson and David P Woodruff. 2009. Numerical linear algebra in the streaming model. In *Proceedings of the forty-first annual ACM symposium on Theory of computing (STOC)*. 205–214.
[21] Kenneth L Clarkson and David P Woodruff. 2013. Low rank approximation and regression in input sparsity time. In *Proceedings of the forty-fifth annual ACM symposium on Theory of Computing (STOC)*. 81–90.
[22] Graham Cormode, Jacques Dark, and Christian Konrad. 2019. Independent Sets in Vertex-Arrival Streams. In *46th International Colloquium on Automata, Languages, and Programming (ICALP)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[23] Jeff Edmonds. 1993. Time-space trade-offs for undirected st-connectivity on a JAG. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*. 718–727.
[24] Jeff Edmonds, Chung Keung Poon, and Dimitris Achlioptas. 1999. Tight lower bounds for st-connectivity on the NNJAG model. *SIAM J. Comput.* 28, 6 (1999), 2257–2284.
[25] Yuval Emek and Adi Rosén. 2014. Semi-Streaming Set Cover. In *International Colloquium on Automata, Languages, and Programming (ICALP)*. Springer, 453–464.
[26] Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. 2004. On graph problems in a semi-streaming model. In *International Colloquium on Automata, Languages, and Programming (ICALP)*. Springer, 531–543.
[27] Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. 2009. Graph distances in the data-stream model. *SIAM J. Comput.* 38, 5 (2009), 1709–1727.
[28] Eldar Fischer, Eric Lehman, Ilan Newman, Sofya Raskhodnikova, Ronitt Rubinfeld, and Alex Samorodnitsky. 2002. Monotonicity testing over general poset domains. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing (STOC)*. 474–483.
[29] Buddhima Gamlath, Sagar Kale, Slobodan Mitrovic, and Ola Svensson. 2019. Weighted matchings via unweighted augmentations. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing (PODC)*. 491–500.
[30] Mohsen Ghaffari, Themis Gouleakis, Christian Konrad, Slobodan Mitrović, and Ronitt Rubinfeld. 2018. Improved massively parallel computation algorithms for mis, matching, and vertex cover. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing (PODC)*. 129–138.
[31] Ashish Goel, Michael Kapralov, and Sanjeev Khanna. 2012. On the communication and streaming complexity of maximum bipartite matching. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms (SODA)*. SIAM, 468–485.
[32] Mika Göös, Sajin Koroth, Ian Mertz, and Toniann Pitassi. 2020. Automating cutting planes is NP-hard. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*. ACM, 68–77.
[33] Mika Goos, Toniann Pitassi, and Thomas Watson. 2017. Query-to-Communication Lifting for BPP. In *FOCS*.
[34] Venkatesan Guruswami and Krzysztof Onak. 2016. Superlinear lower bounds for multipass graph processing. *Algorithmica* 76, 3 (2016), 654–683.
[35] Sariel Har-Peled, Piotr Indyk, Sepideh Mahabadi, and Ali Vakilian. 2016. Towards tight bounds for the streaming set cover problem. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS)*. 371–383.
[36] Monika Rauch Henzinger, Prabhakar Raghavan, and Sridhar Rajagopalan. 1998. Computing on data streams. *External memory algorithms* 50 (1998), 107–118.
[37] Shunhua Jiang, Zhao Song, Omri Weinstein, and Hengjie Zhang. 2021. Faster dynamic matrix inverse for faster lps. In *53rd Annual ACM Symposium on Theory of Computing (STOC)*. arXiv preprint arXiv:2004.07470.
[38] Michael Kapralov. 2013. Better bounds for matchings in the streaming model. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms (SODA)*. SIAM, 1679–1697.
[39] Yi Li and David P Woodruff. 2016. On approximating functions of the singular values in a stream. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing (STOC)*. 726–739.
[40] S Cliff Liu, Zhao Song, and Hengjie Zhang. 2020. Breaking the $n$-Pass Barrier: A Streaming Algorithm for Maximum Weight Bipartite Matching. *arXiv preprint arXiv:2009.06106* (2020).
[41] László Lovász and Michael D Plummer. 2009. *Matching theory*. Vol. 367. American Mathematical Soc.
[42] Andrew McGregor. 2005. Finding graph matchings in data streams. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*. Springer, 170–181.
[43] Rajeev Motwani and Prabhakar Raghavan. 1995. *Randomized Algorithms*. Cambridge University Press.
[44] Sagnik Mukhopadhyay and Danupon Nanongkai. 2020. Weighted min-cut: sequential, cut-query, and streaming algorithms. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*. 496–509.
[45] Jelani Nelson and Huy L Nguyên. 2013. OSNAP: Faster numerical linear algebra algorithms via sparser subspace embeddings. In *2013 ieee 54th annual symposium on foundations of computer science (FOCS)*. IEEE, 117–126.
[46] Aviad Rubinstein, Tselil Schramm, and Seth Matthew Weinberg. 2018. Computing exact minimum cuts without knowing the graph. In *9th Innovations in Theoretical Computer Science (ITCS)*. Schloss Dagstuhl-Leibniz-Zentrum fur Informatik GmbH, Dagstuhl Publishing, 39.
[47] Imre Z Ruzsa and Endre Szemerédi. 1978. Triple systems with no six points carrying three triangles. *Combinatorics (Keszthely, 1976), Coll. Math. Soc. J. Bolyai* 18 (1978), 939–945.
[48] Zhao Song, David P Woodruff, and Peilin Zhong. 2017. Low rank approximation with entrywise l1-norm error. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*. 688–701.
[49] Mariano Zelke. 2011. Intractability of min-and max-cut in streaming graphs. *Inform. Process. Lett.* 111, 3 (2011), 145–150.