Two Player Hidden Pointer Chasing and Multi-Pass Lower Bounds in Turnstile Streams

Anay Mehrotra¹, Vibhor Porwal¹, and Raghunath Tewari¹

¹Indian Institute of Technology, Kanpur

March 2, 2020

Abstract

Assadi, Chen, and Khanna [ACK19a] define a 4-player hidden-pointer-chasing (HPC⁴), and using it, give strong multi-pass lower bounds for graph problems in the streaming model of computation and a lower bound on the query complexity of sub-modular minimization. We present a two-player version (HPC²) of HPC⁴ that has matching communication complexity to HPC⁴. Our formulation allows us to lower bound its communication complexity with a simple direct-sum argument. Using this lower bound on the communication complexity of HPC², we retain the streaming and query complexity lower bounds by [ACK19a].

Further, by giving reductions from HPC², we prove new multi-pass space lower bounds for graph problems in turnstile streams. In particular, we show that any algorithm which computes the exact weight of the maximum weighted matching in an n-vertex graph requires $O(n^2)$ space unless it makes $\omega(\log n)$ passes over the turnstile stream, and that any algorithm which computes the minimum s-t distance in an n-vertex graph requires $n^{2-o(1)}$ space unless it makes $n^{\Omega(1)}$ passes over the turnstile stream. Our reductions can be modified to use HPC⁴ as well.
Contents

1 Introduction 3
  1.1 Our contributions ................................................. 4

2 Preliminaries and Other Related Work 5

3 Our Results 7
  3.1 Two player HPC (HPC\textsuperscript{2}k) ............................................. 7
  3.2 A lower bound for computing exact weight of MWM ................................. 10
  3.3 A lower bound for computing exact length of s-t shortest path ................. 11

4 Proofs 13
  4.1 Proof of Theorem 3.2 .................................................. 13
  4.2 Proof of Theorem 3.8 .................................................. 17
  4.3 Proof of Theorem 3.11 .................................................. 20

A Additional Preliminaries 28

B Four Player HPC (HPC\textsuperscript{4}k) 29

C Proofs of prior lower bounds by HPC\textsuperscript{4} using HPC\textsuperscript{2} 30
  C.1 Proof of Claim 3.7 .................................................. 30
  C.2 Lower bound for Max-Flow in the streaming model ............................... 30
  C.3 Lower bound on LMIS in the streaming model ...................................... 32
  C.4 Proof of Theorem 3.6 .................................................. 33
1 Introduction

Massive dynamic graphs are pervasive today and frequently arise in online social networks, internet topologies, and routing networks. Algorithms processing such massive graphs, are often limited by space in addition to time. This motivates the graph streaming model, where an algorithm computes a function of the graph with a small memory and processes the edges in a sequential order. In particular, this restricts the algorithm from having random access to the graph’s edges. Formally, in the insert-only streams the edges of a graph $G(V,E)$ are presented to the algorithm in an arbitrary sequential order. Under the semi-streaming restriction the algorithm makes one or few passes over the input, and uses $O(n \text{polylog } n)$ space for an $n$ vertex graph.

Turnstile streams, extend the insert-only streams to dynamic graphs by presenting the graph as a sequence of unit weight updates. Formally, a turnstile stream $\sigma := x_1x_2\ldots$, is a sequence of updates $x_i \in E \times \{+1, -1\}$, where $(e, 1)$ and $(e, -1)$ increment and decrement the weight of edge $e \in E$ by a unit respectively. We initialize all edge weights to 0, and define the weight of edge $e \in E$ at time $i \in \mathbb{N}$ as $w_i(e) := \sum_{k \in [i]} x_{k=(e,1)} - \sum_{k \in [i]} x_{k=(e,-1)}$. We assume that $w_i(e) \geq 0$ for all $i \in \mathbb{Z}_{\geq 0}$, and unless mentioned, that $\sigma$ has poly$(n)$ length.

Since graph problems were first studied in the streaming model [HRR98, FKM+05], they have attracted much attention, both in the dynamic and the insert-only settings [McG14, ACK19b, sub]. This led to efficient single-pass algorithms for several problems, including, estimating the weight of the minimum spanning tree of a dynamic graph [AGM12a], approximating the size of the maximum matching [Zel08, KKS14, FHM+20], and maintaining spectral sparsifiers [KNST19]. However, other graph problems, such as computing the global minimum cut or the edges of maximum matching [Zel11, Kap13], remain intractable with a single pass in the semi-streaming model even in insert-only streams. Here, allowing the algorithm to make multiple passes can lead to efficient solutions [McG05a, RSW18]. This contributes to the increasing popularity of multi-pass graph streaming algorithms [SGP11, AG13, Tir18, GKMS19].

Hidden Pointer Chasing. Assadi, Chen, and Khanna [ACK19a] introduce a new four player communication problem, Hidden Pointer Chasing (HPC$^4$), to derive space vs pass trade offs for graph streaming problems with poly$(n)$ number of passes. In contrast, most prior techniques were limited to proving lower bounds for algorithms making $O(\log(n))$ passes over the stream [GO13, CW16, BO17, AK18], and the few works which considered poly$(n)$ number of passes [FKM+08, HSSW12, Ass17] could only prove lower bounds scaling with the reciprocal of the number of passes $p$, that is, they could not capture more nuanced tradeoffs like $\Omega(n^2/p^5)$. HPC$^4$ is a variant of pointer chasing [PS82, NW91, DJS96, PRV99] where the pointers are “hidden” from the players. The four players are divided into two pairs $(P_A, P_B)$ and $(P_C, P_D)$, and each pair is given $n$ instances of the Set-Int problem (see Section 2). Given the current pointer, say $i \in [n]$ of one pair, this pair must solve the $i$-th Set-Int problem to find the next pointer for the other pair. The goal of HPC$^4_k$ is to start from a fixed 0-th pointer (of $P_A, P_B$), follow the pointers for a fixed number of steps, and output the $k$-th pointer. The players communicate over multiple rounds, where in odd (resp. even) rounds $(P_A, P_B)$ (resp. $(P_C, P_D)$) communicate with each other. A round ends when one pair sends one message to the other pair. They show that solving HPC$^4_k$ with a constant probability requires large amount of communication between the players.

Theorem 1.1 (Theorem 5, [ACK19a] - Informal). Any $k$-round protocol that finds the $k$-th pointer in HPC$^4_k$ with constant probability requires $\Omega(n^2/k^2 + n)$ bits of communication.

Using Theorem 1.1 they give multi-pass lower bounds for the following graph streaming problems.
Theorem 1.2 (Theorem 6 and 7, [ACK19a] – Informal). Any $p$-pass streaming algorithm for insert-only streams that with a constant probability finds the

- \text{min } s\text{-t cut value in a weighted directed/undirected graph requires } \Omega(n^2/p^5) \text{ space (Theorem 6).}
- \text{lexicographically-first MIS of an undirected graph requires } \Omega(n^2/p^5) \text{ space (Theorem 7).}

Can we design a HPC$_k$ like communication problem with fewer players that retains multi-pass pass lower bounds in Theorem 1.2? Further, can we use the hidden pointer chasing problem to prove multi-pass lower bounds for other graph streaming problems?

1.1 Our contributions

We make two major contributions in this paper, giving affirmative answers to both questions.

Two player version of HPC. Our main contribution is a two player version (HPC$^2$) of the hidden pointer chasing problem (Section 3.1), which has the same communication complexity as the four player version (HPC$^4$) (Theorem 3.2). In particular, we show that the communication complexity of HPC$^2$ is large when the players have only $(k - 1)$ rounds to find the $k$-th pointer, where as usual, in each round one player sends at most one message (of any length) to the other player. Unlike HPC$^4$ where there are $k$-rounds and one pair ($P_A, P_B$ or $P_C, P_D$) can send any number of messages in one round, our restrictions are necessary to make HPC$^2$ a “hard” communication problem. This in turn, retains the pass lower bounds in Theorem 1.2 (Section 3.1.1). Surprisingly, our communication model also allows us to use a simple direct-sum argument in our proofs, as opposed to a more involved one in [ACK19a]. This is our main technical contribution in designing a two player version of HPC$^4$.

Note that we use the lower bound on the communication complexity of $\epsilon$-solving Set-Int from [ACK19a] (Theorem 2.1) as a black-box in our proofs. Interestingly, we believe that restricting the number of messages exchanged per round in HPC$^4$ would lead to a similar simple direct-sum argument for proving the lower bound on its communication complexity.

Multi-pass lower bounds for graph problems in turnstile streams. Next, we show that any $p$-pass turnstile stream algorithm which finds the exact weight of the maximum weighted matching (MWM) with a constant probability in requires at least $\Omega(n^2/p^5)$-space if $p = O(\log n)$ (Theorem 3.8). Moreover, if we allow the turnstile stream to have exponential weight updates, that is, updates from the set $E \times [e^{O(n)}]$, then the same bound holds for all $p \geq 1$ (Corollary 3.9). Prior to our work, the best multi-pass lower bound for this problem was a $n^{1+\Omega(1/p)}/p^{O(1)}$ space lower bound for $p$-pass algorithms [GO13] (which also holds in insert-only streams). Theorem 3.8 significantly improves this lower bound in the turnstile stream. At the same time, it extends the range of passes for which we have a non-trivial lower bound in the semi-streaming setting from $O(\log n / \log \log n)$ to $O(\log n)$. In contrast to our lower bound, on the exact weight of MWM, an $(1 + \epsilon)$-approximation of the MWM (as opposed to just its weight) can be computed with $\tilde{O}(n\epsilon^{-4})$ space and $O(\epsilon^{-4} \log n)$ passes in insert-only streams [AG13,AG15].

As another illustration of Theorem 3.2, we show that any $p$-pass turnstile algorithm which finds the minimum $s$-$t$ distance with any constant probability requires at least $\Omega(n^2/p^5)$-space (Theorem 3.11). Notice, that unlike the lower bound in Theorem 3.8 this lower bound also holds for $p = \omega(\log n)$. This result significantly improves the prior best lower bound of $n^{1+\Omega(1/p)}/p^{O(1)}$ for $p$-pass turnstile algorithms algorithms [GO13] (which also holds in insert-only streams).
We prove these lower bounds by reductions from HPC$^2$. Our reductions can be modified to give reductions from HPC$^4$ as well. Thus Theorem 3.8 and Theorem 3.11 also make progress on Assadi, Chen, and Khanna [ACK19a]'s conjecture that hidden pointer chasing would lead to multi-pass lower bounds for other graph problems in the streaming setting.

Organization. The rest of the paper is organized as follows. We set up the required notation and discuss further related work in Section 2. We present a technical overview of our results in Section 3. We finalize the proof of the lower bound on the communication complexity of HPC$^2$ (Theorem 3.2) in Section 4.1, and our streaming lower bounds (Theorem 3.8 and 3.11) in Sections 4.2 and 4.3. For completeness, we give the description of HPC$^4$ and proofs of some preliminaries in Section A and B.

2 Preliminaries and Other Related Work

Notation. We use the capital ‘sans-serif’ font, for example, $A$, to denote random variables, and $d(\cdot)$ to denote their distribution. Let $\mathcal{U}$ be the uniform distribution on $[n]$. For a $n$-dimensional tuple, $A = (a_1, \ldots, a_n)$ and an index $i \in [n]$, define $A^{<i} := (a_1, \ldots, a_{i-1})$. We defer some preliminaries, which would be relevant later, to Section 3.1.1

Information theory and communication complexity. Given a random variable $X$, let its Shannon entropy be $H(X) := -\sum_x p(X = x) \log p(X = x)$ and its conditional entropy given a random variable $Y$ be $H(X|Y) := \sum_{y \in \mathcal{Y}} p(Y = y) H(X|Y = y)$. We define the mutual information of $X, Y$ as $\mathbb{I}(X; Y) := H(X) - H(X|Y)$. Given distributions $\mu, \nu: \Omega \rightarrow [0, 1]$ the total variation distance between them is $\Delta_{TV}(\mu, \nu) := \max_{E \subseteq \Omega} \mu(E) - \nu(E)$.

We use the standard two-way communication model of Yao [Yao79]. Consider a function $f: \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{Z}$, a protocol $\pi$, and two players $P_X$, $P_Y$. Let $P_X$ receive $X \in \mathcal{X}$ and $P_Y$ receive $Y \in \mathcal{Y}$. The communication cost, $CC_D(\pi)$, of $\pi$ on an input distribution $D$ is the maximum number of bits transmitted between the players to compute $f$ when $(X,Y) \sim D$. Define the communication complexity of $f$ as $CC(f) := \min_{\pi} \max_D CC_D(\pi)$ [KN97]. Let $\Pi$ be the transcript of $\pi$. Then, the internal information cost of $\pi$ on $D$ is $IC_D(\pi) := \mathbb{I}(\Pi; X|Y) + \mathbb{I}(\Pi; Y|X)$. Since conveying 1 bit of information requires at least 1 bit of communication, the communication cost of a protocol not smaller than its internal information cost, that is, $CC_D(\pi) \geq IC_D(\pi)$.

Set intersection. Set-Int is a two player problem, where the two players are given sets, $A \subseteq [n]$ and $B \subseteq [n]$ respectively, with the promise that there is a unique element $t$ at their intersection, that is, $\{t\} = A \cap B$. Their goal is to find this unique target element. We say $\pi \varepsilon$-solves Set-Int on distribution $D$ if it alters the distribution of the target element by at least $\varepsilon$ in total variation distance, that is, $\mathbb{E}_{\Pi \sim \Pi_\varepsilon} \left[ \Delta_{TV}(d(T|\Pi), d(T)) \right] \geq \varepsilon$, where original distribution of $T$ depends on $D$. We consider the following hard distribution, $D_{SI}$, for Set-Int.

**Hard distribution for Set-Int.** We sample $(A,B)$ from $D_{SI}$ as

- Uniformly sample $t \in [n]$ and add $t$ to both $A$ and $B$.

- For all $i \in [n]\backslash\{t\}$, sample $\mu_i$ iid from the uniform distribution over $\{(0,1), (1,0), (0,0)\}$, if $\mu_i = (0,1)$ add $i$ to $A$, else if $\mu_i = (1,0)$ add $i$ to $B$, otherwise do nothing.

Notice that the target $t$ has a uniform distribution, that is, $d(T) \sim \mathcal{U}$, if $(A,B) \sim D_{SI}$. We have the following lower bound for $\varepsilon$-solving the Set-Int problem on $D_{SI}$.
Dynamic graph streams. Ahn, Guha, and McGregor [AGM12a] introduce a related model for studying dynamic graphs in the streaming setting called dynamic graph streams. Under the natural assumption that we have simple graphs (without parallel edges or self loops), the dynamic graph stream a special of the turnstile stream we consider in Section 3.2. In the dynamic graph stream assumption that we have simple graphs (without parallel edges or self loops), the dynamic graph stream sequence of updates $x_k \in [n] \times [n] \times \mathbb{Z}_{\geq 0}$. Given an update $(u, v, w)$, we set $W_{uv}$ to $w$. Note that if $w = 0$, set $W_{uv}$ to 0, that is, we delete edge $e_{uv}$. We assume that $\sigma_d$ has poly$(n)$ length, and for all vertices $u \in [n]$, $W_{uu}$ is always 0.

Matching

Given a graph $G(V, E)$, a matching $M \subseteq E$ is a set of edges such that no two edges in $M$ share a common vertex. The maximum cardinality matching (MCM) is a matching with the maximum size. Let the weight of a matching be the sum of weights of its edges. A maximum weighted matching (MWM) is a matching with the maximum weight.

Approximating the matching edges.

Single pass. [FKM+05] give a 6-approximation algorithm to find MWM using $O(n \log n)$ space in the insert-only streams. [Zel08] improve this to a 5.585-approximation. Currently, the best approximation factor among single-pass algorithms is $(2 + \varepsilon)$ by a $O(n \log n \cdot \log n \cdot \log (1/\varepsilon))/\varepsilon$-space algorithm [PS17, GW19] improve this algorithm to use $O(n \log n \log (1/\varepsilon))/\varepsilon$ space. On the lower bound side, [Kap13] proved that any $(\varepsilon/(\varepsilon-1)$ or better approximation for MCM in insert-only streams requires at least $n^{1+\Omega(1/\log \log n)}$ space. Recently, [AKLY16] showed that $\tilde{\Theta}(n^2/\alpha^3)$ space is both necessary and sufficient for finding an $\alpha$-approximation of MCM in dynamic streams. Their lower bound is proved for linear sketches finding MCM in dynamic streams, combined with the results of [LNW14] and [AHLW16], the complexity of finding MCM in dynamic streams is resolved. [Kon15] independently prove similar, but slightly weaker, bounds.

Multi-pass. [McG05b] give a $(2 + \varepsilon)$-approximation algorithm for MWM with $O(\varepsilon^{-3})$ passes in insert-only streams using $O(n)$ space. [AG13] improve the approximation factor to $(1 - \varepsilon)^{-1}$ with $O(\varepsilon^{-4} \log n)$ passes and $O(\varepsilon^{-4} n)$ space.

Approximating matching size or weight.

Single pass. [KMNT20] give a $\log^2 n$-approximation algorithm with $O(\log^2 n)$ space for MCM size in random insert-only streams. [FHM+20] recently give a single-pass $6/11$-approximation algorithm for MCM size, their result improves upon the algorithm of [GKMS19] and is the current best known single-pass algorithm for approximating MCM size in random streams. [AKL17] give a $O(n^2/\alpha^4)$-space algorithm with approximation factor $\alpha \geq 2$ for the same problem, which also works for dynamic graph streams. In the same setting, they give an $\Omega(n/\alpha^2)$ space lower bound for computing MCM size on dense graphs, and a weaker $\Omega(\sqrt{n}/\alpha^{2.5})$ space lower bound, which holds for sparse and $O(\alpha)$ arboricity graphs.
Computing the exact matching size or weight. [CCE+16] give a single-pass \(\tilde{O}(k^2)\)-space algorithm for computing the MCM size in dynamic streams, assuming that this size is bounded by \(k\). On the lower bound side, [GO12] prove for any \(p = O\left(\frac{\log n}{\log \log n}\right)\), a \(p\)-pass algorithm deciding if a graph has a perfect matching in insert-only streams requires \(\Omega(n^{1+\Omega(1/p)}/p^{O(1)})\) space. This also implies the same lower bound for finding the MWM weight in insert-only and turnstile streams.

To the best of our knowledge, this is the only multi-pass lower bound for computing the exact MWM weight in the streaming setting. We significantly improve this lower bound for computing the weight of MWM in turnstile streams.

Estimating the length of shortest path

Single pass in insert-only stream. [FKM+05] construct an \(((1 + \varepsilon) \cdot \log n)\)-spanner of weighted and undirected graph with \(\tilde{O}(\log_{1 + \varepsilon}(rn))\) space, where \(r := \frac{u_{\text{max}}}{u_{\text{min}}}\), and satisfies \(\log r = \text{polylog } n\). [Bas08] construct a \((2k - 1)\)-spanner with an expected size of \(O(\min(m, kn^{1/k}))\) of an unweighted graph using \(O(kn^{1+1/k})\) space. On the lower bound side, [FKM+08] show that any 1-pass algorithm computing a \(k\) or better approximation of the distance between two vertices requires \(\Omega(n^{1+1/k})\) space. Unlike their result our lower bound holds for multi-pass algorithms.

Single pass in dynamic stream. Later, [KW14] give an algorithm for constructing \(n/d\)-additive spanners of unweighted graphs in dynamic streams which uses \(\tilde{O}(nd)\) space.

Multi-pass. The results of [BS07] imply a \(O(k)\)-pass and \(O(n^{1+1/k})\)-space algorithm for constructing \((2k - 1)\)-spanners of unweighted, undirected graphs in dynamic streams. [AGM12b] compute \((k\log_4(5) - 1)\)-spanners in the same setting with \(\log k\) passes over the stream and \(O(n^{1+1/k})\) space. [KW14] show that \(2^k\)-spanner of a weighted graph can be constructed in two passes and \(\tilde{O}(n^{1+1/k})\) space in dynamic streams. [HKN16] give a \(n^{o(1)}\)-pass and \(n^{1+o(1)}\)-space algorithm for weighted, undirected graphs with polynomially bounded edge weights which approximates all shortest path length from a given vertex \(s\) within a factor of \(1 + o(1)\) in insert-only streams.

[GO12] prove that for any \(p = O\left(\frac{\log n}{\log \log n}\right)\), a \(p\)-pass algorithm deciding if \(s, t \in V\) are at a distance of at most \(2(p + 1)\) in an undirected graph requires \(\Omega(n^{1+\Omega(1/p)}/p^{O(1)})\) space. This implies the same lower bound for computing the \(s\)-\(t\) shortest distance in the insert-only and turnstile stream.

To the best of our knowledge, this is the only multi-pass lower bound for computing the \(s\)-\(t\) shortest distance in the streaming setting. We significantly improve this lower bound for computing the \(s\)-\(t\) shortest distance in turnstile streams.

3 Our Results

3.1 Two player HPC (HPC\(^2\)_\(k\))

HPC\(^2\)_\(k\) is a two player communication problem between two players, \(P_X\) and \(P_Y\). Both players are given a universe \([n]\), and the 0-th pointer \(z_0 \in [n]\). Further, for each \(x \in [n]\), \(P_X\) and \(P_Y\) are given an instance, \((A_x, B_x)\), of Set-Int: \(P_X\) is given a set \(A_x \subseteq [n]\) and \(P_Y\) is given a set \(B_x \subseteq [n]\), with the promise that for all \(x \in [n]\), there is a unique element \(t_x \in [n]\) such that \(A_x \cap B_x = \{t_x\}\). \(P_X\) and \(P_Y\) communicate over \(k\) rounds in the following manner: In the even (resp. odd) rounds \(P_X\) (resp. \(P_Y\)) sends one message of arbitrary length to \(P_X\) (resp. \(P_Y\)). The goal of \(P_X\) and \(P_Y\) is to calculate the \((k + 1)\)-th pointer, \(z_{k+1}\), using the least amount of communication; where we define \(i\)-th pointer as follows.
Definition 3.1 (\(i\)-th pointer). Given \(i \in \mathbb{Z}_{\geq 0}\), define the \(i\)-th pointer, \(z_i \in [n]\), as \(t_{zi-1} \in [n]\) if \(i \geq 1\), that is, the next pointer of \(z_{i-1}\), and as \(z_0\) if \(i = 0\).

It is easy to see that given \((k + 1)\) rounds, \(\text{HPC}_k^2\) has a \(O(k \cdot n)\) bit communication protocol: In even rounds, \(r\), \(P_X\) sends \((A_{z_r}, z_r)\) to \(P_Y\) with \((n + \log n)\) bits, and in odd rounds, \(r\), \(P_Y\) sends \((B_{z_r}, z_r)\) to \(P_X\) with \((n + \log n)\) bits. However, if the players only have \(k\) rounds we show that solving \(\text{HPC}_k^2\) with any constant probability requires large communication.

Theorem 3.2 (A communication lower-bound for \(\text{HPC}_k^2\)). For any integer \(k \geq 1\), any \(k\)-round protocol that outputs the correct solution to \(\text{HPC}_k^2\) with a constant probability requires \(\Omega(n^2/k^2 + n)\) bits of communication.

Finding the \(k\)-th pointer \(z_{k+1}\) with any constant probability implies solving the \(z_k\)-th \(\text{Set-Int}\) problem with a constant probability. Then, since \(\varepsilon\)-solving \(\text{Set-Int}\) implies solving \(\text{Set-Int}\) with \(\varepsilon\) probability, the additive \(\Omega(n)\) term follows from Theorem 2.1. In the proof, we focus on deriving \(\Omega(n^2/k^2)\) which is the main term. Since, \(\Omega(n^2/k^2)\) is dominated by \(\Omega(n)\) when \(k = \Omega(\sqrt{n})\), we can assume that \(k = o(\sqrt{n})\) in the proof. We first prove the theorem for deterministic protocols \((\pi_{\text{HPC}^2})\) on a fixed hard distribution, \(D_{\text{HPC}^2}\), over the inputs (defined below). This lower bound extends to randomized protocols using the well-known Yao’s minimax principle [Yao83].

### Hard distribution for \(\text{HPC}^2\).

The distribution, \(D_{\text{HPC}^2}\), of input \(\{(A_i, B_i)\}_{i=1}^n\) to \(\text{HPC}^2\) is a product of distribution \(D_{\text{SI}}\), where for all \(i \in [n]\), \((A_i, B_i) \sim D_{\text{SI}}\).

For all \(0 \leq i \leq n\), let \(Z_i\) be the random variable corresponding to the \(i\)-th pointer \(z_i\). Define \(Z := (z_0, z_1, \ldots, z_k)\), then for all \(j \leq k\), \(Z^{<j} := (z_0, z_1, \ldots, z_{j-1})\) and \(Z^{\leq j} := (z_0, z_1, \ldots, z_j)\). For each round \(j \in [k]\), let \(\Pi_j\) be the transcript of \(\pi_{\text{HPC}^2}\) in that round, and let \(\Pi_j\) be the corresponding random variable. Define \(\Pi := \{\Pi_1, \Pi_2, \ldots, \Pi_k\}\), and \(E_j\) as the random variable \(E_j := (Z^{<j}, \Pi^{<j})\), which is a superset of the transcript available to the players at the start of the \(j\)-th round.

The following lemma is the main step in the proof of Theorem 3.2, which easily follows by choosing \(j = k\) in Lemma 3.3, and using the fact that the output of \(\pi_{\text{HPC}^2}\) is fixed conditioned on \(E_{k+1}\). This is true since \(E_{k+1}\) contains the entire transcript of \(\pi_{\text{HPC}^2}\). This argument is analogous to the one in [ACK19a] and we defer it to Section 4.1.3.

**Lemma 3.3** (\(d(Z_{j+1})\) is close to uniform in \(j\) rounds). For all \(0 \leq j \leq k\), we have

\[
\mathbb{E}_{(E_{j+1})} \left[ \Delta_{TV}(d(Z_{j+1} \mid E_{j+1}), \mathcal{U}) \right] \leq O\left( j \cdot \sqrt{\text{CC}_{D_{\text{HPC}^2}}(\pi_{\text{HPC}^2}) + j} \right).
\]

The above lemma says the distribution of \(Z_{j+1}\) is close to uniform unless \(\text{CC}_{D_{\text{HPC}^2}}(\pi_{\text{HPC}^2}) = \Omega(n^2/k^2)\). It is useful to think of the total variation distance as a measure of the “information” about \(Z_{j+1}\) known to the players given \(E_{j+1}\).

**Proof overview of Lemma 3.3** We first show that if the distribution of \(Z_j\) is uniform at the start of the \(j\)-th round, that is, \(d(Z_j) \sim \mathcal{U} \mid E_j\), then the distribution of \(Z_{j+1}\) remains “close” to uniform at the start of the \((j + 1)\)-th round, unless \(\pi_{\text{HPC}^2}\) communicates \(\Omega(n^2/k^2)\) bits. Lemma 3.4 formalizes this idea. Then the proof follows by an induction over \(j\). Here, we use the fact that \(\pi_{\text{HPC}^2}\) has only \(k\) rounds (one less) to find \(z_{k+1}\). We note that the claim holds for the base case, \(j = 0\), since

\[\text{Note that the value of } Z_0 \text{ (namely, } z_0) \text{ is known to the players. But, we still represent it has a random variable for ease of notation.}\]
\( \Delta_{TV}(d(Z_j|Z_0 = z_0), \ U) = 0 \). We maintain the invariant that for all \( 0 \leq \ell \leq j \) the distribution of \( Z_\ell \) is “close” to uniform at the start of the \( \ell \)-th round. Then, in the \( \ell \)-th inductive step, we apply Lemma 3.4 (which needs \( d(Z_\ell) \) to be exactly uniform) by approximating \( d(Z_\ell) \) by \( U \) and bounding the total error thus introduced over the first \( \ell \) steps.

**Lemma 3.4.** If for some \( 1 \leq j \leq k \), \( d(Z_j|E_j) \sim U \), then for any single message transcript \( \Pi_j \)

\[
\mathbb{E}_{(E_j, \Pi_j, Z_j \sim z_j)} \left[ \Delta_{TV}(d(Z_{j+1}|E_j, Z_j = z_j, \Pi_j), \ U) \right] = O\left( \sqrt{\frac{CC_{D_{\text{HPC}^2}}(\pi_{\text{HPC}^2}) + j}{n}} \right)
\]

A simple proof of Lemma 3.4 is our main technical contribution in this section. We present its overview here and defer the proof to Section 4.1.1.

Lemma 5.4 in [ACK19a] shows that all protocols \( \pi \), which communicate \( o(n^2/k^2) \) bits change the distribution of the target element of a randomly chosen instance of **Set-Int** in \( \text{HPC}^4 \) by a “small” amount from the uniform distribution, that is, \( \Delta_{TV}(d(T_x|\Pi), \ U) = o(1) \) if \( x \sim U \). This might appear to be similar to Lemma 3.4. However, one crucial difference between the two is that we give \( z_{j-1} \) to \( \pi_{\text{HPC}^2} \) and also allow it to use an additional communication round to find \( z_{j+1} \). Therefore, the above lemma does not follow by adapting Lemma 5.4 in [ACK19a] to \( \text{HPC}^2 \).

**Example 3.5.** To gain some intuition, let us allow \( \pi_{\text{HPC}^2} \) to make two passes to find \( z_{j+1} \) after we give it \( z_{j-1} \). Then, in the first round, \( \pi \) can find \( z_j \) exactly communicating \( O(n) \) bits, and further, find \( z_{j+1} \) exactly in the second round communicating \( O(n) \) more bits.

**Proof overview of Lemma 3.4.** In the proof, we show if the protocol only has one additional round it only makes a “small” change to the distribution of \( Z_{j+1} \) from uniform, unless it communicates a large number of bits. Let \( \varepsilon := \mathbb{E}_{(E_j, \Pi_j, Z_j \sim z_j)} \left[ \Delta_{TV}(d(Z_{j+1}|E_j, Z_j = z_j, \Pi_j), \ U) \right], \) and for all \( i \in [n] \), let \( \varepsilon_i := \mathbb{E}_{E_j} \mathbb{E}_{\Pi_j|Z_j=i,E_j} \left[ \Delta_{TV}(d(Z_{j+1}|E_j, Z_j = i, \Pi_j), \ U) \right] \). Since \( (Z_j \sim U|E_j) \), we can expand the expectation in \( \varepsilon \) to show that \( \varepsilon \leq j/n + \sum_{i \in [n]} \varepsilon_i/n \). Also, using the fact that the players can send at most one message in each round, and that if \( i \not\in Z^{<j} \) then \( (Z_{j+1} \perp Z^{<j}|Z_j = i) \), we can simplify the expectation in \( \varepsilon_i \) to show that \( \varepsilon_i = \mathbb{E}_{\Pi \leq j} \left[ \Delta_{TV}(d(T_i|\Pi \leq j), \ U) \right] \).

Note that this is exactly the definition of \( \varepsilon \)-solving the \( i \)-th **Set-Int** (see Section 2). From the known lower bounds for \( \rho \)-solving **Set-Int** it follows that, \( \Pi \perp\!
\!\!\perp (\Pi^{\leq j}; A_i \mid B_j) + \Pi(\Pi^{\leq j}; B_i \mid A_i) = \Omega(\varepsilon^2 \cdot n) \). Further, since the instances of **Set-Int** are independent of each other, using a simple direct-sum argument, we can lower bound \( IC_{D_{\text{HPC}^2}}(\pi_{\text{HPC}^2}) \) as \( IC_{D_{\text{HPC}^2}}(\pi_{\text{HPC}^2}) \geq \sum_{i \in [n]} \Pi(\Pi^{\leq j}; A_i \mid B_j) + \Pi(\Pi^{\leq j}; B_i \mid A_i) \). Finally, Lemma 3.4 follows by using the QM-AM inequality (cf. Chapter 4, [Cve12]) on \( \varepsilon \leq j/n + \sum_{i \in [n]} \varepsilon_i/n \) and using the fact that \( CC(\cdot) \geq IC(\cdot) \) for all protocols. This overview skips some details, for instance, the additional conditioning with \( Z^{<j+1} \). This conditioning results in the additive \( j \) term in \( \varepsilon \leq j/n + \sum_{i \in [n]} \varepsilon_i/n \) and the lemma. We consider these details in the proof.

### 3.1.1 Reproving prior lower bounds by \( \text{HPC}^4 \) using \( \text{HPC}^2 \)

**Notation.** Let \( (A, B) := \{(A_i, B_i)\}_{i=1}^n \), where for all \( i \in [n] \), \((A_i, B_i)\) is an instance of **Set-Int**, likewise, let \( (C, D) := \{(C_i, D_i)\}_{i=1}^n \), where for all \( i \in [n] \), \((C_i, D_i)\) is an instance of **Set-Int**. For completeness, we give a description of \( \text{HPC}^4_k \) in Section B.

**Lexicographically smallest MIS.** A set of vertices \( I \subseteq V \) in an undirected graph \( G(V, E) \) is said to be independent if for all vertices \( v, u \in I \), \((v, u) \notin E \). Given an undirected graph \( G(V, E) \), we define \( \text{LMIS}(G) \) to be the lexicographically maximal independent set of \( G \).
**Sub-modular function minimization.** A set-function $f : U \to [M]$ is said to be sub-modular if for all $A \subseteq B \subseteq U$ and element $a \in U \setminus B$, $f(A \cup \{a\}) - f(A) \geq f(B \cup \{a\}) - f(B)$. In the sub-modular minimization problem, we are given an evaluation oracle which gives a set $S \subseteq U$ returns $f(S)$, and the goal is to find $\arg\min_{S \subseteq U} f(S)$. Here, an algorithm is said to be $k$-adaptive if it makes no more than $k$-rounds of adaptive queries to an evaluation oracle, where all queries from the same round are done in parallel. [ACK19a] give the following lower bound for sub-modular minimization which with a constant probability outputs the min

**Theorem 3.6 (Theorem 8, [ACK19a]).** For all $k \geq 1$, any $k$-adaptive algorithm for sub-modular function minimization which with a constant probability outputs the $\min_{S \subseteq U} f(S)$ for $f : U \to [M]$, where $|U| = N$ and $M = O(N^{k+1})$ requires $\Omega\left(\frac{N^2}{k^2 \log M}\right)$ queries to the evaluation oracle.

In this section, we use HPC$^2$ to re-prove the lower bounds which [ACK19a] prove using HPC$^4$. The proofs largely remain the same, and so, we only present their overviews here and defer them to Section C.

We prove the lower bounds using reductions from HPC$^2_{k-1}$ to the specific streaming problem. We cover the lower bound on the query complexity of $k$-adaptive sub-modular algorithms as a special case at the end of this section. Our reduction from HPC$^2$ to a graph follows straightforwardly from the graphs in the reduction to HPC$^4$ by adapting them to HPC$^2_{k-1}$. In particular, given an input $(A, B, C, D)$ of HPC$^4$, let $G_4(A, B, C, D)$ be the graph in one of the reductions by [ACK19a]. Then, given an instance $(A, B)$ of HPC$^2_{k-1}$ we consider the graph $G_2(A, B) := G_4(A, B, A, B)$. Here, intuitively, the player $P_X$ receives the edges of $(P_A, P_C)$ and $P_Y$ receives the edges of $(P_B, P_D)$ in $G_2(A, B)$, while the graphs remains the same. Formally, we prove the following claim in Section C.1.

**Claim 3.7.** Given an instance $(A, B)$ of HPC$^2_{k-1}$, let $s_k$ be the solution of HPC$^4_k$ with input $(A, B, A, B)$, then $z_k = s_k$.

Before proceeding, we note that Claim 3.7 is not a reduction from HPC$^2_{k-1}$ to HPC$^4_1$. As unlike HPC$^2_{k-1}$, HPC$^4_k$ a pair of players, can exchange any number of messages in one round (see Section B) whereas in HPC$^2_{k-1}$ one player can only send at most one message in each round. Thus, a protocol for HPC$^4_k$ cannot solve HPC$^2_{k-1}$.

Finally, in the reductions by [ACK19a], solving the particular graph problem on $G_4(A, B, A, B) = G_2(A, B)$ gives us $s_k$, and thus by Claim 3.7, also $z_k$. This proves the correctness of the reduction. The rest of the proofs follow by well known arguments relating the space complexity of streaming algorithms and communication complexity.

Furthermore, the lower bound on the query complexity of a $k$-adaptive algorithm for sub-modular minimization follows from the fact that the $s$-$t$ cut function is sub-modular. In the proof [ACK19a] invoke their reduction for calculating the minimum weighted $s$-$t$ cut of a graph. They show that a sub-modular function query done by the streaming algorithm can be answered by the players using $O(\log M)$ bits. The same arguments also work in our reduction, for completeness, we provide the argument in Section C.4.

### 3.2 A lower bound for computing exact weight of MWM

Given a graph $G$, let $\text{opt}(G) \geq 0$ be the weight of the maximum weighted matching in $G$. We prove the following lower bound on computing opt($G$) in the turnstile streams.

**Theorem 3.8 (log $n$-pass lower bound for computing exact weight of MWM).** For all $p = O(\log n)$, any $p$-pass algorithm that finds the weight of MWM in a $n$-vertex graph $G$ with a constant probability in the turnstile stream requires $\Omega(n^2/p^5)$ space.
We give an overview of the proof of Theorem 3.8 below and defer the complete proof to Section 4.2.

Proof overview. The proof is by a reduction from $\text{HPC}_{k-1}^2$. We show how to convert an instance of the $\text{HPC}_{k-1}^2$ problem into a graph $G$, such that, $\text{opt}(G)$ determines $z_k$ in $\text{HPC}_{k-1}^2$. Then the rest of the proof follows by the well-known connection between the communication complexity of protocols and the space complexity of streaming algorithms.

Our construction is inspired by an idea used in [GO13] to prove a lower bound on deciding if a graph has a perfect matching. Namely, we construct $G$ such that it has a simple “almost” perfect matching where finding the best augmenting path is “hard”. On a high level, $G$ is a $2(k+1)+1$ layered graph with $n$ vertices in each layer, except the first one, which has $n-1$ vertices, and the last one, which has a single vertex $s \in V$. Without loss of generality assume that $z_0 = 1$, otherwise we can reorder the elements of $[n]$. Number the vertices in the first layer from 2 to $n$, and in all other layers (except the last) from 1 to $n$. We construct an almost perfect matching, $M \subseteq E$, by adding edges between vertices with the same number in an odd layer and the next even layer. So that the only unmatched vertices are the vertex 1 in the second layer and $s \in V$ in the last layer. Notice that these edges are input independent, and so, $M$ can be determined without any communication between the players. Next, we add additional input dependent edges between vertices an even layer and those in the next odd layer, such that, the edge connecting vertex $i$ to the solution $t_i$ of the $i$-th Set-Int, $(A_i, B_i)$, in the next layer has a larger weight than those connecting $i$ to other vertices in the next layer. These edges depend on the inputs of both the players. To construct the stream for $G$ without any communication between the players we have to use turnstile streams.

We can choose the weights of edges in $G$ such the path $P$, connecting pointers $\{z_0, \ldots, z_k\}$ and $s \in V$ is the optimal augmenting path for $M$, and extending $M$ with $P$ gives the unique MWM in $G$. Further, we choose weights of edges from the $2(k+1)$-th layer to $s \in V$ to encode the identities of the vertices in this layer. Finally, by setting all but these edge weights to be 0 mod $(n+1)$ we can identity $z_k$ as $\text{opt}(G) \text{ mod } (n+1)$ (Lemma 4.2). See Figure 1 for an illustration of $G$.  

The graph $G$ used in the proof has edge-weights as large as $\Theta(n \cdot 3^{2p})$. In order to construct this graph with a poly($n$) length turnstile stream, we require that $p$ is $O(\log n)$. This is the only place where we need this fact. Therefore, if we allow the turnstile stream to change the edge-weight by an exponential amount ($e^{O(n)}$) in one update, that is, turnstile updates are chosen from the set $E \times [e^{O(n)}]$, then we can avoid this restriction. Let this be the turnstile model with exponential weight updates.

**Corollary 3.9 (poly $n$-pass lower bound for computing exact weight of MWM).** For $p \geq 1$, any $p$-pass algorithm that finds the weight of MWM in a $n$-vertex graph $G$ with a constant probability in the turnstile stream with exponential weight updates requires $\Omega(n^2/p^5)$ space.

**Remark 3.10.** Theorem 3.8 holds for algorithms which only output $\text{opt}(G)$, for the graph $G$ defined by the complete stream, that is, the algorithms which do not maintain the $\text{opt}(G_i)$ for the graphs, $G_i$, defined by the first $i$ elements of the stream. In this sense our lower bounds are related to algorithms in the insert-only setting.

### 3.3 A lower bound for computing exact length of s-t shortest path

Given a graph $G$, let $\text{dist}(s, t) \geq 0$ be the length of the $s-t$ shortest path in $G$. We prove the following lower bound on computing $\text{dist}(s, t)$ in the turnstile streams.
Figure 1: Illustration of the graph used in the reduction from HPC\textsuperscript{2} to MWM problem with \( n = 5 \). The thicker edges in the figure mark all input independent edges, which also form the “almost” perfect matching, \( P \). The yellow path shows the optimal augmenting path \( M \), connecting the pointers \( \{ z_0, \ldots, z_3 \} \) and \( s \). For simplicity, we omit input augmenting edges with “non-pointer” left end point. The weight of all edges, apart from the weights in the figure, are multiples of \(( n + 1 )\).

**Theorem 3.11 (poly\( n \)-pass lower bound for computing exact length of shortest path).** For \( p \geq 1 \), any \( p \)-pass algorithm that finds the length of \( s-t \) shortest path in a \( n \)-vertex graph \( G \) with a constant probability in the turnstile stream requires \( \Omega( n^2 / p^5 ) \) space.

We give an overview of the proof of Theorem 3.11 below and defer the complete proof to Section 4.3.

**Proof overview.** The proof is by a reduction from HPC\textsuperscript{2}\( _{k-1} \). We show how to convert an instance of the HPC\textsuperscript{2}\( _{k-1} \) problem into a graph \( G \), such that, \( \text{dist}( s, t ) \) determines \( z_k \) in HPC\textsuperscript{2}\( _{k-1} \). The rest of the proof follows by the well-known connection between the communication complexity of protocols and the space complexity of streaming algorithms.

We construct \( G \) such that the shortest path in \( G \) passes through all the “pointer” vertices. \( G \) has \( k + 3 \) layers, each having \( n \) vertices, except the first and the last, which have single vertices \( s \) and \( t \) respectively. We connect \( s \) to the vertex corresponding to \( z_0 \) in the second layer, and connect all vertices in the second-last layer to \( t \). Additionally, we connect a vertex (except \( s \) and \( t \)) to all the vertices in the next layer. Let \( t_i \) be the solution of the \( i \)-th Set-Int, (\( A_i, B_i \)). We choose edge-weights such that, the edge connecting the \( i \)-th vertex \( v_i \), in one layer to the \( t_i \)-th in the next layer has smaller weight than an edge \( v_i \) to any other vertex in the next layer. This ensures that the \( s \) to \( t \) shortest path passes through \( z_0, z_1 := t_{ z_0 }, \ldots, z_{ k+1 } \). The weights of edges in \( G \) are dependent on the input of both players, and constructing the stream for \( G \) with no communication between the players requires the turnstile streams. In particular, \( G \) cannot be constructed without communication in dynamic streams.

We choose weights of edges from the \(( k + 2 )\)-th layer to \( t \in V \) to encode the identities of the vertices in this layer. Finally, by setting all but the edge-weights in the shortest path to be 0 mod \(( n + 1 )\) we can identity \( z_k \) as \( z_k \equiv \text{dist}( s, t ) \mod ( n + 1 ) \) (Lemma 4.6). See Figure 2 for an illustration of \( G \).

**Remark 3.12.** Note that unlike Theorem 3.8 the lower bound in Theorem 3.11 holds for algorithms making poly(\( n \)) passes over the stream. This is because, unlike Theorem 3.8 here, the graph constructed in the reduction has poly(\( n \)) edge weights for \( k = \text{poly}( n ) \).
4 Proofs

In this section, we finalize the proof of the lower bound on the communication complexity of HPC$^2$ (Theorem 3.2) presented in Section 3.1, and the proof of our streaming lower bounds (Theorem 3.8 and 3.11) presented in Sections 3.2 and 3.3.

4.1 Proof of Theorem 3.2

4.1.1 Proof of Lemma 3.4

We use the following claim in the proof of Lemma 3.4.

Claim 4.1. For any protocol $\pi$ for HPC$^2_k$ on $D_{HPC^2}$ with transcript $\Pi$, we have

$$IC_{D_{HPC^2}}(\pi) \geq \sum_{i \in [n]} I(\Pi; A_i \mid B_i) + I(\Pi; B_i \mid A_i).$$

Proof. Let $A := \{A_i\}_{i=1}^n$ and $B := \{B_i\}_{i=1}^n$. From the definition of internal information cost, we know $IC_{D_{HPC^2}}(\pi) := I(\Pi; A \mid B) + I(\Pi; B \mid A)$. In the rest of the proof, we bound the first term, the bound on the second term follows by symmetry.

Recall that in $D_{HPC^2}$, all events $\{(A_i, B_i)\}_{i \in [n]}$ are mutually independent. Using Propositions [A.1] it follows that

$$A_i \perp B_j \mid B_i.$$  \hspace{1cm} (2)

This holds because $A_i$ and $B_i$ are functions of $(A_i, B_i)$, and $B_j$ is function of $(A_j, B_j)$, where $(A_i, B_i)$ is sampled independently of $(A_j, B_j)$. Extending this argument to all $j \neq i$, we have that

$$\{A_j\}_{j \neq i} \perp A_i \mid \{B_j\}_{j \neq i}. \hspace{1cm} (3)$$
Now, we are ready to prove the claim. We have

$$I(\Pi; A | B) = I(\Pi; \{A_i\}_{i=1}^n | \{B_i\}_{i=1}^n)$$

$$\geq I(\Pi; A_1 | \{B_i\}_{i=1}^n) + I(\Pi; \{A_i\}_{i=2}^n | \{B_i\}_{i=1}^n, A_1)$$

Continuing this for \((n-1)\) more steps we get

$$I(\Pi; A | B) \geq \sum_{i \in [n]} I(\Pi; A_i | B_i).$$

Symmetrically we have

$$I(\Pi; B | A) \geq \sum_{i \in [n]} I(\Pi; B_i | A_i).$$

This proves the claim. \(\square\)

**Proof of Lemma 3.4** Fixing a value of \(Z_j = \ell \in [n]\), finding \(z_{j+1}\) is equivalent to solving the Set-Int defined by \((A_{\ell}, B_{\ell})\). The proof follows by a direct-sum argument over each value of \(Z_j\) and the known lower-bound on the communication complexity of \(\varepsilon\)-solving Set-Int. Define \(\varepsilon \in [0,1]\) as

$$\varepsilon := \mathbb{E}_{(E_j, I_j, Z_j=z_j)} \left[ \Delta_{TV}(d(Z_{j+1} | I_j, Z_j = z_j, E_j), U) \right].$$

Then we have,

$$\varepsilon = \mathbb{E}_{(I_j, Z_j=z_j)} \left[ \Delta_{TV}(d(Z_{j+1} | I_j, Z_j = z_j, E_j), U) \right]$$

$$\geq \mathbb{E}_{E_j} \left( \sum_{i \in [n]} \frac{1}{n} \cdot \mathbb{E}_{(I_j,E_j)} \left[ \Delta_{TV}(d(Z_{j+1} | I_j, Z_j = z_j, E_j), U) \right] \right)$$

$$\geq \mathbb{E}_{E_j} \left( \sum_{i \in [n] \setminus Z^{<j}} \frac{1}{n} \cdot \mathbb{E}_{(I_j,E_j)} \left[ \Delta_{TV}(d(Z_{j+1} | I_j, Z_j = i, E_j), U) \right] \right) + \frac{j}{n}.$$  \(\text{(4)}\)

Where Equation \(\text{(4)}\) holds since the total variation distance is bounded by 1, that is, \(\Delta_{TV}(\cdot, \cdot) \leq 1\), and that \(|Z^{<j}| \leq j\). If \(Z_j = i\) and \(i \not\in Z^{<j}\), then \(Z_{j+1}\) is a function of \((A_i, B_i)\) whereas \(Z^{<j}\) is not. Then, from Proposition A.1 we have \(Z_{j+1} \perp \! \! \! \perp Z^{<j} | Z_j = i, \text{if } i \not\in Z^{<j}\). Using this we get

$$\varepsilon \leq \mathbb{E}_{E_j} \left( \sum_{i \in [n] \setminus Z^{<j}} \frac{1}{n} \cdot \mathbb{E}_{(I_j,Z_j=i,E_j)} \left[ \Delta_{TV}(d(Z_{j+1} | I_j, Z_j = i), U) \right] \right) + \frac{j}{n}$$

$$\leq \mathbb{E}_{E_j} \left( \sum_{i \in [n]} \frac{1}{n} \cdot \mathbb{E}_{(I_j,E_j)} \left[ \Delta_{TV}(d(Z_{j+1} | I_j, Z_j = i), U) \right] \right) + \frac{j}{n}$$

$$\leq \sum_{i \in [n]} \frac{1}{n} \cdot \mathbb{E}_{(I_j,E_j)} \left[ \Delta_{TV}(d(Z_{j+1} | I_j, Z_j = i), U) \right] + \frac{j}{n}.$$  (Using \(\Delta_{TV}(\cdot, \cdot) \geq 0\))

Now, since \(I_j\) is a single message transcript and \(Z_j\) has a uniform distribution at the start of the \(j\)-th round (i.e., conditioned on \(E_j\) and before sending \(I_j\)), we have \(I_j \perp Z_j \mid E_j\). In other words, \(I_j\) is independent of the particular value of \(Z_j\). Using this we get

$$\varepsilon \leq \sum_{i \in [n]} \frac{1}{n} \cdot \mathbb{E}_{I_j \perp I_j} \left[ \Delta_{TV}(d(Z_{j+1} | I_j, Z_j = i), U) \right] + \frac{j}{n}$$

$$\leq \sum_{i \in [n]} \frac{1}{n} \cdot \mathbb{E}_{I_j \perp I_j} \left[ \Delta_{TV}(d(Z_{j+1} | I_j, Z_j = i), U) \right] + \frac{j}{n}$$

$$\leq \sum_{i \in [n]} \frac{1}{n} \cdot \mathbb{E}_{I_j \perp I_j} \left[ \Delta_{TV}(d(T_i \mid I_j), U) \right] + \frac{j}{n}.$$  \(\text{(5)}\)

14
For all $i \in [n]$, let $\varepsilon_i := \mathbb{E}_{\Pi \leq j} \left[ \Delta_{TV}(d(T_i \mid \Pi \leq j), U) \right]$. Then Equation (5) simplifies to
\[ \varepsilon \leq j/n + \sum_{i\in[n]} \varepsilon_i/n. \quad (6) \]

From the definition of $\varepsilon$-solving $\text{Set-Int}$ (see Section 2), it follows that $\pi_{\text{HPC2}} \varepsilon_i$-solves the $i$-th $\text{Set-Int}$ defined by $(A_i, B_i)$ in $j$ rounds. Using Theorem 2.1 (from [ACK19a]) we have the following lower bound for all $i \in [n]$
\[ I(\Pi \leq j ; A_i \mid B_i) + I(\Pi \leq j ; B_i \mid A_i) = \Omega(\varepsilon_i^2 \cdot n). \quad (7) \]

Recall that all $(A_i, B_i)$ for $i \in [n]$, are independent of each other. From Claim 4.1 we have
\[ \text{IC}_{\text{D}_{\text{HPC2}}} \left( \pi \right) \geq \sum_{i\in[n]} I(\Pi \mid A_i \mid B_i) + I(\Pi \mid B_i \mid A_i) \geq \sum_{i\in[n] \setminus Z \leq j} I(\Pi \leq j ; A_i \mid B_i) + I(\Pi \leq j ; B_i \mid A_i) \overset{(7)}{\geq} \sum_{i\in[n] \setminus Z \leq j} \Omega(\varepsilon_i^2 \cdot n). \quad (8) \]

Now, from the QM-AM inequality (cf. Chapter 4, [Cve12]) we have
\[ \sqrt{\sum_{i\in[n] \setminus Z \leq j} \varepsilon_i^2} \geq \frac{\sum_{i\in[n] \setminus Z \leq j} \varepsilon_i}{n-j}. \quad (9) \]

Using Equation (6), the fact that $\varepsilon_i \leq 1$, and that $\sum_{i\in[n]} \varepsilon_i = \sum_{i\in[n] \setminus Z \leq j} \varepsilon_i + \sum_{i\in Z \leq j} \varepsilon_i$, we get
\[ \sum_{i\in[n] \setminus Z \leq j} \varepsilon_i \geq n \cdot \varepsilon - (2j + 1). \quad (10) \]

Combining Equation (9) and (10) we get
\[ \sum_{i\in[n] \setminus Z \leq j} \varepsilon_i^2 \geq (n-j) \cdot \left( \frac{n}{n-j} \cdot \varepsilon - \frac{2j+1}{n-j} \right)^2. \quad (11) \]

Substituting this in Equation (8) we get
\[ \text{IC}_{\text{D}_{\text{HPC2}}} \left( \pi \right) \geq n(n-j) \cdot \Omega \left( \frac{n}{n-j} \cdot \varepsilon - \frac{2j+1}{n-j} \right)^2. \]

Finally, since $\text{CC}_{\text{D}_{\text{HPC2}}} (\pi_{\text{HPC2}}) \geq \text{IC}_{\text{D}_{\text{HPC2}}} (\pi_{\text{HPC2}})$, we get the required lower bound on $\varepsilon$
\[ \varepsilon = O \left( \frac{\sqrt{\text{CC}_{\text{D}_{\text{HPC2}}} (\pi_{\text{HPC2}})} \sqrt{n-j} + (2j+1)\sqrt{n}}{\sqrt{n}(n)} \right) \leq O \left( \frac{\sqrt{\text{CC}_{\text{D}_{\text{HPC2}}} (\pi_{\text{HPC2}}) + j}}{n} \right). \]
4.1.2 Proof of Lemma 3.3

Proof of Lemma 3.3. We use an induction argument over \( j \) in Lemma 3.4. The base case \((j = 0)\) is true since, \(\Delta_{\text{TV}}(d(Z_1 \mid Z_0 = z_0), U) = 0\). Assuming that Equation (1) holds for \((j − 1) < k\), that is,

\[
\mathbb{E}_E \left[ \Delta_{\text{TV}}(d(Z_j \mid E_j), U) \right] = O\left( (j − 1) \cdot \sqrt{\frac{\text{CC}_{\text{HPC2}}(\pi_{\text{HPC2}})}{n}} \right),
\]

we would show that it holds for \( j \). We have

\[
\mathbb{E}_{E_{j+1}} \left[ \Delta_{\text{TV}}(d(Z_{j+1} \mid E_{j+1}), U) \right]
= \mathbb{E}_E \left[ \sum_{i \in [n]} \mathbb{P}_{\pi \mid E_j} \left[ Z_j = i \mid E_j \right] \cdot \Delta_{\text{TV}}(d(Z_{j+1} \mid E_{j+1}), U) \right],
\]

and

\[
\mathbb{E}_{E_{j+1}} \Delta_{\text{TV}}(d(Z_{j+1} \mid E_{j+1}), U)
\leq \mathbb{E}_{E_j} \left[ \Delta_{\text{TV}}(d(Z_{j+1} \mid E_{j+1}), U) \right] + O\left( j \cdot \sqrt{\frac{\text{CC}_{\text{HPC2}}(\pi_{\text{HPC2}})}{n}} \right).
\]

This completes the induction argument and the proof of Lemma 3.3.

4.1.3 Proof of Theorem 3.2

Proof of Theorem 3.2 (assuming Lemma 3.3). First, we observe that solving \textit{Set-Int} with a constant probability implies \( \varepsilon \)-solving \textit{Set-Int} for some constant \( \varepsilon \). Then since, solving \textit{HPC}^2 with a constant probability requires solving the \( z(k_{+1}) \)-th \textit{Set-Int} with a constant probability we get an \( \Omega(n) \) communication lower bound on \textit{HPC}^2 from Theorem 2.1. This proves the theorem when \( k = \omega(\sqrt{n}) \).

In the rest of the proof we assume that \( k = o(\sqrt{n}) \). Consider any deterministic protocol \( \pi_{\text{HPC2}} \) for \textit{HPC}^2_k on \( \mathcal{D}_{\text{HPC2}} \). Then from Lemma 3.3 we have

\[
\mathbb{E}_{(E_{k+1})} \left[ \Delta_{\text{TV}}(d(Z_{k+1} \mid E_{k+1}), U) \right] \leq O\left( k \cdot \frac{\sqrt{\text{CC}_{\text{HPC2}}(\pi_{\text{HPC2}})}}{k} \right) \quad (\text{CC}_{\text{HPC2}}(\pi_{\text{HPC2}}) = o(n^2/k^2))
\leq o\left( \frac{n}{k} \right) + O\left( \frac{k}{n} \right),
\]

(Using \( k = o(\sqrt{n}) \), 13)
Recall that \((E_{k+1})\) contains the whole transcript, \(\Pi\), of \(\pi_{\text{HPC}2}\). Therefore, conditioned on \((E_{k+1})\) the output of \(\pi_{\text{HPC}2}\) is fixed. Let \(\text{OUTPUT}(E_{k+1})\) be this output. We have
\[
\Pr_{E_{k+1}}[\pi_{\text{HPC}2}\text{ is correct}] = \mathbb{E}_{E_{k+1}} \Pr_{(Z_{k+1}|E_{k+1})} [Z_{k+1} = \text{OUTPUT}(E_{k+1})]
\]
\[
\leq \mathbb{E}_{E_{k+1}} \left[ \Pr_{(Z_{k+1} \sim U)} [Z_{k+1} = \text{OUTPUT}(E_{k+1})] + \Delta_{TV}(d(Z_{k+1} | E_{k+1}), U) \right]
\]
(For all events \(\mathcal{E}, \Delta_{TV}(\mu, \eta) \geq |\Pr_\mu(\mathcal{E}) - \Pr_\eta(\mathcal{E})|\))
\[
\leq \frac{1}{n} + o(1).
\]
Therefore, \(\pi_{\text{HPC}2}\) cannot output the solution, \(Z_{k+1}\), of \(\text{HPC}_k^2\) on \(\mathcal{D}_{\text{HPC}2}\) with any constant probability. This proves the theorem for deterministic protocols. We can extend this lower bound to randomized protocols using Yao’s minimax principle [Yao83]. \(\square\)

### 4.2 Proof of Theorem 3.8

#### 4.2.1 Proof of Theorem 3.8 (Assuming Lemma 4.2)

**Proof of Theorem 3.8** The proof follows by a reduction from \(\text{HPC}_{k-1}^2\) on \(\mathcal{P}_{\text{HPC}2}\). For ease of notation, assume \(z_0 = 1\) without loss of generality. If \(z_0 \neq 1\), we can reorder the elements of \([n]\).

Given an instance of \(\text{HPC}_{k-1}^2\), \(\{(A_x, B_x)\}_{x \in [n]}\) over universe \([n]\), we turn it into a graph \(G(V, E, w)\) as follows (see Figure 3 for an example)

(a) Partition the vertices into \((2(k + 1) + 1)\) layers, \(V := \tilde{V}_0 \cup V_0 \cup \ldots \tilde{V}_k \cup V_k \cup \{s\}\), where \(\tilde{V}_0 := \{\tilde{v}(0,2), \ldots, \tilde{v}(0,n)\}\), and for all \(j \in [k]\), \(V_j := \{v(j,1), \ldots, v(j,n)\}\), and \(\tilde{V}_j := \{\tilde{v}(j,1), \ldots, \tilde{v}(j,n)\}\). Note that each layer has \(n\) vertices, except \(V_0\), which has \(n - 1\) vertices, and the last layer, which has a single vertex \(s\).

(b) Define weights \(w_j := (n + 1) \cdot 3^{2k-2j}\) and \(\tilde{w}_j := (n + 1) \cdot 3^{2k-2j+1}\) for all \(0 \leq j < k\), and \(\tilde{w}_k = 3 \cdot (n + 1)\). We have, \(w_j = 3 \cdot \tilde{w}_j + 1\), and \(\tilde{w}_j = 3 \cdot w_j\).

(c) \(E\) contains the following input-independent edges.

- (First layer) For all \(2 \leq i \leq n\), connect \(\tilde{v}(0,i)\) to \(v(0,i)\) with weight \(\tilde{w}_0\).
- (Last layer) For all \(i \in [n]\), connect \(v(k,i)\) to \(s\) with weight \(i\).
- (Other layers) For all \(j \in [k], i \in [n]\), connect \(\tilde{v}(j,i)\) to \(v(j,i)\) in \(V_j\) with weight \(\tilde{w}_j\).

(d) \(E\) contains the following input-dependent edges. For all \(j \in [k], i \in [n]\)

- (Other edges) For all \(\ell \in A_i \triangle B_i\), connect \(v(j-1,i)\) to \(\tilde{v}(j,\ell)\) with weight \(w_{j-1}\).
- (Target edge) For the unique \(\ell \in A_i \cap B_i\), connect \(v(j-1,i)\) to \(\tilde{v}(j,\ell)\) with weight \(2 \cdot w_{j-1}\).

This concludes the description of the graph \(G(V, E, w)\). See Figure 3 for an illustration. We will show later that the players can construct this graph with no communication.

Let \(M^*\) be any maximum weight matching in \(G\). By definition, it has weight \(\text{opt}(G)\). Let \(G^*\) be the graph constructed from the instance of \(\text{HPC}_{k-1}^2\) with same input \(\{(A_x, B_x)\}_{x \in [n]}\). Notice that \(G = G_k^*\) by definition. Let the vertices corresponding to pointer \(z_j\) be \(u_j \in V_j\) for \(0 \leq j \leq k\) and \(\tilde{u}_j \in \tilde{V}_j\) for \(j \in [k]\). Also, let \(u_0 = v_{(0,1)}\).

\[\text{We can give a similar reduction from } \text{HPC}_{k}^2 \text{ as well.}\]
Lemma 4.2. \( z_k \equiv \text{opt}(G) \pmod{(n+1)} \).

Now, we can use the well-known connection between communication protocols (communication complexity) and streaming algorithms (space complexity) to complete the proof.

Given an instance of \( \text{HPC}_k^2 \) with \( k = 2p + 1 \), players construct the graph \( G \) together as a turnstile stream. This does not require any communication between the players as we show below. Let \( N \) be the number of vertices in \( G \). The players create a turnstile stream \( \sigma \), in which the updates depending on \( P_B \)'s input appear first, the updates depending on \( P_A \)'s input appear next, and finally, the input independent updates appear at the end. Let \( \tilde{E} := \{(v_{(j-1,i)}, \tilde{v}_{(j,t,i)}) \mid j \in [k], \ i \in [n]\} \). An edge, \((v_{(j-1,i)}, \tilde{v}_{(j,t,i)}) \in \tilde{E} \) occurs \( 2w_j \) times in \( \sigma \), \( w_j \) times for \( P_A \)'s input and \( w_j \) times for \( P_B \)'s input, each time with a unit weight update. Weight updates for remaining edges are in either \( P_A \)'s part of \( \sigma \) or \( P_B \)'s part of \( \sigma \).

Let \( A \) be any algorithm making \( p \)-passes over \( \sigma \) for computing \( \text{opt}(G) \). From Lemma 4.2, we know that we can get pointer \( z_k \) if we know \( \text{opt}(G) \). Thus, we can get a protocol \( \pi_{\text{HPC}_2} \) for \( \text{HPC}_k^2 \) using \( A \). Each pass of \( A \) over \( \sigma \) is translated to at most two rounds of \( \pi_{\text{HPC}_2} \), and hence \( \pi_{\text{HPC}_2} \) has at most \( k - 1 \) rounds \( (k = 2p + 1) \). So, total communication during the run of protocol \( \pi_{\text{HPC}_2} \) is \( O(p \cdot S) \) where \( S \) is the space complexity of \( A \).

We know that \( \text{CC}(\pi_{\text{HPC}_2}) = \Omega(n^2/k^2) \) from Theorem 3.2. Since \( N = O(k \cdot n) \) and \( k = 2p + 1 \), we get \( \text{CC}(\pi_{\text{HPC}_2}) = \Omega(N^2/p^4) \). Using the fact that \( \text{CC}(\pi_{\text{HPC}_2}) = O(p \cdot S) \) we get \( S = \Omega(N^2/p^5) \). This completes the proof.

4.2.2 Proof of Lemma 4.2

First, we prove two auxiliary claims which help us in proving Lemma 4.2. Assume \( k \geq 1 \).
Claim 4.3. In $M^*$, exactly one vertex of layer $V_j$ is matched with a vertex of layer $\hat{V}_{j+1}$ for $0 \leq j \leq k - 1$ and, $n - 1$ vertices of layer $\hat{V}_j$ are matched with $n - 1$ vertices of the next layer $V_j$ for $0 \leq j \leq k$.

Proof. We will prove this by induction on the number of layers in the graph.

Base case. For graph $G^1$, the claim follows from the fact that $\hat{w}_0 > 2w_0$ and $w_0 > 2\hat{w}_1$, so for getting a maximum weight matching we try to match as many vertices as possible with a vertex in previous layer. It is possible to match all $n - 1$ vertices in layer $\hat{V}_0$, so that we can match just one vertex in layer $V_0$ with a vertex in the next layer $\hat{V}_1$, and in layer $\hat{V}_1$ we can match just $n - 1$ vertices to the vertices in next layer $V_1$. Thus, we see that the claim is true for $G^1$ and the base case follows.

Induction hypothesis. Now, assume that the claim is true for $G^{k-1}$.

Induction step. Consider graph $G = G^k$. We show the following.

- In $M^*$, all vertices in layer $\hat{V}_0$ should be matched. We will show this by contradiction. Assume there is some vertex, $\hat{v}_{(0,i)} \in \hat{V}_0$ which is unmatched. $\hat{v}_{(0,i)}$ is adjacent to $v_{(0,i)}$ so $v_{(0,i)}$ should be matched otherwise we can just match $\hat{v}_{(0,i)}$ with $v_{(0,i)}$ and the weight of $M^*$ would increase. Now, the maximum weight edge using which $v_{(0,i)}$ can be matched is $2w_0$, since $\hat{w}_0 > 2w_0$ we can increase the weight of $M^*$ by matching $\hat{v}_{(0,i)}$ with $v_{(0,i)}$ and removing the matched vertex with $v_{(0,i)}$ from $M^*$. Thus, it follows that $\hat{v}_{(0,i)}$ should be matched and this is a contradiction which implies that in a maximum weight matching of $G$ all vertices in layer $\hat{V}_0$ are matched.

- In $M^*$, $v_{(0,1)}$ can’t be unmatched. We will show this by contradiction. Assume $v_{(0,1)}$ is unmatched in $M^*$. Now, we know that there is at least one vertex adjacent to $v_{(0,1)}$, let any such vertex be $\hat{v}_{(1,i)}$. $\hat{v}_{(1,i)}$ must be matched in the next layer, if not we can just match $v_{(0,1)}$ with $\hat{v}_{(1,i)}$. Also, since $w_0 > \hat{w}_1$ we can increase the weight of $M^*$ by matching $v_{(0,1)}$ with $\hat{v}_{(1,i)}$ and removing the matched vertex with $\hat{v}_{(1,i)}$ from $M^*$. This is a contradiction which implies that $v_{(0,1)}$ should be matched in $M^*$.

From the above statements, it follows that among all vertices in layer $V_0$, only $v_{(0,1)}$ is matched with a vertex in layer $\hat{V}_1$. Now, remove layer $\hat{V}_0$, $V_0$ and the vertex with which $v_{(0,1)}$ is matched, call this vertex $\hat{v}_{(1,i)}$ from the graph, the remaining graph $\hat{G}$ is equivalent to $G^{k-1}$ up to renaming of vertices. By induction hypothesis, the claim is true for $\hat{G}$ and hence by principle of induction, the claim follows for $G$ as well.

Claim 4.4. In $M^*$, vertex $u_j$ is matched with $\hat{u}_{j+1}$ for $0 \leq j \leq k - 1$.

Proof. We will prove this by induction on the number of layers in the graph.

Base Case. In $G^1$, the weight of edge between $u_0$ and $\hat{u}_1$ is $2w_0$ which is greater than any other edge between $u_0$ and a vertex in layer $\hat{V}_1$. By Claim 4.3, it follows that $u_0$ is the only vertex in layer $V_0$ which is matched with a vertex in the next layer. Combining these two facts, we get that $u_0$ is matched with $\hat{u}_1$ in a maximum weight matching of $G^1$. Hence, the claim follows for $G^1$ which implies that the base case holds.

Induction hypothesis. Now, assume that the claim is true for $G^{k-1}$.

Induction step. Consider graph $G = G^k$. In the proof of Claim 4.3, we saw that in $M^*$, $v_{(0,1)}$ is matched with a vertex in layer $\hat{V}_1$. Now, we will show that $v_{(0,1)} = u_0$ is matched with $\hat{u}_1 \in \hat{V}_1$. Let $\hat{u}_1 = \hat{v}_{(1,i)}$. Let $W_1$ be the minimum possible weight of the maximum weight matching in which
\[ W_1 \geq 2 \cdot w_0 + \sum_{r=0}^{k} (n-1) \cdot \hat{w}_r + \sum_{r=1}^{k-1} 2 \cdot w_r + 1 \]

\[ W_2 \leq w_0 + \sum_{r=0}^{k} (n-1) \cdot \hat{w}_r + \sum_{r=1}^{k-1} 2 \cdot w_r + n. \]

Subtracting the above equations we get

\[ W_1 - W_2 \geq w_0 - (n-1) \]
\[ \geq 3^{2k} \cdot (n+1) - (n-1) \]
\[ > 3^2 \cdot (n+1) - (n-1) \]
\[ > 8 \cdot n + 10 \]
\[ > 0. \]

This shows that in \( M^* \), \( u_0 \) is matched with \( \hat{u}_1 \). Now, remove layer \( \hat{V}_0 \), \( V_0 \) and vertex \( \hat{v}_{(1,\ell)} \) from the graph, the remaining graph \( \hat{G} \) is equivalent to \( G^{k-1} \) up to renaming of vertices and hence by induction hypothesis, it follows that in a maximum weighted matching of \( \hat{G} \), vertex corresponding to pointer in layer \( \hat{V}_j \) will be matched with the vertex corresponding to pointer in layer \( \hat{V}_{j+1} \) for \( 0 \leq j \leq k-2 \). Also, by Claim \ref{claim:4.3} it follows that in \( M^* \), \( n-1 \) vertices of layer \( \hat{V}_j \) are matched with a vertex in next layer for \( 0 \leq j \leq k \). Thus, we get,

\[ \]
(c) $E$ contains the following input-independent edges.

- (First layer) Connect $s$ to $v_{(0,1)} \in V_0$ with weight $w_0$.
- (Last layer) For all $i \in [n]$, connect $v_{(k,i)} \in V_k$ to $t$ with weight $i$.
- (Other layers) For all $j \in [k], i \in [n]$, connect $v_{(j-1,i)} \in V_{j-1}$ to $v_{(j,i)} \in V_j$ for all $\ell \in [n]$ with weight $w_0$.

(d) $E$ contains the following input-dependent edges. For all $j \in [k], i \in [n]$

- (Other edges) For all $\ell \in A_i \triangle B_i$, decrease the weight of edge $(v_{(j-1,i)}, v_{(j,\ell)})$ by $w_1$.
- (Target edge) For the unique $\ell \in A_i \cap B_i$, decrease the weight of edge $(v_{(j-1,i)}, v_{(j,\ell)})$ by $2w_1$.

---

**Figure 4:** Illustration of the graph used in the reduction from $\text{HPC}^2_r$ to $s$-$t$ shortest path problem with $n = 5$. The edge connecting a “pointer” vertex to the next “pointer” vertex (yellow edge) has the smaller weight ($\Delta_2$) compared to the other (blue) edges, which have weight $\Delta_1$. So the path between $s$ and $t$ which passes through the yellow edges is the shortest. In particular, this path passes through all the “pointer” vertices, $\{z_0, \ldots, z_3\}$. The edges connecting vertices in layer $V_3$ to $t$ encode the identity of the corresponding vertex. For simplicity, we omit input dependent edges with “non-pointer” left end point. That is, we omit the input dependent edges with the left end point as $v_{(j,i)} \in V_j$ where $i \neq z_j$.

This concludes the description of the graph $G(V, E, w)$. We will show later that the players can construct this graph with no communication using turnstile streams.

Let $P^*$ be any shortest $s$-$t$ path in $G$. By definition, it has weight $\text{dist}(s, t)$. Let $G^r$ be the graph constructed from the instance of $\text{HPC}^2_{r-1}$ with same input $\{(A_x, B_x) \}_{x \in [n]}$. Notice that $G = G^k$ by definition. Let the vertices corresponding to pointer $z_j$ be $u_j \in V_j$ for $0 \leq j \leq k$.

It turns out that we can find the pointer $z_k$ given the $s$-$t$ shortest distance in $G$. Formally, we have the following lemma.

**Lemma 4.6.** $z_k \equiv \text{dist}(s,t) \pmod{(n+1)}$.

First, we show that the optimal $s$-$t$ path passes through the pointer vertices in the following claim and then prove Lemma 4.6.

**Claim 4.7.** $P^*$ passes through $\{u_0, u_1, \ldots, u_k\}$.  

21
Thus, starting from weight of the shortest path starting from vertices in \( G \) turnstile stream. This does not require any communication as we show now. Let Theorem 3.11. Induction step. Consider graph \( G^1 \). Since \( w(u_0, u_1) + n < w(u_0, v_{(1,\ell)}) + w(v_{(1,\ell)}, t) \) for all \( v_{(1,\ell)} \neq u_1 \), the shortest path in \( G^1 \) passes through \( u_1 \). Thus the base case follows.

Induction hypothesis. Now, assume that the claim is true for \( G^{k-1} \).

Base case. Consider graph \( G \). Let \( A^\dagger \) be any algorithm making \( 2^{\pi_k} \) passes over \( \sigma \)’s input, each time with a unit weight decrease. Weight updates necessarily pass through \( u_0 \) and by induction hypothesis a shortest path from \( u_1 \) to \( t \) passes through \( u_1, u_2, \ldots, u_k \). This implies \( P^* \) passes through \( \{u_0, u_1, \ldots, u_k\} \) and the claim follows.

Now, we prove Lemma 4.6.

Proof of Lemma 4.6. By definition of \( P^* \) its weight of \( \text{dist}(s, t) \). From Claim 4.7 we have

\[
\text{dist}(s, t) = w(s, u_0) + \sum_{j \in [k]} w(u_{j-1}, u_j) + w(u_k, t).
\]

Taking modulo \((n + 1)\), and using the fact that for all \( j \in [k] \), \( w(u_{j-1}, u_j) \equiv 0 \) mod \((n + 1)\) and \( w(s, u_0) \equiv 0 \) mod \((n + 1)\), we get

\[
\text{dist}(s, t) \equiv w(u_k, t) \mod (n + 1)
\equiv w(v_{(k, z_k)}, t) \mod (n + 1)
\equiv z_k \mod (n + 1).
\]

Now, we can use the well-known connection between communication protocols (communication complexity) and streaming algorithms (space complexity) to complete the remaining proof of Theorem 3.11.

Given an instance of \( \text{HPC}_{k-1}^2 \) with \( k = 2p + 1 \), the players construct the graph \( G \) together as a turnstile stream. This does not require any communication as we show now. Let \( N \) be the number of vertices in \( G \). The players create a turnstile stream \( \sigma \), in which the updates independent of the input appear first, then the updates depending on \( P_B \)’s input appear, and finally the updates depending on \( P_A \)’s input appear. Let \( \tilde{E} := \{(v_{(j-1,i)}, v_{(j,i)}) \mid j \in [k], i \in [n]\} \). An edge, \((v_{(j-1,i)}, v_{(j,i)}) \in \tilde{E} \) occurs \( 2w_i \) times in \( \sigma \) (in addition to occurring \( w_i \) times for input-independent part of \( \sigma \)), \( w_j \) times for \( P_A \)’s input and \( w_j \) times for \( P_B \)’s input, each time with a unit weight decrease. Weight updates for remaining edges are in either \( P_A \)’s part of \( \sigma \) or \( P_B \)’s part of \( \sigma \). Note that length of \( \sigma \) is \( \text{poly}(n) \).

Let \( A \) be any algorithm making \( p \)-passes over \( \sigma \) for computing \( \text{dist}(s, t) \). From Lemma 4.6 if we know \( \text{dist}(s, t) \), we can get pointer \( z_k \). Thus, we can get a protocol \( \pi_{\text{HPC}_2^2} \) for \( \text{HPC}_{k-1}^2 \) using \( A \). Each pass of \( A \) over \( \sigma \) is translated to at most two rounds of \( \pi_{\text{HPC}_2^2} \), and hence \( \pi_{\text{HPC}_2^2} \) has at most \( k - 1 \) rounds \((k = 2p + 1)\). So, total communication during the run of protocol \( \pi_{\text{HPC}_2^2} \) is \( O(p \cdot S) \) where \( S \) is the space complexity of \( A \).

We already know that \( \text{CC}(\pi_{\text{HPC}_2^2}) = \Omega(n^2/k^2) \) from Theorem 3.2. Since \( N = O(k \cdot n) \) and \( k = 2p + 1 \), we get \( \text{CC}(\pi_{\text{HPC}_2^2}) = \Omega(N^2/p^4) \). Finally, using the fact that \( \text{CC}(\pi_{\text{HPC}_2^2}) = O(p \cdot S) \) we get \( S = \Omega(N^2/p^5) \). This completes the proof.
References

[ACK19a] Sepehr Assadi, Yu Chen, and Sanjeev Khanna. Polynomial pass lower bounds for graph streaming algorithms. In Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019, pages 265–276, 2019.

[ACK19b] Sepehr Assadi, Yu Chen, and Sanjeev Khanna. Sublinear algorithms for $(\Delta + 1)$ vertex coloring. In Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019, pages 767–786, 2019.

[AG13] Kook Jin Ahn and Sudipto Guha. Linear programming in the semi-streaming model with application to the maximum matching problem. Inf. Comput., 222:59–79, 2013.

[AG15] Kook Jin Ahn and Sudipto Guha. Access to data and number of iterations: Dual primal algorithms for maximum matching under resource constraints. In Proceedings of the 27th ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2015, Portland, OR, USA, June 13-15, 2015, pages 202–211, 2015.

[AGM12a] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012, pages 459–467, 2012.

[AGM12b] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2012, Scottsdale, AZ, USA, May 20-24, 2012, pages 5–14, 2012.

[AHLW16] Yuqing Ai, Wei Hu, Yi Li, and David P. Woodruff. New characterizations in turnstile streams with applications. In 31st Conference on Computational Complexity, CCC 2016, May 29 to June 1, 2016, Tokyo, Japan, pages 20:1–20:22, 2016.

[AK18] Sepehr Assadi and Sanjeev Khanna. Tight bounds on the round complexity of the distributed maximum coverage problem. In Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018, pages 2412–2431, 2018.

[AKL17] Sepehr Assadi, Sanjeev Khanna, and Yang Li. On estimating maximum matching size in graph streams. In Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19, pages 1723–1742, 2017.

[AKLY16] Sepehr Assadi, Sanjeev Khanna, Yang Li, and Grigory Yaroslavtsev. Maximum matchings in dynamic graph streams and the simultaneous communication model. In Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016, pages 1345–1364, 2016.

[Ass17] Sepehr Assadi. Tight space-approximation tradeoff for the multi-pass streaming set cover problem. In Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium
on Principles of Database Systems, PODS 2017, Chicago, IL, USA, May 14-19, 2017, pages 321–335, 2017.

[Bas08] Surender Baswana. Streaming algorithm for graph spanners - single pass and constant processing time per edge. Inf. Process. Lett., 106(3):110–114, 2008.

[BO17] Mark Braverman and Rotem Oshman. A rounds vs. communication tradeoff for multi-party set disjointness. In 58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017, pages 144–155, 2017.

[BS07] Surender Baswana and Sandeep Sen. A simple and linear time randomized algorithm for computing sparse spanners in weighted graphs. Random Struct. Algorithms, 30(4):532–563, 2007.

[CCE+16] Rajesh Chitnis, Graham Cormode, Hossein Esfandiari, MohammadTaghi Hajiaghayi, Andrew McGregor, Morteza Monemizadeh, and Sofya Vorotnikova. Kernelization via sampling with applications to finding matchings and related problems in dynamic graph streams. In Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016, pages 1326–1344, 2016.

[CT12] T. M. Cover and J. A. Thomas. Elements of Information Theory, volume 2. John Wiley & Sons, November 2012.

[Cve12] Zdravko Cvetkovski. Inequalities: theorems, techniques and selected problems. Springer Science & Business Media, 2012.

[CW16] Amit Chakrabarti and Anthony Wirth. 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 2016, Arlington, VA, USA, January 10-12, 2016, pages 1365–1373, 2016.

[DJS96] Carsten Damm, Stasys Jukna, and Jirí Sgall. Some bounds on multiparty communication complexity of pointer jumping. In STACS 96, 13th Annual Symposium on Theoretical Aspects of Computer Science, Grenoble, France, February 22-24, 1996, Proceedings, pages 643–654, 1996.

[FHM+20] Alireza Farhadi, Mohammad Taghi Hajiaghayi, Tung Mai, Anup Rao, and Ryan A. Rossi. Approximate maximum matching in random streams. In Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020, pages 1773–1785, 2020.

[FKM+05] Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. Theor. Comput. Sci., 348(2-3):207–216, 2005.

[FKM+08] Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. Graph distances in the data-stream model. SIAM J. Comput., 38(5):1709–1727, 2008.

[GKMS19] Buddhima Gamlath, Sagar Kale, Slobodan Mitrovic, and Ola Svensson. Weighted matchings via unweighted augmentations. In Proceedings of the 2019 ACM Symposium
on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019, pages 491–500, 2019.

[GO13] Venkatesan Guruswami and Krzysztof Onak. Superlinear lower bounds for multipass graph processing. In Proceedings of the 28th Conference on Computational Complexity, CCC 2013, K.lo Alto, California, USA, 5-7 June, 2013, pages 287–298, 2013.

[GW19] Mohsen Ghaffari and David Wajc. Simplified and space-optimal semi-streaming (2+epsilon)-approximate matching. In Jeremy T. Fineman and Michael Mitzenmacher, editors, 2nd Symposium on Simplicity in Algorithms, SODA@SODA 2019, January 8-9, 2019 - San Diego, CA, USA, volume 69 of OASICS, pages 13:1–13:8. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[HKN16] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. A deterministic almost-tight distributed algorithm for approximating single-source shortest paths. In Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016, pages 489–498, 2016.

[HRR98] Monika Rauch Henzinger, Prabhakar Raghavan, and Sridhar Rajagopalan. Computing on data streams. In External Memory Algorithms, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, May 20-22, 1998, pages 107–118, 1998.

[HSSW12] Magnús M. Halldórsson, Xiaoming Sun, Mario Szegedy, and Chengu Wang. Streaming and communication complexity of clique approximation. In Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part I, pages 449–460, 2012.

[Kap13] Michael Kapralov. Better bounds for matchings in the streaming model. In Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013, pages 1679–1697, 2013.

[KKS14] Michael Kapralov, Sanjeev Khanna, and Madhu Sudan. Approximating matching size from random streams. In Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014, pages 734–751, 2014.

[KMNT20] Michael Kapralov, Slobodan Mitrovic, Ashkan Norouzi-Fard, and Jakab Tardos. Space efficient approximation to maximum matching size from uniform edge samples. In Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020, pages 1753–1772, 2020.

[KN97] Eyal Kushilevitz and Noam Nisan. Communication complexity. Cambridge University Press, 1997.

[KNST19] Michael Kapralov, Navid Nouri, Aaron Sidford, and Jakab Tardos. Dynamic streaming spectral sparsification in nearly linear time and space. CoRR, abs/1903.12150, 2019.

[Kon15] Christian Konrad. Maximum matching in turnstile streams. In Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings, pages 840–852, 2015.
[KW14] Michael Kapralov and David P. Woodruff. Spanners and sparsifiers in dynamic streams. In ACM Symposium on Principles of Distributed Computing, PODC ’14, Paris, France, July 15-18, 2014, pages 272–281, 2014.

[Lin09] Henry Lin. Reducing Directed Max Flow to Undirected Max Flow, 2009.

[LNW14] Yi Li, Huy L. Nguyen, and David P. Woodruff. Turnstile streaming algorithms might as well be linear sketches. In Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014, pages 174–183, 2014.

[McG05a] Andrew McGregor. Finding graph matchings in data streams. In Approximation, Randomization and Combinatorial Optimization, Algorithms and Techniques, 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2005 and 9th International Workshop on Randomization and Computation, RANDOM 2005, Berkeley, CA, USA, August 22-24, 2005, Proceedings, pages 170–181, 2005.

[McG05b] Andrew McGregor. Finding graph matchings in data streams. In Approximation, Randomization and Combinatorial Optimization, Algorithms and Techniques, 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2005 and 9th International Workshop on Randomization and Computation, RANDOM 2005, Berkeley, CA, USA, August 22-24, 2005, Proceedings, pages 170–181, 2005.

[McG14] Andrew McGregor. Graph stream algorithms: a survey. SIGMOD Record, 43(1):9–20, 2014.

[NW91] Noam Nisan and Avi Wigderson. Rounds in communication complexity revisited. In Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, May 5-8, 1991, New Orleans, Louisiana, USA, pages 419–429, 1991.

[PRV99] Stephen Ponzio, Jaikumar Radhakrishnan, and Srinivasan Venkatesh. The communication complexity of pointer chasing: Applications of entropy and sampling. In Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing, May 1-4, 1999, Atlanta, Georgia, USA, pages 602–611, 1999.

[PS82] Christos H. Papadimitriou and Michael Sipser. Communication complexity. In Harry R. Lewis, Barbara B. Simons, Walter A. Burkhard, and Lawrence H. Landweber, editors, Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA, pages 196–200. ACM, 1982.

[PS17] Ami Paz and Gregory Schwartzman. A \((2 + \varepsilon)\)-approximation for maximum weight matching in the semi-streaming model. In Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19, pages 2153–2161, 2017.

[Ros06] Jeffrey S Rosenthal. A first look at rigorous probability theory. World Scientific Publishing Company; 2006.

[RSW18] Aviad Rubinstein, Tselil Schramm, and S. Matthew Weinberg. Computing exact minimum cuts without knowing the graph. In 9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA, pages 39:1–39:16, 2018.
[SGP11] Atish Das Sarma, Sreenivas Gollapudi, and Rina Panigrahy. Estimating pagerank on graph streams. *J. ACM*, 58(3):13:1–13:19, 2011.

[sub] List of Open Problems in Sublinear Algorithms. [https://sublinear.info/index.php](https://sublinear.info/index.php).

[Tir18] Sumedh Tirodkar. Deterministic algorithms for maximum matching on general graphs in the semi-streaming model. In 38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2018, December 11-13, 2018, Ahmedabad, India, pages 39:1–39:16, 2018.

[Yao79] Andrew Chi-Chih Yao. Some complexity questions related to distributive computing (preliminary report). In *Proceedings of the 11th Annual ACM Symposium on Theory of Computing*, April 30 - May 2, 1979, Atlanta, Georgia, USA, pages 209–213, 1979.

[Yao83] Andrew Chi-Chih Yao. Lower bounds by probabilistic arguments (extended abstract). In 24th Annual Symposium on Foundations of Computer Science, Tucson, Arizona, USA, 7-9 November 1983, pages 420–428, 1983.

[Zel08] Mariano Zelke. Weighted matching in the semi-streaming model. In *STACS 2008, 25th Annual Symposium on Theoretical Aspects of Computer Science*, Bordeaux, France, February 21-23, 2008, *Proceedings*, pages 669–680, 2008.

[Zel11] Mariano Zelke. Intractability of min- and max-cut in streaming graphs. *Inf. Process. Lett.*, 111(3):145–150, 2011.
A Additional Preliminaries

We use the following fact about functions of discrete random variables.\footnote{A similar fact also holds for functions of continuous random variables, if the functions satisfy some additional natural properties, for example, if they are Borel-measurable (see \cite{R06} pages 31 and 32).}

**Proposition A.1.** Let $A, B \in \mathcal{X}$ be two independent random variables over a discrete universe $\mathcal{X}$, and $f, g, h : \mathcal{X} \to \mathcal{X}$ be functions then $f(A) \perp g(B) \mid h(A)$.

**Proof.** Since $f, g, h$ are discrete functions their inverses $f^{-1}, g^{-1}, h^{-1} : \mathcal{X} \to \mathcal{X}$ exist. For any set of values $A_1, A_2, B_1 \subseteq \mathcal{X}$ we have

\[
\Pr\left(f(A) \in A_1 \land g(B) \in B_1 \mid h(A) \in A_2\right) = \Pr\left(A \in f^{-1}(A_1) \land B \in g^{-1}(B_1) \mid A \in h^{-1}(A_2)\right) = \Pr\left(A \in f^{-1}(A_1) \mid A \in h^{-1}(A_2)\right) \cdot \Pr\left(B \in g^{-1}(B_1) \mid A \in h^{-1}(A_2)\right) = \Pr\left(f(A) \in A_1 \mid h(A) \in A_2\right) \cdot \Pr\left(g(B) \in B_1 \mid h(A) \in A_2\right).
\]

Thus it follows that $f(A) \perp g(B) \mid h(A)$.

We use the following basic properties of the Shannon entropy and mutual information in the paper. We refer the reader to the excellent treatise of the topic by Cover and Thomas \cite{CT12} for their proofs.

**Proposition A.2 (cf. \cite{CT12}).** Given four (possibly correlated) random variables $A, B, C, D$

(a) (Non-negativity of entropy) $\mathbb{H}(A) \geq 0$.

(b) (Conditioning decreases entropy) $\mathbb{H}(A \mid B) \leq \mathbb{H}(A)$, with equality iff $A \perp B$.

(c) (Chain of entropy) $\mathbb{H}(A, B \mid C) = \mathbb{H}(A \mid C) + \mathbb{H}(B \mid A, C)$.

(d) (Mutual information) $I(A ; B) := \mathbb{H}(A) - \mathbb{H}(A \mid B)$.

(e) (Non-negativity of mutual information) $I(A ; B) \geq 0$, with equality iff $A \perp B$.

(f) (Chain of mutual information) $I(A, B ; C \mid D) = I(A ; C \mid D) + I(B ; C \mid A, D)$.

**Proposition A.3.** If $B \perp D \mid C$, then

\[
I(A ; B \mid C, D) \geq I(A ; B \mid C).
\]

**Proof.** Since $B$ and $D$ are independent conditioned on $C$, from Proposition A.2(b) we have that

\[
\mathbb{H}(B \mid C, D) = \mathbb{H}(B \mid C) \quad (16)
\]
\[
\mathbb{H}(B \mid A, C) \geq \mathbb{H}(B \mid A, C, D). \quad (17)
\]

Using the above two equations we get

\[
I(A ; B \mid C) = \mathbb{H}(B \mid C) - \mathbb{H}(B \mid A, C) = \mathbb{H}(B \mid C, D) - \mathbb{H}(B \mid A, C) \leq I(A ; B \mid C, D).
\]
Proposition A.4. Given (possibly correlated) random variables $A_1, \ldots, A_n$ and $B$ we have
\[
\mathbb{I}(A_1, A_2, \ldots, A_n; B) = \sum_{i \in [n]} \mathbb{I}(A_i; B \mid A_1, A_2, \ldots, A_{i-1}).
\]

Proof. Proposition A.4 follows by repeated applications of Proposition A.2.f.
\[
\mathbb{I}(A_1, A_2, \ldots, A_{m+1}; B) \overset{\text{Prop. A.2.f}}{=} \mathbb{I}(A_1; B) + \mathbb{I}(A_2, \ldots, A_{m+1}; B \mid A_1)
\]
\[
\mathbb{I}(A_1; B) + \mathbb{I}(A_2; B \mid A_1) + \mathbb{I}(A_3, \ldots, A_{m+1}; B \mid A_1, A_2)
\]
\[
\vdots
\]
\[
\sum_{i \in [m+1]} \mathbb{I}(A_i; B \mid A_1, A_2, \ldots, A_{i-1}).
\]

Proposition A.5. Given random variables $A, B, C,$ and $D$ we have
\[
\mathbb{I}(A, B; C \mid D) \geq \mathbb{I}(A; C \mid D).
\]

Proof. The proof follows by applying Proposition A.2.e and Proposition A.2.f
\[
\mathbb{I}(A, B; C \mid D) \overset{\text{Prop. A.2.f}}{=} \mathbb{I}(A; C \mid D) + \mathbb{I}(B; C \mid A, D)
\]
\[
\overset{\text{Prop. A.2.e}}{\geq} \mathbb{I}(A; C \mid D).
\]

B Four Player HPC ($\text{HPC}_k^4$)

$\text{HPC}_k^4$ is a $k$-round problem with four players ($P_A, P_B, P_C, P_D$). The players are grouped into two pairs, ($P_A, P_B$) and ($P_C, P_D$), where each pair is given $n$ instances of Set-Int. The goal of $\text{HPC}_k^4$ is to start from a fixed instance $s_0$, of $P_A$ and $P_B$, and follow the pointers for a fixed number of steps, and output the $k$-th pointer.

Formally, we have two disjoint sets, $\mathcal{X} := \{x_1, \ldots, x_n\}$ and $\mathcal{Y} := \{y_1, \ldots, y_n\}$. Input to ($P_A, P_B$) are the $n$ instances of Set-Int over universe $\mathcal{Y}$, $(A_{x_i}, B_{x_i}), i \in [n]$, and the input of $(P_C, P_D)$ are the $n$ instances of Set-Int over universe $\mathcal{X}$, $(C_{y_i}, D_{y_i}), i \in [n]$. Given $s_0 \in \mathcal{X}$, define $i$-th pointer, $s_i$ inductively as

1. $s_i := t_{s_i-1}$ in the instance $(A_{s_{i-1}}, B_{s_{i-1}})$ of Set-Int if $i$ is odd, and

2. $s_i := t_{s_i-1}$ in the instance $(C_{s_{i-1}}, D_{s_{i-1}})$ of Set-Int if $i$ is even.

In $\text{HPC}_k^4$ the communication is divided into $k$-rounds\footnote{Note that the rounds in $\text{HPC}_k^4$ are different from the rounds in $\text{HPC}_k^2$. With some abuse of notation we refer to both as “rounds”. We hope that the particular definition is clear from the context.}. The pair ($P_C, P_D$) communicate (that is, are active) in the odd rounds (1, 3, . . . ) and ($P_A, P_B$) communicate (that is, are active) in the even rounds (2, 4, . . . ). The players can exchange any number of messages in each round, and the round ends when the “active” pair sends one message to the “dormant” pair.

The goal of $\text{HPC}_k^4$ is to find the $k$-th pointer $s_k$, with the least amount of communication.
C Proofs of prior lower bounds by HPC\(^4\) using HPC\(^2\)

C.1 Proof of Claim 3.7

Claim C.1 (Restatement of Claim 3.7). Given an instance \((A, B)\) of HPC\(^2\)(\(k-1\))\(-1\), let \(s_k\) be the solution of HPC\(^4\)\(k\) with input \((A, B, A, B)\), then \(z_k = s_k\).

Proof. We will show this by induction on \(k\). Consider the given instance \((A', B', C', D') := (A, B, A, B)\) of HPC\(^4\)\(h\). Let \(s_h\) be the \(k\)-th pointer (see Section B for the definition) of HPC\(^4\)\(h\) on this input.

Base case \((k = 1)\). By definition,

\[
\{s_1\} = A'_{z_0} \cap B'_{z_0} = A_{z_0} \cap B_{z_0} = \{z_1\}.
\]

Induction hypothesis. Assume that \(s_h = z_h\).

Induction step. If \(h + 1\) is odd we have,

\[
\{s_{h+1}\} = A'_{s_h} \cap B'_{s_h} = A_{s_h} \cap B_{s_h}
= A_{z_h} \cap B_{z_h} \quad \text{(Induction hypothesis)}
= \{z_{h+1}\}.
\]

Similarly, if \(h + 1\) is even we have,

\[
\{s_{h+1}\} = C'_{s_h} \cap D'_{s_h} = A_{s_h} \cap B_{s_h}
= A_{z_h} \cap B_{z_h} \quad \text{(Induction hypothesis)}
= \{z_{h+1}\}.
\]

Now we have \(s_{h+1} = z_{h+1}\). Thus, the claim is true by principle of induction. \(\square\)

C.2 Lower bound for Max-Flow in the streaming model

Theorem C.2 (Polynomial pass lower bound for Max-Flow). Any \(p\)-pass streaming algorithm that finds the maximum flow in an \(n\)-vertex weighted graph (directed or undirected) requires \(\Omega(n^2/p^3)\) space. By the max-flow min-cut theorem, the same lower bound holds for computing \(s-t\) min-cut in weighted graphs.

Proof. Given an instance of HPC\(^2\)(\(k-1\)), \((A, B)\) over universe \([n]\), we turn it into an instance of Max-Flow. We construct a directed graph, \(G(V, E, w)\) as follows:

- Partition the vertices into \(k + 3\) layers, \(V := \{s\} \cup V_0 \cup V_1 \cup \cdots \cup V_k \cup \{t\}\). Nodes \(s\) and \(t\) represent source and sink respectively.
- Layer \(V_j, 0 \leq j \leq k\) has \(n\) vertex indexed by \(1 \leq i \leq n\). Vertex indexed \(i\) in layer \(V_j\) is denoted by \(v_{(j,i)}\).
- Define the following sequence of weights \(w_0, w_1, \ldots, w_k\), where \(w_j := (n+1)^{k+1-j}\) for \(0 \leq j \leq k\).
- \(E\) contains the following input-independent edges.
- source $s$ is connected to $v_{(0,1)}$ with an edge of weight $w(s,v_{(0,1)}) = w_0$.
- for $0 < j \leq k$, every vertex $v_{(j,i)} \in V_j$ is connected to sink $t$ with weight $w(v_{(j,i)},t) = w_j$.
- any vertex $v_{(k,i)}$ in layer $V_k$ is connected to sink $t$ with weight $w(v_{(k,i)},t) = i - 1$ (notice that $v_{(k,i)}$ also has another edge of weight $w_k$ to $t$ by the previous part).

- $E$ contains the following input-dependent edges.
- for $i \in [n]$, if $A_i \in A$ (resp. $B_i \in B$) contains $\ell \in [n]$, we connect $v_{(j,i)} \in V_j$ to $v_{(j+1,\ell)} \in V_{j+1}$ with weight $w_{j+1}$ for $0 \leq j < k$.

This concludes the description of the graph $G(V,E,w)$ in the reduction. Clearly, this graph can be constructed from an instance $(A, B)$ with no communication between the players using a graph stream.

Now, observe that $G$ is same as the graph constructed in the reduction of [ACK19a] from HPC$_k^4$ with instance $(A, B, A, B)$ to Max-Flow, $G_4(A, B, A, B)$. According to Lemma 6.1 of [ACK19a], the weight of $s$-$t$ min-cut in graph $G = G_4(A, B, A, B)$ gives the value $S_k$. Hence, by claim 3.7, the following lemma holds, showing the correctness of the reduction.

**Lemma C.3.** Let $w^*$ be the maximum $s$-$t$ flow in graph $G$ in the reduction, then $z_k = (w^* \mod (n+1)) + 1$.

We can now prove Theorem C.2 using this reduction and the standard connection between communication protocols and streaming algorithms.

From an instance of HPC$_{k-1}^2$ with $k = 2p + 1$, players construct the graph $G$ without any communication. Let $N$ be the number of vertices in $G$. Next, they create a stream of edges $\sigma$ in which edges depending on $P_B$’s input appear first and then the edges depending on $P_A$’s input appear. Finally, the input-independent edges appear.

Let $A$ be any algorithm making $p$-passes over $\sigma$ for finding the maximum $s$-$t$ flow in $G$. From Lemma C.3, the value of the maximum $s$-$t$ flow in $G$ immediately determines the pointer $z_k$. Thus, we can get a protocol $\pi_{\text{HPC2}}$ for HPC$_k^2$ using $A$. Each pass of $A$ over $\sigma$ is translated to at most two rounds of $\pi_{\text{HPC2}}$, and hence $\pi_{\text{HPC2}}$ has at most $k - 1$ rounds. So, total communication during the run of protocol $\pi_{\text{HPC2}}$ is $O(p \cdot S)$ where $S$ is the space complexity of $A$. We have

\[
\begin{align*}
\text{CC}(\pi_{\text{HPC2}}) & = \Omega(n^2/k^2), \\
p \cdot S & = \Omega(n^2/k^2), \\
S & = \Omega(N^2/(k^4 \cdot p)), \\
& = \Omega(N^2/p^5).
\end{align*}
\]

Thus, we find that the space complexity of $A$ is $\Omega(N^2/p^5)$, completing the proof.

Note that this lower bound for computing $s$-$t$ min-cut is also applicable for undirected and simple graphs, see [Lin09] for the procedure to convert the constructed graph into undirected and simple graph.

---

Note that we will add two edges between $v_{(j,i)}$ and $v_{(j+1,\ell)}$ iff $\ell \in A_i \cap B_i$ and we will keep both copies of these edges in $G$. Thus, $G$ is now a multi-graph.
C.3 Lower bound on LMIS in the streaming model

Theorem C.4 (Polynomial pass lower bound for LMIS). Any $p$-pass streaming algorithm that outputs $\text{LMIS}(G)$, for an undirected graph $G(V, E)$, $|V| = n$, with a constant probability requires $\Omega(n^2/p^5)$ space.

*Proof.* Given an instance of $\text{HPC}_k^4$, $(A, B)$ over universe $[n]$, we turn it into an instance of $\text{LMIS}$. The construction of graph $G(V, E, w)$ is as follows:

- Partition the vertices into $k + 1$ layers, $V := V_0 \cup V_1 \cup \cdots \cup V_k$.
- Each layer, $V_j$, $0 \leq j \leq k$ has $n$ vertex indexed by $1 \leq i \leq n$. Vertex indexed $i$ in layer $V_j$ is denoted by $v_{(j, i)}$. Lexicographically, the vertex in layer $V_0$ appear first, followed by the vertex in $V_1, \ldots, V_k$ in this order. Inside layer $V_j$, $0 \leq j \leq k$, the ordering is by the index, that is, in the order $v_{(j, 1)}, \ldots, v_{(j, n)}$.
- $E$ contains the following *input-independent* edges.
  - vertex $v_{(0, 1)}$ is connected to all other vertices in $V_0$.
- $E$ contains the following *input-dependent* edges.
  - for $i \in [n]$, if $A_i \in A$ (resp. $B_i \in B$) does not contain $\ell \in [n]$, we connect $v_{(j, i)} \in V_j$ to $v_{(j+1, \ell)} \in V_{j+1}$ for $0 \leq j < k$.

This concludes the description of the graph $G(V, E, w)$ in the reduction. Clearly, this graph can be constructed from an instance $(A, B)$ with no communication between the players using a graph stream.

Now, observe that $G$ is same as the graph constructed in the reduction of $\text{ACK19a}$ from $\text{HPC}_k^4$ with instance $(A, B, A, B)$ to $\text{LMIS}$, $G_4(A, B, A, B)$. According to Lemma 6.6 of $\text{ACK19a}$, $S_k$ is determined by the vertex of layer $V_k$ in $\text{LMIS}$ of $G = G_4(A, B, A, B)$. Hence, by claim 3.7, the following lemma holds, showing the correctness of the reduction.

**Lemma C.5.** In the reduction above, the pointer $z_k = i$ iff $v_{(k, i)}$ belongs to the lexicographically-first maximal independent set of $G$.

We can now prove Theorem C.4 using this reduction and the standard connection between communication protocols and streaming algorithms.

From an instance of $\text{HPC}_k^{32}$ with $k = 2p + 1$, players construct the graph $G$ without any communication. Let $N$ be the number of vertices in $G$. Next, they create a stream of edges $\sigma$ in which edges depending on $P_B$’s input appear first and then the edges depending on $P_A$’s input appear. Finally, the input-independent edges appear.

Let $\mathcal{A}$ be any algorithm making $p$-passes over $\sigma$ for finding the LMIS in $G$. From Lemma C.5, we can determine the pointer $z_k$ by knowing which vertex of layer $V_k$ is present in LMIS of $G$. Thus, we can get a protocol $\pi_{\text{HPC}^2}$ for $\text{HPC}_k^{32}$ using $\mathcal{A}$. Each pass of $\mathcal{A}$ over $\sigma$ is translated to at most two rounds of $\pi_{\text{HPC}^2}$, and hence $\pi_{\text{HPC}^2}$ has at most $k - 1$ rounds. So, total communication during the run of protocol $\pi_{\text{HPC}^2}$ is $O(p \cdot S)$ where $S$ is the space complexity of $\mathcal{A}$. We have

$$\text{CC}(\pi_{\text{HPC}^2}) \begin{eqnarray*} & = & \Omega(n^2/k^2), \\
p \cdot S & = & \Omega(n^2/k^2), \\
S & = & \Omega(N^2/(k^4 \cdot p)), \\
& = & \Omega(N^2/p^5). \end{eqnarray*}$$

Thus, we find that the space complexity of $\mathcal{A}$ is $\Omega(N^2/p^5)$, completing the proof. \hfill \Box
C.4 Proof of Theorem 3.6

Proof. The proof relies on the fact that s-t cut function is sub-modular and uses the reduction from HPC_{2k+1}^2 used in the proof of Theorem C.2. Let G(V, E, w) be the graph formed from an instance of HPC_{2k+1}^2 in the proof of Theorem C.2. Define U := V \ {s, t} and f(S) to be the value of the cut between (S \cup \{s\}, V \setminus (S \cup \{s\})). Clearly, f(S) ≤ \sum_{e \in E} w_e. By construction of G, |U| = N = O(nk) and M = O(n^{k+1}). Also, min_{S \subseteq U} f(S) corresponds to computing the min s-t cut of G. Now, we turn a k-adaptive algorithm, A, for computing f into a protocol for HPC_{2k+1}^2. Let the query complexity of A is Q. First, observe that any query by A can be answered by the players in HPC_{2k+1}^2 using O(\log M) bits. If A asks for a query S, then each player determine the weight of edges crossing the cut (S \cup \{s\}, V \setminus (S \cup \{s\})) and communicates to others using O(\log M) bits. Finding the weight of the input-dependent edges crossing the cut does not require any communication, and thus players can compute f(S) using O(\log M) bits. So, players can get a protocol π_{HPC2} for HPC_{2k+1}^2 using O(Q \cdot \log M) total communication. The correctness of the protocol follows from Lemma C.3 in the proof of Theorem C.2. Also, each round of adaptive query translates into at most two rounds in π_{HPC2}. So, π_{HPC2} has less than 2k + 1 rounds. Thus we have,

\[ CC(π_{HPC2}) = Q \cdot \log M, \]

Theorem 3.2 \[ = \Omega\left(\frac{n^2}{k^2}\right). \]

Simplifying this we get

\[ Q = \Omega\left(\frac{N^2}{k^4 \cdot \log M}\right), \]

\[ = \Omega\left(\frac{N^2}{k^5 \log n}\right), \]

\[ = \Omega\left(\frac{N^2}{k^5 \log N}\right). \]

Thus, we find that the k-round adaptive query complexity of A is \( \Omega\left(\frac{N^2}{k^5 \log N}\right) \), completing the proof. \( \blacksquare \)