Edit Distance to Monotonicity
in Sliding Windows

Ho-Leung Chan[1], Tak-Wah Lam[1,⋆], Lap-Kei Lee[2], Jiangwei Pan[1],
Hing-Fung Ting[1], and Qin Zhang[2]

[1] Department of Computer Science, University of Hong Kong, Hong Kong
{hlchan,twlam,jwpan, hfting}@cs.hku.hk
[2] MADALGO⋆⋆, Department of Computer Science, Aarhus University, Denmark
{lklee,qinzhang}@madalgo.au.dk

Abstract. Given a stream of items each associated with a numerical
value, its edit distance to monotonicity is the minimum number of items
to remove so that the remaining items are non-decreasing with respect to
the numerical value. The space complexity of estimating the edit distance
to monotonicity of a data stream is becoming well-understood over the
past few years. Motivated by applications on network quality monitoring,
we extend the study to estimating the edit distance to monotonicity of
a sliding window covering the w most recent items in the stream for any
$w \geq 1$. We give a deterministic algorithm which can return an estimate
within a factor of $(4 + \epsilon)$ using $O(\frac{1}{\epsilon^2} \log^2(\epsilon w))$ space.

We also extend the study in two directions. First, we consider a
stream where each item is associated with a value from a partial or-
dered set. We give a randomized $(4 + \epsilon)$-approximate algorithm using
$O(\frac{1}{\epsilon^2} \log \epsilon^2 w \log w)$ space. Second, we consider an out-of-order stream
where each item is associated with a creation time and a numerical value,
and items may be out of order with respect to their creation times. The
goal is to estimate the edit distance to monotonicity with respect to
the numerical value of items arranged in the order of creation times.
We show that any randomized constant-approximate algorithm requires
linear space.

1 Introduction

Estimating the sortedness of a numerical sequence has found applications in, e.g.,
sorting algorithms, database management and webpage ranking (such as Pager-
ank [4]). For example, sorting algorithms can take advantage of knowing the
sortedness of a sequence so as to sort efficiently [9]. In relational database, many
operations are best performed when the relations are sorted or nearly sorted over
the relevant attributes [3]. Maintaining an estimate on the sortedness of the rela-
tions can help determining whether a given relation is sufficiently nearly-sorted
or a sorting operation on the relation (which is expensive) is needed. One com-
mon measurement of sortedness of a sequence is its *edit distance to monotonicity*

(or ED, in short) [2,7,8,10,11]: given a sequence σ of n items, each associated with a value in $[m] = \{1, 2, \ldots, m\}$, the ED of σ, denoted by $\text{ed}(\sigma)$, is the minimum number of edit operations required to transform σ to the sequence obtained by sorting σ in non-decreasing order. Here, an edit operation involves removing an item and re-insert it into a new position of the sequence. Equivalently, $\text{ed}(\sigma)$ is the minimum number of items in σ to delete so that the remaining items have non-decreasing values. A closely related measurement is the *length of the longest increasing subsequence* (or LIS) of σ, denoted by $\text{lis}(\sigma)$. It is not hard to see that $\text{lis}(\sigma) = n - \text{ed}(\sigma)$.

With the rapid advance of data collection technologies, the sequences usually appear in the form of a data stream, where the stream of items is massive in size (containing possibly billions of items) and the items are rapidly arriving sequentially. This gives rise to the problem of estimating ED in the data stream model: An algorithm is only allowed to scan the sequence sequentially in one pass, and it also needs to be able to return, at any time, an estimate on ED of the items arrived so far. The main concern is the space usage and update time per item arrival, which, ideally, should both be significantly smaller than the total data size (preferably polylogarithmic).

Estimating ED of a data stream is becoming well-understood over the past few years [8,10,11]. Gopalan et al. [11] showed that computing the ED of a stream exactly requires $\Omega(n)$ space even for randomized algorithms, where n is the number of items arrived so far. They also gave a randomized $(4 + \epsilon)$-approximate algorithm for estimating ED using space $O(\frac{1}{\epsilon^2} \log^2 n)$, where $0 < \epsilon < 1$. Later, Ergun and Jowhari [8] improved the result by giving a deterministic $(2 + \epsilon)$-approximate algorithm using space $O(\frac{1}{\epsilon^2} \log^2(\epsilon n))$. For the closely related LIS problem, Gopalan et al. [11] also gave a deterministic $(1 + \epsilon)$-approximate algorithm for estimating LIS using $O(\sqrt{\frac{n}{\epsilon}})$ space. This space bound is proven to be optimal in [10].

ED in sliding windows. The above results consider the sortedness of all items in the stream arrived so far, which corresponds to the *whole stream model*. Recently, it is suggested that ED can be an indicator of network quality [12]. The items of the stream correspond to the packets transmitted through a network, each associated with a sequence number. Ideally, the packets would arrive in increasing order of the sequence number. Yet network congestion would result in packet retransmission and distortion in the packet arrival order, which leads to a large ED value. One of the main causes to network congestion is that traffic is often bursty. Thus, the network quality can be measured more accurately if the measurement is based on only recent traffic. To this end, we propose studying the *sliding window model* where we estimate the ED of a window covering the latest w items in the stream. Here w is a positive integer representing the window size. The sliding window model is no easier than the whole data stream model because when w is set to be infinity, we need to estimate ED for all items arrived.

Our results. We give a deterministic $(4 + \epsilon)$-approximate algorithm for estimating ED in a sliding window. The space usage is $O(\frac{1}{\epsilon^2} \log^2(\epsilon w))$, where w is the

window size. Our algorithm is a generalization of the algorithm by Gopalan et al. [11]. In particular, Gopalan et al. show that ED of the whole stream can be approximated by the number of "inverted" items j such that many items arrived before j has a value bigger than j. We extend this definition to the sliding window model. Yet, maintaining the number of inverted items in a sliding window is non-trivial. An item j may be inverted when it arrives, but it may become not inverted due to the expiry of items *arrived earlier*. We give an interesting algorithm to estimate the number of inverted items using existing results on basic counting and quantile estimation over sliding windows. Our algorithm also incorporates an idea in [8] to remove randomization.

We also consider two extensions of the problem.

- *Partial ordered items.* In some applications, each item arrived is associated with multiple attributes, e.g., a network packet may contain both the IP address of the sender and a sequence number. To measure the network quality, it is sometimes useful to estimate the *most congested* traffic coming from a particular sender. This corresponds to estimating the ED of packets with respect to sequence number from the same sender IP address. In this case, only sequence numbers with the same IP address can be ordered. We model such a situation by considering items each associated with a value drawn from a partial ordered universe. We are interested in estimating the minimum number of items to delete so that the remaining items are sorted with respect to the partial order. We give a randomized $(4 + \epsilon)$-approximate algorithm using $O(\frac{1}{\epsilon^2} \log \epsilon^2 w \log w)$ space.

- *Out-of-order streams.* When a sender transmits packets to a receiver through a network, the packets will go through some intermediate routers. To measure the quality of the route between the sender and an intermediate router, it is desirable to estimate the ED of the packets received by the router from the sender. Yet in some cases, the router may not be powerful enough to deploy the algorithm for estimating the ED. We consider delegating the task of estimation to the receiver. To model the situation, whenever a packet arrives, the intermediate router marks in the packet a timestamp recording the number of packets received thus far (which can be done by maintaining a single counter). Hence, when the packets arrive at the receiver, each packet has both a sequence number assigned by the sender and a timestamp marked by the router. Note that the packets arrived at the receiver may be out-of-order with respect to the timestamp. Such stream corresponds to an *out-of-order stream*.

To measure the network quality between the sender and the router, the receiver can estimate the ED with respect to the sequence number when the items are arranged in increasing order of the timestamps. Intuitively, the problem is difficult as items can be inserted in arbitrary positions of the sequence according to the timestamp. We show strong space lower bounds even in the whole stream model. In particular, any randomized constant-approximate algorithm for estimating ED of an out-of-order stream requires $\Omega(n)$ space, where n is the number of items arrived so far. An identical lower bound holds for estimating the LIS. Like most streaming lower bounds, our lower bounds are proved based on reductions from two communication problems, namely, the INDEX problem

and the DISJ problem. Optimal communication lower bounds for randomized protocols are known for both problems [1,14].

Organization. Section 2 and 3 give the formal problem definitions and our main algorithm for estimating ED, respectively. Section 4 considers out-of-order streams. Due to the page limit, extension to partial ordered items is left to the full paper.

2   Formal Problem Definitions

Sortedness of a stream. Consider a stream σ of n items, $\langle \sigma(1), \sigma(2), \ldots, \sigma(n) \rangle$ where each $\sigma(i)$ is drawn from $[m] = \{1, 2, \ldots, m\}$. The *edit distance to monotonicity* (ED) of σ, denoted by $\text{ed}(\sigma)$, is the minimum number of items required to remove so as to obtain an increasing subsequence of σ, i.e., $\langle \sigma(i_1), \sigma(i_2), \ldots, \sigma(i_k) \rangle$ such that $\sigma(i_1) \leq \sigma(i_2) \leq \cdots \leq \sigma(i_k)$ for some $1 \leq i_1 < i_2 < \cdots < i_k \leq n$. We use $\text{lis}(\sigma)$ to denote the *length of the longest increasing subsequence* (LIS) of σ. Note that $\text{lis}(\sigma) = n - \text{ed}(\sigma)$. The sortedness can be computed based on the *whole stream* (all items in σ received thus far) or a *sliding window* covering the most recent w items, denoted by σ_w, for $w \geq 1$. Note that the whole stream model can be viewed as a special case of the sliding window model with window size $w = \infty$. A streaming algorithm has only limited space and can only maintain an estimate on the sortedness of σ_w. For any $r \geq 1$, a r-approximate algorithm for estimating $\text{ed}(\sigma_w)$ returns, at any time, an estimate $\widehat{\text{ed}}(\sigma_w)$ such that $\text{ed}(\sigma_w) \leq \widehat{\text{ed}}(\sigma_w) \leq r \cdot \text{ed}(\sigma_w)$. We can define a r-approximate algorithm for estimating $\text{lis}(\sigma_w)$ similarly.

Partial ordered universe. We also consider a partial ordered universe with binary relation $\preceq$. A subsequence of σ with length ℓ, $\langle \sigma(i_1), \sigma(i_2), \cdots, \sigma(i_\ell) \rangle$, is increasing if for any $k \in [\ell - 1]$, $\sigma(i_k) \preceq \sigma(i_{k+1})$. Then for any window size $w \geq 1$, $\text{ed}(\sigma_w)$ and $\text{lis}(\sigma_w)$ can be defined analogously as before.

Out-of-order stream. The data stream described above is an *in-order* stream, which assumes items arriving in the same order as their creation time. In an *out-of-order stream*, each item is associated with a distinct integral time-stamp recording its creation time, which may be different from its arrival time. Precisely, an out-of-order stream σ is a sequence of tuples $\langle t_i, v_i \rangle$ $(i \in [n])$ where t_i and v_i are the timestamp and value of the i-th item. The sortedness of σ is defined based on the permuted sequence $V(\sigma) = \langle v_{i_1}, v_{i_2}, \ldots, v_{i_n} \rangle$ such that $t_{i_1} \leq t_{i_2} \leq \cdots \leq t_{i_n}$, i.e., $\text{ed}(\sigma) := \text{ed}(V(\sigma))$ and $\text{lis}(\sigma) := \text{lis}(V(\sigma))$.

3   A $(4 + \epsilon)$-Approximate Algorithm for Estimating ED

In this section, we consider a stream σ of items with values drawn from a set $[m] = \{1, 2, \ldots, m\}$, and we are interested in estimating the ED of a sliding window covering the most recent w items in σ. We give a deterministic $(4 + \epsilon)$-approximate algorithm which uses $O(\frac{1}{\epsilon^2} \log^2(\epsilon w))$ space.

Our algorithm is based on an estimator $R(i)$, which is a generalization of the estimator in [11] to the sliding window model. Let i be the index of the latest arrived item. The sliding window we consider is $\sigma_{[i-w+1,i]} = \langle \sigma(i-w+1), \sigma(i-w+2), \ldots, \sigma(i) \rangle$. For any item $\sigma(j)$, let $inv(j)$ be the set of items arrived before $\sigma(j)$ but have greater values than $\sigma(j)$, i.e., $inv(j) = \{k : k < j \text{ and } \sigma(k) > \sigma(j)\}$. We define an estimator $R(i)$ for $\text{ed}(\sigma_{[i-w+1,i]})$ as follows.

Definition 1. *Consider the current sliding window $\sigma_{[i-w+1,i]}$. We define $R(i)$ to be the set of indices $j \in [i-w+1, i]$ such that there exists $k \in [i-w+1, j-1]$ with $|[k, j-1] \cap inv(j)| > \frac{j-k}{2}$.*

Lemma 1 ([11]). $\text{ed}(\sigma_{[i-w+1,i]})/2 \le |R(i)| \le 2 \cdot \text{ed}(\sigma_{[i-w+1,i]})$.

Hence, if we know $|R(i)|$, we can return $2|R(i)|$ as an estimation for $\text{ed}(\sigma_{[i-w+1,i]})$ and it gives a 4-approximation algorithm. However, maintaining $R(i)$ exactly requires space linear to the window size. In the following, we show how to approximate $R(i)$ using significantly less space.

3.1   Estimating $R(i)$

We first present our algorithm and then show that it can approximate $R(i)$. Our algorithm will make use of two data structures. Let ϵ' be a constant in $(0, 1)$ (which will be set to $\epsilon/35$ later).

ϵ'**-approximate quantile data structure $\mathcal{Q}$**: Let Q be a set of items. The rank of an item in Q is its position in the list formed by sorting Q from the smallest to the biggest. For any $\phi \in [0, 1]$, the ϵ'-approximate ϕ-quantile of Q is an item with rank in $[(\phi - \epsilon')|Q|, (\phi + \epsilon')|Q|]$. We maintain an ϵ'-approximate ϕ-quantile data structure given in [13] which can return, at any time, an ϵ'-approximate ϕ-quantile of the most recent w' items for any $w' \le w$. This data structure takes $O(\frac{1}{(\epsilon')^2} \log^2(\epsilon'w))$ space.

ϵ'**-approximate basic counting data structure $\mathcal{B}$**: When an item $\sigma(i)$ arrives, we may associate a token with some item $\sigma(k)$ where $k < i$. The association is permanent and an item may be associated with more than one token. At any time, we are interested in the number of tokens associated with the most recent w items. We view it as a stream σ_{token} of tokens, each of which has a timestamp k if it is associated to $\sigma(k)$, and we want to return the number of tokens with timestamp in $[i-w+1, i]$. Note that the tokens may be out-of-order with respect to the timestamp, leading to the basic counting problem for out-of-order stream considered in [6]. We maintain their ϵ'-approximate basic counting data structure on σ_{token} which can return, at any time, an estimate $\hat{t}$ such that $|\hat{t} - t| \le \epsilon't$, where t is the number of tokens associated with the latest w items. It takes $O(\frac{1}{\epsilon'} \log w \log(\frac{\epsilon'B}{\log w}))$ space, where B is the maximum number of tokens associated within any window of w items. As we may associate one token upon any item arrival, B is at most w.

We are now ready to define our algorithm, as follows.

Algorithm 1. Estimating ED in sliding windows

Item arrival: Upon the arrival of item $\sigma(i)$, do

    For $k = i - 1, i - 2, \cdots, i - w + 1$

        Query $\mathcal{Q}$ for the $(\frac{1}{2} - \epsilon')$-quantile of $\sigma_{[k,i-1]}$. Let a be the returned value.
        If $a > \sigma(i)$, associate a token to $\sigma(k)$, i.e., add an item with timestamp k
        to the stream σ_{token}. Break the for loop.

Query: Query $\mathcal{B}$ on the stream σ_{token} for the number of tokens associated with the last w items and let $\hat{t}$ be the returned answer. Return $\hat{t}/(\frac{1}{2} - 2\epsilon')(1 - \epsilon')$ as the estimation $\widehat{\text{ed}}(\sigma_{[i-w+1,i]})$.

Let $R'(i)$ be the set of indices j such that when $\sigma(j)$ arrives, we associate a token to an item $\sigma(k)$ where $k \in [i - w + 1, i]$. Observe that $R'(i)$ is an approximation of $R(i)$ in the following sense.

Lemma 2. $R'(i)$ *contains all indices* $j \in [i - w + 1, i]$ *satisfying that there exists* $k \in [i - w + 1, j - 1]$ *such that* $|[k, j - 1] \cap inv(j)| > (\frac{1}{2} + 2\epsilon')(j - k)$. *Furthermore, all indices* j *contained in* $R'(i)$ *must satisfy that there exists* $k \in [i - w + 1, j - 1]$ *such that* $|[k, j - 1] \cap inv(j)| > \frac{j - k}{2}$.

Proof. An index j is in $R'(i)$ if $\sigma(j) < a$ when $\sigma(j)$ arrives, where a is the ϵ'-approximate $(\frac{1}{2} - \epsilon')$-quantile for some interval $\sigma_{[k,j-1]}$. Note that the rank of a in $\sigma_{[k,j-1]}$ is at least $(\frac{1}{2} - 2\epsilon')(j - k)$. Therefore, if $|[k, j - 1] \cap inv(j)| > (\frac{1}{2} + 2\epsilon')(j - k)$, the rank of $\sigma(j)$ is less than $(j - k) - (\frac{1}{2} + 2\epsilon')(j - k) = (\frac{1}{2} - 2\epsilon')(j - k)$, so $\sigma(j) < a$ and j must be included in $R'(i)$. On the other hand, the rank of a in $\sigma_{[k,j-1]}$ is at most $\frac{j-k}{2}$. Since $a > \sigma(j)$, we conclude that all indices $j \in R'(i)$ satisfy $|[k, j - 1] \cap inv(j)| > \frac{j-k}{2}$. $\qquad\square$

We show that $|R'(i)|$ is a good approximation for $\text{ed}(\sigma_{[i-w+1,i]})$, as follows.

Lemma 3. $(\frac{1}{2} - 2\epsilon') \cdot \text{ed}(\sigma_{[i-w+1,i]}) \leq |R'(i)| \leq 2 \cdot \text{ed}(\sigma_{[i-w+1,i]})$.

Proof. We observe that by Lemma 2, any index j in $R'(i)$ must be also in $R(i)$. Hence, $R'(i) \subseteq R(i)$ and $|R'(i)| \leq |R(i)| \leq 2 \cdot \text{ed}(\sigma_{[i-w+1,i]})$ (by Lemma 1).

Now, we show $(\frac{1}{2} - 2\epsilon') \cdot \text{ed}(\sigma_{[i-w+1,i]}) \leq |R'(i)|$ by giving an iterative pruning procedure to obtain an increasing subsequence (may not be the longest). First let $x = i + 1$ and $\sigma(x) = \infty$. Find the largest j such that $i - w + 1 \leq j < x$ and $j \notin R'(i) \cup inv(x)$ and delete the interval $[j + 1, x - 1]$. We then let $x = j$ and repeat the process until no such j is found. As each x is not in $R'(i)$, Lemma 2 implies that in every interval that we delete, the fraction of items of $R'(i)$ is at least $(\frac{1}{2} - 2\epsilon')$. Note that eventually all items in $R'(i)$ will be deleted. Thus, $|R'(i)| \geq (\frac{1}{2} - 2\epsilon') \cdot (\text{number of deleted items}) \geq (\frac{1}{2} - 2\epsilon') \cdot \text{ed}(\sigma_{[i-w+1,i]})$. $\qquad\square$

Note that $|R'(i)|$ equals the number of tokens associated with the most recent w items. Since $\mathcal{B}$ is only an ϵ'-approximate data structure, the value $\hat{t}$ returned only

satisfies that $(1-\epsilon')|R'(i)| \leq \hat{t} \leq (1+\epsilon')|R'(i)|$. Since we report $\widehat{\mathrm{ed}}(\sigma_{[i-w+1,i]}) = \hat{t}/(\frac{1}{2}-2\epsilon')(1-\epsilon')$ as the estimation, we conclude with the following approximation ratio.

Lemma 4. $\mathrm{ed}(\sigma_{[i-w+1,i]}) \leq \widehat{\mathrm{ed}}(\sigma_{[i-w+1,i]}) \leq \frac{2(1+\epsilon')}{(1/2-2\epsilon')(1-\epsilon')} \cdot \mathrm{ed}(\sigma_{[i-w+1,i]})$.

For any $\epsilon \leq 1$, we can set $\epsilon' = \epsilon/35$. Then, $\frac{2(1+\epsilon')}{(1/2-2\epsilon')(1-\epsilon')} \cdot \mathrm{ed}(\sigma_{[i-w+1,i]}) \leq (4+\epsilon) \cdot \mathrm{ed}(\sigma_{[i-w+1,i]})$. The total space usage of the two data structures is $O(\frac{1}{\epsilon^2}\log^2(\epsilon w) + \frac{1}{\epsilon}\log w \log(\epsilon w))$. If $\epsilon > \frac{1}{w}$, $\log w = O(\frac{1}{\epsilon}\log(\epsilon w))$ and thus the total space usage is $O(\frac{1}{\epsilon^2}\log^2(\epsilon w))$. Otherwise, we can store all items in the window, which only requires $O(w) = O(\frac{1}{\epsilon})$ space.

Improving the running time. The per-item update time of the algorithm is $O(w)$ because the algorithm checks the interval $I = [k, i-1]$ for every length $|I| \in [w-1]$. An observation in [8] is that an $\frac{\epsilon'}{2}$-approximate ϕ-quantile of an interval with length $|I|$ is also an ϵ'-approximate ϕ-quantile for all intervals with length $|I|+1, \cdots, (1+\frac{\epsilon'}{2})|I|$. Hence we only need to check $O(\frac{1}{\epsilon'}\log w)$ intervals of length $1, 2, \cdots, (1+\frac{\epsilon'}{2})^i, (1+\frac{\epsilon'}{2})^{i+1}, \cdots, w$. Then we obtain an ϵ'-approximate quantile for every interval. Note that the query time for returning an approximate quantile is $O(\frac{1}{\epsilon'}\log^2 w)$, and the per-item update time of the two data structures is $O(\frac{1}{\epsilon^2}\log^3 w)$ [6,13]. We conclude with the main result of this section.

Theorem 1. *There is a deterministic $(4+\epsilon)$-approximate algorithm for estimating ED in a sliding window of the latest w items. The space usage is $O(\frac{1}{\epsilon^2}\log^2(\epsilon w))$ and the per-item update time is $O(\frac{1}{\epsilon^2}\log^3 w)$.*

Remark. For the whole stream model, the state-of-the-art result is a $(2+\epsilon)$-approximation in [8]. They gave an improved estimator $R(i)$ as the set of indices j such that there exists $k < j$ with $|[k, j-1] \cap inv(j)| > |[k, j-1] \cap R(i)|$. In other words, whether an index belongs to $R(i)$ or not depends on the number of members of $R(i)$ before that index. Note that a member of $R(i)$ could become a nonmember due to window expiration. Therefore, an index j that is not a member of $R(i)$ initially, may later become a member if some of the previous $R(i)$ members become nonmembers. This makes estimating this improved $R(i)$ difficult in the sliding window model.

4   Lower Bounds for Out-of-Order Streams

In this section, we consider an out-of-order stream σ consisting of a sequence of items $\sigma(i) = \langle t_i, v_i \rangle$ for $i \in [N]$, where t_i and v_i are the timestamp and value of the i-th item, respectively. Recall that the sortedness of the stream is measured on the derived value sequence by rearranging the items in non-decreasing order of the timestamps. We show that even for the whole data stream model, any randomized constant-approximate algorithm for estimating ED or LIS requires $\Omega(N)$ space. In fact, a stronger lower bound holds for ED: any randomized algorithm that decides whether ED equals 0 uses $\Omega(N)$ space. Our proofs follow from reductions from two different communication problems.

4.1   Estimating ED in an Out-of-Order Stream

Theorem 2. *Consider an out-of-order stream σ of size N. Any randomized algorithm that distinguish between the cases that $\text{ed}(\sigma) = 0$ and that $\text{ed}(\sigma) \geq 1$ must use $\Omega(N)$ bits. Therefore, for arbitrary constant $r \geq 1$, any randomized r-approximation to $\text{ed}(\sigma)$ requires $\Omega(N)$ bits.*

We prove the above lower bound by showing a reduction from the classical communication problem INDEX, which has strong communication lower bound.

The problem $\text{INDEX}(x, i)$ is a two-player one-way communication game. Alice holds a binary string $x \in \{0, 1\}^n$ and Bob holds an index $i \in [n]$. In this communication game, Alice sends one message to Bob and Bob is required to output the i-th bit of x, i.e. x_i, based on the message received. A trivial protocol is for Alice to send all her input string x to Bob, which has communication complexity of n bits. It turns out that this protocol is optimal. Particularly, Alice must communicate $\Omega(n)$ bits in any randomized protocol for INDEX [1].

Proof (of Theorem 2). Given an out-of-order stream with length N, suppose there is a randomized algorithm $\mathcal{A}$ that can determine whether its ED equals to 0 or is at least 1 using S memory bits. We define a randomized protocol $\mathcal{P}$ for $\text{INDEX}(x, i)$ for $n = N - 1$: Alice constructs (hypothetically) an out-of-order stream σ with length n by setting

$$\sigma(j) = \begin{cases} \langle 2j - 1, 3j - 2 \rangle, & \text{if } x_j = 0 \\ \langle 2j - 1, 3j \rangle, & \text{if } x_j = 1 \end{cases} \tag{1}$$

Alice then simulates algorithm $\mathcal{A}$ on stream σ and sends the content of the working memory to Bob. Bob constructs another stream item $\sigma(n+1) = \langle 2i, 3i - 1 \rangle$ to continue running algorithm $\mathcal{A}$ on it and obtains the output. If the output says $\text{ed}(\sigma) = 0$, Bob outputs 0; otherwise, Bob outputs 1.

It is not hard to see that $\text{INDEX}(x, i) = x_i = 0$ implies $\text{ed}(\sigma) = 0$ and $\text{INDEX}(x, i) = 1$ implies $\text{ed}(\sigma) = 1$. Therefore, if algorithm $\mathcal{A}$ reports the correct answer with high probability, the protocol $\mathcal{P}$ outputs correctly with high probability, and thus is a valid randomized protocol for INDEX. In the protocol, the number of bits communicated by Alice is at most S. Combining the $\Omega(n) = \Omega(N)$ lower bound, we obtain that $S = \Omega(N)$, completing the proof. $\square$

4.2   Estimating LIS in an Out-of-Order Stream

Theorem 3. *Consider an out-of-order stream σ with size N. Any randomized algorithm that outputs an r-approximation on $\text{lis}(\sigma)$ must use $\Omega(N/r^2)$ bits.*

Proof. We prove the lower bound by considering the t-party set disjointness problem DISJ. The input to this communication game is a binary $t \times \ell$ matrix $\mathbf{x} \in \{0, 1\}^{t\ell}$, and each player P_i holds one row of $\mathbf{x}$, the 1-entries of which

indicate a subset A_i of $[\ell]$. The input $\mathbf{x}$ is called *disjoint* if the t subsets are pairwise disjoint, i.e., each column of $\mathbf{x}$ contains at most one 1-entry; and it is called *uniquely intersecting* if the subsets A_i share a unique common element y and the sets $A_i - \{y\}$ are pairwise disjoint, meaning that in $\mathbf{x}$, except one column with entries all equal to 1, all the other columns have at most one 1-entry. The objective of the game is to distinguish between the two types of inputs. To obtain the space lower bound, we only need to consider a restricted version of DISJ where, according to some probabilistic protocol, the first $t - 1$ players in turn send a message privately to his next neighboring player and the last player P_t outputs the answer.

An optimal lower bound of $\Omega(\ell/t)$ total communication is known for DISJ even for general randomized protocols (with constant success probability) [14], and thus the lower bound also holds for our restrited one-way private communication model. By giving a reduction and setting the parameters appropriately, we can obtain the space lower bound.

Given a randomized algorithm that outputs r-approximation to the LIS of any out-of-order stream with length N, using S memory bits, we define a simple randomized protocol for DISJ for $t = 2r = o(N)$ and $\ell = N + 1 - t = \Theta(N)$. Let $\mathbf{x}$ be the input $t \times \ell$ matrix. The first player P_1 creates an out-of-order stream σ by going through his row of input $R_1(\mathbf{x})$ and inserting a new item $\langle (j - 1)t + 1, (\ell - j)t + 1 \rangle$ to the end of the stream whenever an entry x_{1j} equals to 1. He then runs the streaming algorithm on σ and sends the content of the memory to the second player. In general, player P_i appends a new item $\langle (j - 1)t + i, (\ell - j)t + i \rangle$ to the stream for each nonzero entry x_{ij}, simulates the streaming algorithm and communicates the updated memory state to the next player. Finally, player P_t obtains the approximated LIS of stream σ. If it is at most r he reports that the input $\mathbf{x}$ is disjoint; else, he reports it is uniquely intersecting. It's easy to verify that if the input $\mathbf{x}$ is disjoint, the correct LIS of stream σ is 1, while if it is uniquely intersecting, the correct LIS of σ is t. Consequently, if the streaming algorithm outputs an r-approximation to $\text{lis}(\sigma)$ with probability at least $2/3$, the protocol for DISJ is correct with constant probability, using total communication at most $(t - 1)S$. Following the lower bound for DISJ, this implies $(t - 1)S \geq \Omega(\ell/t)$, i.e., $S = \Omega(\ell/t^2) = \Omega(N/r^2)$. Theorem 3 follows.

Remark. Actually, for deterministic algorithms, we can obtain a slightly stronger lower bound of $\Omega(N/r)$ for r-approximation, by a reduction from the HIDDEN-IS problem used in [10] to prove the $\Omega(\sqrt{N})$ lower bound for approximating LIS of an in-order stream. The reduction is similar to the above, and if we set the approximation ratio r to a constant, the lower bounds become linear in both cases. Therefore, we neglect the details here.

References

1. Ablayev, F.: Lower bounds for one-way probabilistic communication complexity and their application to space complexity. Theoretical Computer Science 157(2), 139–159 (1996)
2. Ajtai, M., Jayram, T.S., Kumar, R., Sivakumar, D.: Approximate counting of inversions in a data stream. In: Proc. STOC, pp. 370–379 (2002)
3. Ben-Moshe, S., Kanza, Y., Fischer, E., Matsliah, A., Fischer, M., Staelin, C.: Detecting and exploiting near-sortedness for efficient relational query evaluation. In: Proc. ICDT, pp. 256–267 (2011)
4. Brin, S., Page, L.: The anatomy of a large-scale hypertextual web search engine. Computer Networks 30(1-7), 107–117 (1998)
5. Chakrabarti, A.: A note on randomized streaming space bounds for the longest increasing subsequence problem. In: ECCC, p. 100 (2010)
6. Cormode, G., Korn, F., Tirthapura, S.: Time-decaying aggregates in out-of-order streams. In: Proc. PODS, pp. 89–98 (2008)
7. Cormode, G., Muthukrishnan, S.M., Şahinalp, S.C.: Permutation Editing and Matching via Embeddings. In: Yu, Y., Spirakis, P.G., van Leeuwen, J. (eds.) ICALP 2001. LNCS, vol. 2076, pp. 481–492. Springer, Heidelberg (2001)
8. Ergun, F., Jowhari, H.: On distance to monotonicity and longest increasing subsequence of a data stream. In: Proc. SODA, pp. 730–736 (2008)
9. Estivill-Castro, V., Wood, D.: A survey of adaptive sorting algorithms. ACM Computing Surveys 24, 441–476 (1992)
10. Gál, A., Gopalan, P.: Lower bounds on streaming algorithms for approximating the length of the longest increasing subsequence. In: Proc. FOCS, pp. 294–304 (2007)
11. Gopalan, P., Jayram, T.S., Krauthgamer, R., Kumar, R.: Estimating the sortedness of a data stream. In: Proc. SODA, pp. 318–327 (2007)
12. Gopalan, P., Krauthgamer, R., Thathachar, J.: Method of obtaining data samples from a data stream and of estimating the sortedness of the data stream based on the samples. United States Patent 7,797,326 B2 (2010)
13. Lin, X., Lu, H., Xu, J., Yu, J.X.: Continuously maintaining quantile summaries of the most recent n elements over a data stream. In: Proc. ICDE, pp. 362–374 (2004)
14. Jayram, T.S.: Hellinger Strikes Back: A Note on the Multi-party Information Complexity of AND. In: Dinur, I., Jansen, K., Naor, J., Rolim, J. (eds.) RANDOM 2009. LNCS, vol. 5687, pp. 562–573. Springer, Heidelberg (2009)