TRANSACTIONS ON DATA PRIVACY 9 (2016) 73–100

On Differentially Private Longest Increasing Subsequence Computation in Data Stream

Luca Bonomi[1,*], **Li Xiong**[2]

[1]University of California San Diego, La Jolla, CA.

[2]Emory University, Atlanta, GA.

E-mail: lbonomi@ucsd.edu, lxiong@emory.edu

Revised and extended version of a paper selected from PAIS 2015

Abstract. Many important applications require a continuous computation of statistics over data streams. Activities monitoring, surveillance and fraud detections are some settings where it is crucial for the monitoring applications to protect user's sensitive information in addition to efficiently compute the required statistics. In the last two decades, a broad range of techniques for time-series and stream data monitoring has been developed to provide provable privacy guarantees employing the formal notion of differential privacy. Although these solutions are well established, they are mostly limited to count based statistics (e.g. number of distinct elements, heavy hitters) and do not apply in settings where more complex statistics are needed. In this paper, we consider a more general problem of estimating the sortedness of a data stream by privately computing the length of the longest increasing subsequence (LIS). This important statistic can be used to detect surprising trends in time-series data (e.g. finance) and perform approximate string matching in computational biology domains. Our proposed approaches employ the differential privacy notion which provides strong and provable privacy guarantees. Our solutions estimate the length of the LIS using block decomposition and local approximation techniques. We provide a rigorous analysis to bound the approximation error of our algorithms in terms of privacy level and length of the stream. Furthermore, we extend our solutions to computing the length of the LIS over sliding windows and we show the beneficial effects of this formulation on the final utility. An extensive experimental evaluation of our proposed solutions on real-world data streams demonstrates the effectiveness of our approaches for computing accurate statistics and detecting surprising trends.

1 Introduction

Sequential data are central in a broad range of domains and applications, such as biomedical, financial and health-care setting where data are continuously collected for monitoring purpose or for mining behavioral patterns. For example, individual household power consumption data may be collected by smart meters to provide billing information or for monitoring purpose. Despite the importance of these tasks, the release of the real data value may disclose sensitive user information. Therefore privacy preserving solutions are employed to compute the required statistics while providing privacy for users. Among the recent privacy models proposed in literature, the popular notion of *differential privacy* [8] is widely used to construct privacy preserving algorithms becoming the de facto standard of privacy in the database community. In this model, the privacy is achieved

*This paper is based on work conducted while at Emory University.

by bounding the adversary inference ability in determining the presence of any event in the data stream [9, 12, 11, 5]. Despite the strength of such a privacy model, the current solutions are limited to count based statistics.

In this paper, we study the problem of privately computing ordered statistics with the goal of enabling applications to monitor sequential data streams. Consider for example, a user who may wish to be advised in his/her financial decisions without incurring the risk of disclosing his/her financial information. In such a setting, it is crucial to design effective solutions that enable third-party to detect user's financial trends while protecting the sensitive information. To address this problem, we propose to study the privacy preserving computation of the *longest increasing subsequence* (LIS) in the stream model.

The computation of the LIS provides useful information about the sortedness of the data stream and it can be used to detect trends in time-series data. In general, the task of computing the sortedness of a data stream is receiving considerable attention from the computer science community [20, 15, 1, 16, 7, 24, 13]. The sortedness of data stream has important implications from both practical and theoretical perspectives. Many applications rely on ranked data and the massive amount of information dynamically generated cannot be processed in an off-line fashion requiring solutions to have small update time and memory requirements.

The computation of the LIS in the data stream model raises new challenges compared to the traditional privacy setting. First of all, privacy requirements in protecting sensitive information for this ordered statistic have a greater impact on the final utility. Count based statistics over a stream are typically computed by decomposition which leads to a considerable reduction of perturbation noise required by the privacy mechanism. However, ordered statistics are generally not easy to approximate via decomposition since they require a global view of the entire stream. Second, the LIS has higher memory requirements compared to standard count based statistics (e.g. counts of distinct elements, heavy hitters). In fact, it has been shown in [15] that there exists a space lower bound of $\Omega(T)$ for any randomized algorithm that computes the LIS exactly over a stream of length T. This strong separation between count based functions and LIS impacts both efficiency and utility of the solutions for this problem.

To address these challenges, we propose a series of solutions for privately computing the LIS while minimizing the error introduced by the perturbation and approximation. The detailed contributions are reported below.

Our Contributions. In this paper, we study the problem of privately computing the LIS in a time-bounded stream of length T. Our contributions are listed below.

- Our proposed solutions compute the length of the LIS providing strong and provable privacy guarantees based on the notion of differential privacy.
- We propose a decomposition framework for approximating the length of the LIS using local information in the stream. This technique allows us to reduce the error due to perturbation noise from the privacy mechanism. Using the Patience Sorting algorithm [17] as a black box for locally computing the exact length of the LIS, we provide an error bound for our framework.
- We conduct an output-sensitive utility analysis for two cases based on the length of the output LIS. In particular, we assume $LIS(\sigma) = \sqrt{T}/\beta$, where T is the length of the input stream σ, and β is a parameter in the range $[1/\sqrt{T}, \sqrt{T}]$. For each solution, we bound the approximation error in the case of long and short LIS respectively depending on the value of β.
- We propose a new streaming approach which computes the LIS using a hierarchy structure of the stream. Our algorithm achieves a $(1 - \frac{T-b}{T+b})$-approximation to the length of the LIS in the

worst case, where the parameter b controls both the perturbation noise to achieve the desired level of privacy and the accuracy.

- We provide a discussion about possible extensions of our solutions to address time-series stream monitoring tasks. Furthermore, we extend our approaches to compute the length of the LIS over sliding windows and we demonstrate the impact of the privacy mechanism on the final utility in this formulation.

- We perform an extensive experimental evaluation of our proposed techniques on two real data streams and show the effectiveness of our solutions in computing accurate statistics and detecting surprising trends.

The rest of the paper is organized as follows. Section 2 summarizes the most relevant privacy preserving works on data streams that are closely related to our techniques. Section 3 provides the problem definition and presents the privacy model. Section 4 illustrates our decomposition schema and Section 5 describes our hierarchy solution. In Section 6, we extend our approaches over sliding windows, and in Section 7 we provide a summary of our results. Section 8 illustrates our experimental results. In Section 9, we conclude the paper.

2 Related Work

In the last decade, a considerable amount of research has been devoted to designing differentially private techniques to compute statistics over data stream. Here, we provide a brief overview of the most related works to ours and we refer the readers to the cited references for a more detailed review of these results. Below, we distinguish the literature in the stream setting in three categories: *count based statistics*, *aggregated statistics* and *distributed statistics*.

Count Based Statistics. In the streaming setting, the data statistics are computed continuously over the stream, that is, for each new incoming element in the stream a new output is generated. In this setting, the item-level differential privacy is commonly adopted to protect the presence of individual items in the stream. Therefore, any change in the input at the current time stamp will not only affect the current computed statistic but it might also impact the future output. Hence, from a utility perspective, privacy preserving algorithms should be able to carefully calibrate the amount of noise injected such that the final utility is not compromised. For count based statistics, many techniques have been proposed to limit the impact of the privacy mechanism. In the work of Chan et al. [5], a binary stream is considered in input and the query task requires to count the number of 1's seen so far. To reduce the impact of the perturbation mechanism a decomposition of the stream based on binary tree is proposed. The authors showed that with such a decomposition the additive error grows only polylogarithmically with the size of the input stream. Independently, Dwork et al. [11], they also construct a differentially private continual counter with similar error guarantees. Recently, Bolot et al. [4] considered an extension of these aggregated statistics by focusing on monitoring the recent past of the stream. Specifically, the authors proposed two differentially private solutions for continual observations over sliding windows and decayed data. The solutions in [4] use a similar tree decomposition of the stream as in [5, 11] where the nodes contributions in the tree are now weighted according to the decay function. Furthermore, Bolot et al. [4] presented a new notion of decayed privacy, where past data are not viewed as private anymore and, thus, they can be used without incurring privacy cost. Recently, the authors in [22] proposed (pan-)privacy solutions using sketches to compute the count of the heavy hitters. In this work, the authors assume that the counter of an item can be increased or decreased arbitrarily, as long as the counter remains non-negative.

Algorithm 1 Patience Sorting

procedure PATIENCE SORTING(σ)
Input: event stream σ
Output: $LIS(\sigma)$ length of the longest increasing subsequence

$P(j) \leftarrow \emptyset$ for $j = 0, 1, \ldots, m-1$
for (any new element $\sigma(i)$) **do**
  Find the largest $P(j)$ such that $P(j) \leq \sigma(i)$
  $P(j+1) = \sigma(i)$
  Output the largest j such that $P(j) \neq \emptyset$
end for
end procedure

Aggregated Statistics. Apart from this counter based setting, Fan and Xiong [14] considered the problem of reporting aggregated statistics over a finite stream. Specifically, aggregated time-series data from multiple users are considered in input and at each time instant an aggregated count is released with guarantees of user-level differential privacy. The impact of the noise on the final result is mitigated using sampling techniques. Recently, Kellaris et al. [18] proposed the notion of w-event privacy, which protects any event sequence occurring in w successive time instants. Furthermore, they showed how the approach in [14] can be extended using this new privacy notion to handle unbounded streams.

Distributed Statistics. Recently, the research community started to study a fully-distributed formulation of the continuous release of data statistics over data stream. Specifically, the computation of data statistics is performed by an untrusted aggregator which combines multiple data streams generated by individual users. In this scenario, the required data statistics have to be computed with high utility while providing differentially privacy guarantees for the individual users. Rastogi et al. [23] and Shi et al. [25] proposed novel algorithms that allow an untrusted aggregator to periodically estimate the sum of n users' values achieving differential privacy guarantees for the individuals. Furthermore, these solutions provide aggregator obliviousness, that is, the aggregator only learns the noisy sum, but no intermediate results. Recently, Chan et al. [6] improved these approaches by developing solutions that in addition to providing privacy guarantees are also robust to user's failure in the protocol.

3 Preliminaries

Given a sequence σ of elements $\sigma(i) = a_i$ defined over a finite alphabet $\Sigma = \{0, 1, \ldots, N-1\}$, an increasing sequence of length k in σ is a subsequence $\{i_0, i_1, \ldots, i_{k-1}\}$ such that $i_0 < i_1 < \cdots < i_{k-1}$ and $a_{i_0} < a_{i_1} < \cdots < a_{i_{k-1}}$. Furthermore, let $\sigma[i, j]$ denote the contiguous sequence $\sigma(i)\sigma(i+1)\ldots\sigma(j)$ in the stream σ, and let $LIS(\sigma)$ be the length of the longest increasing subsequence in σ.

The problem of computing the LIS has received much attention in the streaming setting (see [3] for a survey of results), where the sequence σ is given an element at a time. In such model, data arrive continuously and at every time i algorithmic solutions are required to report $LIS(\sigma[0, i])$ by using a small amount of memory and performing only few passes over the stream. In the rest of the section, we briefly summarize the non-private techniques present in literature by categorizing them as *exact* and *approximate* solutions.

Exact Solution. The study of LIS in the streaming setting was initiated by Liben-Nowell et al.

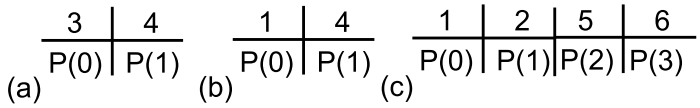


Figure 1: Running example of the Patience Sorting algorithm over the stream $\sigma = 3, 4, 1, 2, 5, 7, 6$.

in [20], where the authors developed an exact one pass algorithm that requires $O(k)$ space for deciding if the length of longest increasing subsequence is at least k. In addition to this technique, the classical algorithm for computing the LIS is based on the Patience Sorting procedure [17]. This approach can be interpreted as a one pass streaming algorithm for computing the exact LIS in $O(T)$ space and it requires $O(\log LIS(\sigma))$ update time where T represents the length of the stream. Since we use this approach to build our solutions, we briefly describe this algorithm here.

In the Patience Sorting procedure, the length of the longest increasing subsequence is computed using a set of sorted *piles* $P(0) < P(1) < \cdots < P(m)$ each storing an element of the stream σ. For any new element $\sigma(i)$ that appears in the stream, the algorithm places $\sigma(i)$ in the leftmost pile $P(j)$ such that $P(j) > \sigma(i)$. The number of non empty piles represents the length of the LIS at any time point. An overview of the Patience Sorting algorithm is illustrated in Algorithm 1. Below, we describe a running example of this algorithm.

Example 1. Let $\sigma = 3, 4, 1, 2, 5, 7, 6$ be a stream in input. The algorithm starts with a set of empty piles $P(j)$, where $j = 0, \ldots, m-1$. When the first element arrives in the stream it is placed in the first pile $P(0)$. After the arrival of the second element, the situation in the piles is illustrated in Figure 1 (a). The number of piles denotes the length of the longest increasing subsequence at each time. Therefore, in this case the length of the LIS is two. When the third element $\sigma(2) = 1$ arrives in the stream, the algorithm places this element in $P(0)$ replacing the element 3 currently presents in the pile, as shown in Figure 1 (b). Following the steps of the algorithm, the final set of piles is reported in Figure 1 (c). At the end of the stream the length of the longest increasing subsequence is four.

Despite the simplicity of this procedure, the Patience Sorting algorithm is optimal from the space complexity perspective. In fact, Gopalan et al. [15] showed a space lower bound of $\Omega(T)$ for any randomized algorithm that computes the LIS exactly.

Approximate Solution. In [15], the authors proposed a $(1+\epsilon)$-approximation for the LIS computation using $O(\sqrt{T/\epsilon})$ space. A series of works have been developed to estimate the length of the LIS using the number of inverted elements in the stream. In this direction, Ajtai et al. [1] proposed a $(1+\epsilon)$-approximation which requires $O(\frac{1}{\epsilon}\log\log T)$ space to estimate the number of inverted pairs. Later this result has been improved by Gupta and Zane [16]. Cormode et al. [7] proposed a series of algorithmic solutions based on distance preserving embeddings. Recently in [24], the authors investigated the problem of computing the LIS in asymmetric edit distance setting.

3.1 Differential Privacy

Differential privacy [8] is a recent notion of privacy that aims to protect the disclosure of information when data statistics are released. In the streaming setting, due to the dynamics of the data, the classical differential privacy notion has been redefined such that the privacy is guaranteed at *event-level* [9, 12, 11, 5]. In other words, the privacy goal is to protect the presence or absence of any single event in the stream. The formal definition of the differential privacy notion adopted in our work is reported below.

Definition 2 (Differential Privacy [5, 11])**.** Two streams σ and σ' of the same length are neighboring streams if they differ exactly in one element at time t. A privacy mechanism M gives α-differential privacy if for any two neighboring streams σ, σ', and for any set of outcomes $S \subseteq Range(M)$,

$$Pr[M(\sigma) \in S] \leq e^{\alpha} \times Pr[M(\sigma') \in S] \tag{1}$$

The parameter α is called the *privacy parameter* and it defines the privacy level of the mechanism. Higher values of α lead to lower level of privacy, while smaller values pose a stronger privacy guarantee. Intuitively, a mechanism is differentially private if an adversary is unable to determine whether an event of interest took place or not by observing the output of the mechanism over the stream.

Our goal consists in designing a mechanism that, at any time t in the stream, reports the length of the longest increasing subsequence while achieving differential privacy. In addition, we would like the mechanism to be useful, that is, its output well approximates the real length of the LIS. To evaluate our solutions, we introduce the following utility notion.

Definition 3 ((ϵ, δ)-Useful)**.** A streaming algorithm $\mathcal{A}$ is (ϵ, δ)-useful, if for any input stream σ and query q, with probability at least $1 - \delta$, the relative distance between the approximate answer from $\mathcal{A}$ and the real answer of q is within ϵ, formally $P\left[\frac{\|\mathcal{A}(\sigma) - q(\sigma)\|}{q(\sigma)} < \epsilon\right] \geq 1 - \delta$

To achieve differential privacy, one well established technique is the *Laplace Mechanism* [10]. Dwork et al. [10] showed that to obtain a α-differentially private solution it is sufficient to perturb the real output of the function by adding a random variable (noise) from a Laplace distribution with probability density function $pdf(x|\lambda) = \frac{1}{2\lambda} e^{-|x|/\lambda}$, where the parameter λ is determined by α and the *sensitivity* of the function to compute. The sensitivity measures the contribution of any single element on the final output. We will use the Laplace mechanism and some other statistical tools to design our privacy preserving solutions.

Statistical tools. In our approaches, we make use of the *Laplace Mechanism* to achieve differential privacy and *sequential composition* property of differential privacy. Furthermore, in our construction the noise may not come from a single Laplace distribution, but rather is composed by a sum of independent Laplace distributions. Therefore, here we state two useful results for sum of independent Laplace distributions.

Theorem 4 (Laplace Mechanism [10])**.** *For a function $f : D^T \rightarrow \mathbb{R}^d$, let $GS(f)$ be the sensitivity of f defined as*

$$GS(f) = \max_{D, D'} \|f(D) - f(D')\|_1 \tag{2}$$

where D' and D are neighbouring, then the algorithm that outputs: $\tilde{f}(D) = f(D) + Lap(GS(f)/\alpha)^d$ satisfies α-differential privacy.

Theorem 5 (Sequential Composition [21])**.** *Let M_i be a non-interactive privacy mechanism which provides α_i-differential privacy. Then, a sequence of $M_i(D)$ over the database D provides $(\sum_i \alpha_i)$-differential privacy.*

Lemma 6 (Sum Of Laplace Distributions [5])**.** *Let $Y = \sum_{i=1}^{n} l_i$ be the sum of $l_1, \ldots, l_n$ independent Laplace random variables with zero mean and parameter b_i for $i = 1, \ldots, n$, and $b_{max} = \max\{b_i\}$. Let $\nu \geq \sqrt{\sum_{i=1}^{n} b_i^2}$, and $0 < \lambda < \frac{2\nu^2}{b_{max}}$. Then $Pr[Y > \lambda] \leq exp\{-\frac{\lambda^2}{8\nu^2}\}$*

Corollary 7 (Measure Concentration [5])**.** *Let Y, $\{b_i\}_i$, λ and b_{max} defined as in Lemma 6. Suppose $0 < \delta < 1$ and $\nu > \max\{\sqrt{\sum_i b_i^2}, b_{max}\sqrt{2 \ln \frac{2}{\delta}}\}$. Then $Pr[|Y| > \nu\sqrt{8 \ln \frac{2}{\delta}}] \leq \delta$*

3.2 Differentially Private Computation of the LIS - A Baseline Approach

In the rest of the paper, we present our solutions for computing the length of the LIS in the stream. Our approaches require the stream to be *time-bounded*, we assume in fact that the length of the stream is T and it is given a priori.

Here we consider a baseline approach that solves the problem of privately computing the length LIS by perturbing directly its real value at every time point. In particular, for every new element $\sigma(i) = a_i$ in the stream, the algorithm first computes the real $LIS(\sigma[0,i])$ (e.g. using any non-private solution, Patience Sorting in this case) and then it adds a perturbation noise η_i. Given the privacy parameter α, due to the composition property of differential privacy, to obtain an overall mechanism of α-differential privacy, the baseline approach applies the Laplace mechanism at each time point with parameter $\alpha' = \alpha/T$. For each new incoming element, it samples a Laplace variable $\eta_i \sim Lap(1/\alpha')$ which will be used to perturb the real value of $LIS(\sigma[0,i])$. Therefore, at every time i, the algorithm will answer the LIS query by returning $\tilde{l}(\sigma[0,i]) = LIS(\sigma[0,i]) + \eta_i$. We can observe that the sensitivity for the LIS function is 1, since replacing an element from the stream may change the length of the longest increasing subsequence by at most 1. Therefore, perturbing the real value of $LIS(\sigma[0,i])$ with η_i is sufficient to achieve privacy. The utility of this approach is reported in the following theorem.

Theorem 8 (Baseline utility)**.** *The baseline algorithm is* $(\frac{\beta\sqrt{T}}{\alpha}\ln\frac{1}{\delta}, \delta)$*-useful for computing the longest increasing subsequence.*

Proof. The released length of the LIS at each time i is obtained by perturbing the real length of the LIS with Laplace noise. Therefore, at every time step in the stream we have that the additive error from the noise can be bounded as follows:

$$Pr[|\eta_i| > \gamma] \leq 2\int_{\gamma}^{\infty} \frac{\alpha}{2T} e^{-x\alpha/T} dx = e^{-\gamma\alpha/T} \tag{3}$$

Hence, with probability at most δ the additive error is at least $\frac{T}{\alpha}\ln\frac{1}{\delta}$. The final result follows by normalizing the error by the $LIS(\sigma) = \sqrt{T}/\beta$. □

Space and Time Analysis. The memory and time complexity for this approach are the same as the non-private algorithm used to compute the real length of the LIS. Therefore, using the Patience Sorting algorithm for example, the space and update time required are $O(LIS(\sigma))$ and $O(\log LIS(\sigma))$ respectively.

4 Decomposition Framework

The baseline approach introduces an additive error that grows linearly with the length of the stream. Therefore, for small LIS this error could dramatically degenerate the utility of this solution. The reason for this large perturbation noise is due to the fact that each individual element in the stream could affect all the possible outputs of the algorithm over the entire stream. This phenomenon could also occur for more sophisticated streaming algorithms that compute the LIS by using a small sketch of the stream ([15, 24] for example). Although such solutions could reduce the space requirements, the use of a sketch does not directly reduce the error due to the perturbation noise since an element of the stream could still affect a large number of outputs (e.g. linear with the length of stream).

To overcome this problem, we decompose the computation of the LIS over segments of the stream. This intuition follows the idea proposed by Chan et al. [5] where a linear and binary decomposition frameworks are employed to privately compute the number of non-zero elements in a binary

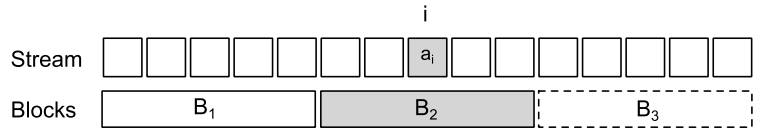


Figure 2: Block Decomposition example at time i: expired blocks (solid lines), active blocks (gray) and the future blocks (dashed lines).

stream. Despite the similarity in these decompositions, the computation of the longest increasing subsequence presents new challenges due to the block composition. For this reason, we study the utility loss in approximating the LIS inflicted by using the local information of the stream. In this work, we consider only an extension of the binary decomposition since it provides better utility with respect to the linear decomposition proposed in the original paper [5].

In our work, we investigate the implications of decomposing the LIS computation over blocks (i.e. stream segments) both from the utility and complexity perspective. It is important to note that the nature of the decomposition should be data-independent to avoid additional privacy cost. In principle, any algorithm $\mathcal{A}_{LIS}$ that computes the LIS (either exact or approximate way) can be used as a building block to compute the LIS on each stream segment so that the perturbation noise required by the privacy mechanism can be reduced with respect to the direct use of $\mathcal{A}_{LIS}$. On the other hand, by limiting our computation on segments we introduce an approximation error.

In the rest of the section, we use the Patience Sorting algorithm [17] as a simple building block. We focus on this particular algorithm because it allows us to have an internal procedure that computes the exact length of the LIS over segments of the stream. In this way, we can directly measure how our decomposition impacts the exact solution. Since the original Patience Sorting algorithm computes not only the length of the LIS but also the elements forming the sequence, we use a modified version that only keeps the top element of the piles in the data structure as illustrated in the Algorithm 1. In this way, we can compute the length of the LIS but using only $O(LIS(\sigma))$ space.

Before presenting our technique, we illustrate some concepts that will be useful in explaining our algorithm. A block $B = \sigma[j, j + b - 1]$ of size b represents a continuous segment of b symbols in the stream σ. Due to the dynamics of the data in the stream, a block assumes three different states over the stream depending on the current time. At time i, the block B can be in one of the following states: **expired** hence the new arrival does not affect the block B (i.e. $j + b - 1 < i$), **active** when the new arrival is contained in the block B (i.e. $j \leq i \leq j + b - 1$) and **future** hence B contains only upcoming elements (i.e. $j > i$). An example of block decomposition of the stream is illustrated in Figure 2. The life cycle of a block B consists of starting as a future block, becoming active, and finally the block expiration.

4.1 Binary Decomposition

We start observing that in general the decomposition of the LIS over blocks may incur large approximation error. In fact, by simply dividing the stream into blocks and combining the length of their LIS as an answer could lead to an approximation error that is proportional to the number of blocks used in the decomposition. To reduce this error, we develop a stream decomposition into variable length blocks, where the number of blocks is bounded by $O(\log T)$. We organize the blocks in a binary tree where at time i the tree has $\log i$ levels. Each level $l = 0, \ldots, \log i$ in the tree partitions the stream into disjoint blocks of length $i/2^l$. Figure 3 illustrates an example of binary decomposition of the stream.

Using this representation, each node k in the tree is associated with a block B_k and it stores the

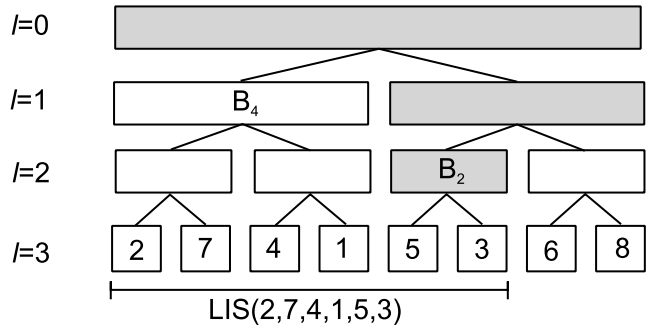


Figure 3: Binary Decomposition example. At time 5 (six symbols), the algorithm updates the active blocks (in gray). It computes the answer to the LIS query by summing the contributions of B_2 and B_4 containing the 2 and 4 most recent symbols respectively.

perturbed value of the $LIS(B_k)$. At any time i the algorithm updates the noisy LIS of the active blocks in the binary tree, and it answers the query $LIS(\sigma[0,i])$ as illustrated in Algorithm 2.

Algorithm Description. In the loop at lines 3-5, the algorithm updates the piles for the active blocks associated with the time i. In particular, the procedure `Update Piles` implements the Patience Sorting algorithm as in Algorithm 1, where in this case the update is performed independently on each active block B for any new coming element $\sigma(i)$. At lines 6-9, the noisy length of the LIS for each block that will expire is computed. At line 10, we compute the binary representation of $i+1$ and let $i_1 < i_2 < \cdots < i_m$ be the positions of non-zeros bits in such representation. Then the answer for $LIS(\sigma[0,i])$ is computed by summing up the length of the LIS for the blocks containing the most recent $2^{i_1}, 2^{i_2}, \ldots, 2^{i_m}$ elements respectively. Therefore at each time i, the output result is obtained by adding the contributions of at most $\Theta(\log i)$ blocks in the loop at lines 13-17.

Privacy Analysis. We can observe that each element affects at most $\log T$ blocks; therefore, perturbing the LIS of each block with a random variable from $Lap(\log T/\alpha)$ is sufficient to satisfy the privacy requirement.

Theorem 9 (Binary Decomposition Privacy)**.** *The Binary Decomposition achieves α-differential privacy.*

Proof. In this decomposition, each element $\sigma(i)$ participates in the LIS of at most $\log T$ active blocks. Therefore, for any two neighboring streams the difference in L_1-norm of their outputs can be bounded by $\log T$. Therefore using Theorem 4, it is sufficient to add to each LIS of each block a random variable from a Laplace distribution with parameter $\log T/\alpha$ to satisfy the privacy requirement. □

Utility Analysis. This decomposition with variable length blocks allows us to reduce the perturbation error due to the privacy mechanism. However, in this way we introduce an approximation error that depends on the number of blocks. We can observe that at most $O(\log T)$ blocks of variable length are needed to answer a LIS query. The utility results for this decomposition are reported below.

Lemma 10 (Binary Block Error Bound)**.** *Let σ be a stream of T symbols, and let $LIS(\sigma) = \frac{\sqrt{T}}{\beta}$, where β is positive. Without loss of generality we assume $T = 2^t - 1$, and we consider a partition of the stream σ into $B_0, B_1, \ldots, B_{t-1}$ non-overlapping blocks, where each block B_k is of size 2^k. Then*

Algorithm 2 Binary Decomposition

procedure BINARY DECOMPOSITION(T, α, σ)
Input: upper bound on the stream length T; privacy parameter α; event stream σ
Output: $\tilde{l}(\sigma)$ released longest increasing subsequence

  for ($i = 0, 1, \ldots, T-1$) **do**
    for (every active B at time i) **do**
      UPDATE PILES($B, \sigma(i)$)
    end for
    for (every block B that will expire at time $i+1$) **do**
      $LIS(B) \leftarrow$ number of piles for the block B
      $\tilde{l}(B) \leftarrow LIS(B) + Lap(2 \log T/\alpha)$
    end for
    Let $i_1 < i_2 < \cdots < i_m$ be the positions of non-zeros in the binary representation of $i+1$
    $\tilde{l}(\sigma) \leftarrow 0$
    $k \leftarrow i$
    for ($j = i_1, i_2, \ldots, i_m$) **do**
      $B \leftarrow \sigma(k - 2^j + 1) \cdots \sigma(k)$ ▷ Retrieve the block to reconstruct the LIS
      $k \leftarrow k - 2^j$
      $\tilde{l}(\sigma) \leftarrow \tilde{l}(\sigma) + \tilde{l}(B)$ ▷ Sum the noisy contributions of the expired block B
    end for
    Output $\tilde{l}(\sigma)$
  end for
end procedure

in reporting the sum of the longest increasing subsequence in each block, $lis(\sigma) = \sum_{k=0}^{t-1} LIS(B_k)$*, we incur the following approximation error.*

$$LIS(\sigma) \leq lis(\sigma) \leq \begin{cases} \log T \cdot LIS(\sigma) & \beta \geq 1 \\ (1 + \log \beta\sqrt{T}) \cdot LIS(\sigma) & \beta \in [1/\sqrt{T}, 1) \end{cases} \tag{4}$$

Proof. First, we start noticing the following lower-bound $lis(\sigma) \geq LIS(\sigma)$. In fact, the part of the real longest increasing subsequence which is contained in each block is at most the length of the longest increasing subsequence in the stream segment represented by the block. Second, we prove the two cases separately. For short value of $LIS(\sigma)$ ($\beta \geq 1$), we consider the case where each segment in each block is monotonic but none of them can be concatenated to form an increasing sequence in the entire stream. Then, we have that $LIS(\sigma) \geq LIS(B_k)$, for $k = 0, \ldots, t-1$, which leads to $\log T \cdot LIS(\sigma) \geq \sum_{k=0}^{t-1} LIS(B_k) = lis(\sigma)$. For the case of long value of LIS ($\beta \in [1/\sqrt{T}, 1)$), we proceed as follows. Let j be a positive integer such that $2^{j-1} < \sqrt{T}/\beta \leq 2^j$. Therefore, for all the blocks B_k with $k \geq j$ we have that $LIS(B_k) \leq \sqrt{T}/\beta$, otherwise there exists a monotonic sequence which is longer than the longest increasing subsequence, hence we have a contradiction. Furthermore, due to the binary tree decomposition the sum of the length of the LIS for the blocks B_k with $k = 0, \ldots, j-1$ can be bounded as follows.

$$\sum_{k=0}^{j-1} LIS(B_k) \leq \sum_{k=0}^{j-1} 2^k = 2^j - 1 \leq 2\sqrt{T}/\beta \tag{5}$$

Therefore, the reported $lis(\sigma)$ can be upper bounded with the value below.

$$\begin{aligned} lis(\sigma) = \sum_{k=0}^{t-1} LIS(B_k) &\leq \sum_{k=0}^{j-1} LIS(B_k) + \sum_{k=j}^{t-1} LIS(B_k) \\ &\leq 2\sqrt{T}/\beta + (t-j)\sqrt{T}/\beta \\ &\approx \sqrt{T}/\beta(1+\log\beta\sqrt{T}) \end{aligned} \tag{6}$$

This concludes the proof of the Lemma. □

Theorem 11 (Binary Decomposition Utility)**.** *The binary decomposition algorithm for computing the length of the longest increasing subsequence achieves the following utility results.*

$$\begin{cases} ((\log T - 1) + \frac{\beta \log^{3/2} T}{\alpha\sqrt{T}} \ln\frac{1}{\delta}, \delta)\text{-}\textit{useful} & \beta \geq 1 \\ (\log\beta\sqrt{T} + \frac{\beta \log^{3/2} T}{\alpha\sqrt{T}} \ln\frac{1}{\delta}, \delta)\text{-}\textit{useful} & \beta \in [1/\sqrt{T}, 1) \end{cases}$$

Proof. This decomposition has the advantage that the number of blocks combined in estimating the length of the LIS is only logarithmic which leads to an approximation error as shown in Lemma 10. This decomposition introduces a perturbation noise which is a sum of at most $O(\log T)$ i.i.d. Laplace random variables with parameter $O(\log T/\alpha)$. Let $\xi = \sum_k \eta_k$ denote the error due to the sum of the Laplace random variables, we can use the result in Corollary 7 to bound this quantity. Choosing $\nu = \sqrt{\sum_k \frac{\log T}{\alpha}}\sqrt{2\ln\frac{2}{\delta}}$ with probability at least $1-\delta$, the quantity ξ is at most $O(\frac{\log^{3/2} T}{\alpha}\ln\frac{2}{\delta})$. Therefore, the final utility follows using the results from Lemma 10 and by normalizing this value by the $LIS(\sigma)$. □

Space Analysis. The space requirement for this approach is related to the number of active blocks that need to be updated and to the space complexity of the internal procedure. Due to the nature of the binary decomposition at any time i there are $\Theta(\log T)$ blocks that are active. Using a similar argument as in Theorem 11, we can show that the space complexity is $O(LIS(\sigma)\ln(\beta^2 LIS(\sigma)))$.

Theorem 12 (Binary Decomposition Space Comp.)**.** *Let $LIS(\sigma)$ be the length of the longest increasing subsequence in the stream σ, then the Binary decomposition framework has space complexity $O(LIS(\sigma)\ln(\beta^2 LIS(\sigma)))$.*

Proof. We begin by recalling that the internal procedure for computing the length of the LIS is the Patience Sort algorithm, where we keep only the top of the piles. At any time i in the stream, $\log T$ blocks are active, one in each level of the tree structure. Furthermore, let j be a positive integer such that $2^{j-1} < LIS(\sigma) \leq 2^j$. Therefore for the blocks in any level $i > j$ in the tree, we can upper bound their space requirements with $LIS(\sigma)(\log T - j + 1)$, since $LIS(\sigma)$ is the current length of the longest increasing subsequence. On the other hand, due to the nature of the binary tree the space required by the blocks below the level i is $2^j - 1$. Therefore the space complexity for this approach is $O(LIS(\sigma)(\log T - j + 1))$. Using the notion that $LIS(\sigma) = \sqrt{T}/\beta$ and $j = \log(LIS(\sigma))$, the previous requirements can be rewritten as $O(LIS(\sigma)\ln(\beta^2 LIS(\sigma)))$. □

Time Analysis. The total update time for this solution is related to the updates of the active blocks. Since at every time i there are $\Theta(\log T)$ active blocks, the update time is $O(\log T \log LIS(\sigma))$ using Patience Sorting algorithm.

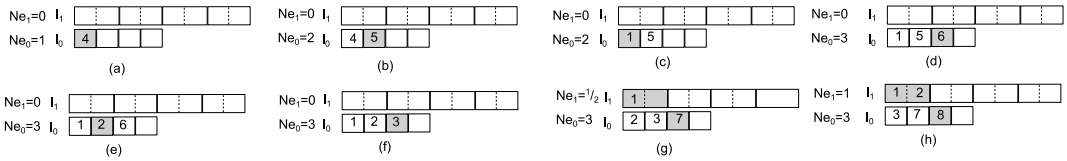


Figure 4: Running example of the Hierarchy mechanism on the input stream $4, 5, 1, 6, 2, 3, 7, 8$, $b = 4$ and $m = 2$.

5 Hierarchy Mechanism

In the previous section, we showed that the binary decomposition considerably reduces the perturbation noise in the final output compared to the baseline approach. However, such technique suffers from the fact that the computation of the LIS generally does not decompose well and in some cases this could lead to a large approximation error. To overcome this problem, we propose a new algorithm which computes the LIS over the stream by simulating the behavior of the Patience Sorting algorithm. In contrast to our previous approaches, this solution computes the length of the LIS by smoothing the impact of each element with the purpose of reducing the perturbation noise while achieving a good approximation ratio.

The main idea is to reduce the impact of those elements that stay too long in the LIS so that the total noise required by the privacy mechanism is decreased. Given an integer $b > 0$, we construct a series of $m = \Theta(\ln \frac{T}{b})$ layers $l_0, l_1, \ldots, l_{m-1}$ with b buckets each, where at layer i each bucket contains 2^i elements. Given the series of elements with index $\{1, 2, \ldots, T\}$ in the stream, each layer simulates the behavior of the Patience Sorting algorithm where in this case the original piles are replaced with buckets that can contain multiple elements. In fact, at layer i the elements in the range $[(j-1)2^i + 1, j2^i]$ can be placed into the same bucket j. Intuitively, each layer has a different granularity, in fact l_0 keeps the exact top elements in the most recent b piles in the Patience Sorting algorithm, while l_1 keeps an approximation of the next $2b$ piles and so forth for the other layers. As the original algorithm, our procedure computes the length of the LIS by counting the number on non-empty buckets. In our case multiple elements may fall in the same bucket; therefore, we use a scaling factor equal to the length of the bucket to compute the contribution of each layer. Furthermore, in addition to insertion and replacement moves allowed in the Patience Sorting algorithm, we introduce an expiration move that forces elements that stay in a bucket at layer l_i for more than $2^i b$ iterations to be moved up to layer l_{i+1}. The algorithm computes the length of the LIS in the stream by adding the contribution at each layer. The code for this procedure is reported in Algorithm 3.

Algorithm Description. The algorithm starts initializing a set of m layers containing b buckets each, at lines 2-3. Within a layer i, each bucket is denoted with $P_i(j)$, for $j = 1, \ldots, b$ and it has size $2^i b$. In the main loop, lines 4-21, each new element coming in the stream is inserted in the first layer using the same rule as the Patience Sorting algorithm, lines 5-7. In the inner loop at lines 9-13, the algorithm checks layer by layer to find the expired elements. When an expired element p in a pile $P_i(j)$ is found, the algorithm removes p and inserts it in the next layer. At line 14, the number of non-empty buckets for each layer is computed by normalizing the number of elements within each bucket with the corresponding bucket's size. In the loop at lines 16-19, the perturbation noise is applied to each count and finally the length of the LIS is returned. We illustrate our hierarchy mechanism in the example below.

Example 13. Consider the situation in Figure 4. When the first element arrives in the stream it is placed in the first bucket at l_0 as shown in (a). The second element that arrives is 5, since it is larger than 4 it is placed in the next bucket (b). The third element in the stream is 1. Since the insertion

Algorithm 3 Hierarchy Mechanism

procedure HIERARCHY MECHANISM(T, α, σ, b)
Input: upper bound on the stream length T; privacy parameter α; event stream σ; accuracy parameter b
Output: $\tilde{l}(\sigma)$ released longest increasing subsequence

$m = \Theta(\ln \frac{T}{b})$
Initialize each layer $l_i = [P_i(1), \ldots, P_i(b)]$ $i = 0, \ldots, m-1$ with b empty buckets
for $(i = 0, 1, \ldots, T-1)$ **do**
  Insert $\sigma(i)$ in l_1
  Find the largest $P_1(j)$ such that $P_1(j) \leq \sigma(i)$
  $P_1(j+1) = \sigma(i)$
  for $(i = 0, \ldots, m-1)$ **do**
    Let p be the element that expires at l_i
    Remove p from l_i and insert it in l_{i+1}
    Find the largest element in $P_{i+1}(j)$ such that $P_{i+1}(j) \leq p$
    $P_{i+1}(j+1) = p$
  end for
  Let Ne_i be the number of non-empty buckets at layer l_i
  $\tilde{l}(\sigma) \leftarrow 0$
  for $(i = 0, 1, \ldots, m-1)$ **do**
    $\widehat{Ne_i} \leftarrow Ne_i + Lap(mb/\alpha)$
    $\tilde{l}(\sigma) \leftarrow \tilde{l}(\sigma) + \widehat{Ne_i}$ ▷ Sum the noisy contribution of each layer
  end for
  Output $\tilde{l}(\sigma)$
end for
end procedure

of the elements in the buckets follows the same rules as the Patience Sorting algorithm, we find the bucket that contains the smallest element larger than 1 and insert this element in that bucket. Therefore, in our case, the element 1 overwrites the element 4 in the first bucket (c). At this point the length of the LIS is 2, as represented by the number of non empty buckets in l_0. The algorithm proceeds in a similar manner for the next three incoming elements (d),(e) and (f). After these new elements are processed, the element 1 in l_0 is moved up to l_1 since it has been in l_0 for more than b steps and the new incoming element 7 is inserted in l_0 (g). In the next step, the element 2 is moved up, and it is inserted in the same bucket with the element 1. At the same time the new element 8 is inserted in l_0 (e). The reported length of the LIS is obtained by summing the contribution of each layer. Layer l_0 contributes with $Ne_0 = 3$ and l_1 contributes with $Ne_1 = 1$. Hence the algorithm reports a length of the LIS of 4 while the exact length is 5.

Privacy Analysis. In this algorithm the contribution of each element on the LIS is progressively decreased according to the layer in which the element appears. The privacy result for our hierarchy mechanism is reported in the following theorem.

Theorem 14 (Hierarchy Mechanism Privacy)**.** *The Hierarchy Mechanism achieves α-differential privacy.*

Proof. Given any two neighboring streams, we can observe that each element can affect at most m layers over the entire stream. In particular, at l_0 an element contributes to the LIS with a factor 1 for b times, at l_1 contributes with factor 1/2 for $2b$ and at the general level l_i contributes with factor $1/2^i$ for $2^i b$ times. Let $\mathbf{Ne}$ be the vector of contributions for each layer for the input stream

$\sigma = a_1, \ldots, a_i \ldots, a_T$. Then, $\forall i \in [1, T]$ and $\sigma' = a_1, \ldots, a'_i \ldots, a_T$ we have that

$$\|\mathbf{Ne}(\sigma) - \mathbf{Ne}(\sigma')\| \leq mb \tag{7}$$

Then adding a random Laplace noise with parameter mb/α to the contribution of each layer i, is sufficient to satisfies α-differential privacy. Furthermore, using Corollary 7 we can see that the additive error introduced by noise is only $O(\frac{b}{\alpha}\log^{3/2}(\frac{T}{b})\log(\frac{2}{\delta}))$. □

Approximation Error. Our algorithm smooths the contribution of each element in the stream according to its layer leading to an underestimated value for the length of the LIS. The following Theorem summarizes the approximation ratio in the worst case.

Theorem 15 (Hierarchy Approximation Error)**.** *Let σ be a stream of length T, and b be the number of buckets in each layer of our algorithm. Then, the hierarchy mechanism returns a $(1 - \frac{T-b}{T+b})$-approximation of the length of LIS.*

Proof. Let k be the length of the LIS over the entire stream. We begin by showing that this algorithm never overestimates the length of the LIS and then proceed by showing the error in the underestimate. To understand why this algorithm always reports a length of the LIS less or equal to the real length we consider the following case. Let us assume that there exists an element $\sigma(j)$ in a bucket at level $i > 0$ in our algorithm that differs from the Patience Sort. Since this element is extra in our algorithm it means that there is an element $\sigma(j')$, $j' > j$ that replaces $\sigma(j)$ in the exact Patience Sort. Since $\sigma(j') < \sigma(j)$, we have that in our structure $\sigma(j')$ has replaced another element $\sigma(j'')$. Due to the nature of our algorithm this operation could only occur in a layer $i' < i$, hence in replacing $\sigma(j'')$ with $\sigma(j')$ in our algorithm we have a larger loss of contribution than replacing $\sigma(j)$. Therefore we cannot have an overestimate length of the LIS.

Now, we examine the error in underestimating the length of the LIS. Consider a worst case scenario where only the first k symbols in σ contribute to the LIS, while the rest of the stream does not increase the length of the LIS. In this situation, as the stream proceeds the elements of the LIS that initially are in layer 0 are progressively moved up introducing a small additive error. Below, we quantify this error. Let $m = \log(\frac{T}{b} + 1) - 1$ be the number of layers in our structure, then the maximum additive errors on the LIS is achieved when all the elements forming the LIS are in layer m. This quantity is computed as follows.

$$\sum_{i=1}^{m} \frac{k}{2^i} = k\left(\frac{T-b}{T+b}\right) \tag{8}$$

Hence the returned value from our algorithm is lower bounded by $LIS(\sigma)(1 - \frac{T-b}{T+b})$. This shows that our returned length $\tilde{l}(\sigma)$ satisfies the following inequality.

$$LIS(\sigma)\left(1 - \frac{T-b}{T+b}\right) \leq \tilde{l}(\sigma) \leq LIS(\sigma) \tag{9}$$

Therefore, our algorithm provides a $(1 - \frac{T-b}{T+b})$-approximation of the length of LIS. □

Space Analysis. Since this algorithm simulates the Patience Sorting algorithm by keeping only the top of the piles forming the LIS, it follows that the space complexity is linear with the length of the LIS in the stream $O(LIS(\sigma))$.

Time Analysis. For any new incoming element in the stream, the total running time is given by the cost required for updating each pile. There are at most $m - 1$ buckets, one for each level, that

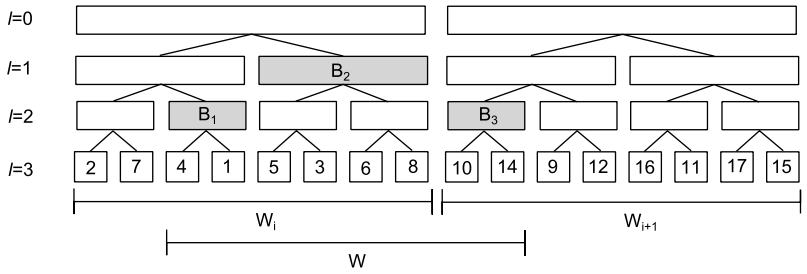


Figure 5: The query on the window W is answered by combining the LIS from the blocks in grey that belong to the binary decompositions of the windows W_i and W_{i+1}.

need update, where each operation requires $O(\log b)$ time. Since $m = \Theta(\log \frac{T}{b})$, the update time is $O(\log b \log \frac{T}{b})$.

This solution points out a strong connection between the approximation ratio and the noise required to achieve privacy. We can see that increasing b has a beneficial effect on the approximation ratio but on the other hand increases the privacy cost. In fact, as an extreme case using $b = T$ the algorithm returns the exact length of the LIS but incurs a large perturbation noise. Compared with our decomposition framework, this algorithm provides the user with a way to balance the approximation ratio and the noise due to the privacy mechanism.

6 Extension: Query over Sliding Windows

In the previous sections, we illustrated two solutions for computing the length of the longest increasing subsequence over the entire stream. However, in some applications the most recent part of the stream is more important than the past in answering queries. In fact, a complete view of the stream may obfuscate transitional phenomena that recently occur in the stream. Consider for example a traffic monitoring setting, where the traffic data arrives in a streaming manner from sensors placed along the highway. Knowing the current traffic situation may help to understand the presence of congested areas along the highway and finally help drivers to select alternative routes. Therefore, in this section we study the problem of computing the longest increasing subsequence on the recent past which is modeled by considering a sliding window W containing the most recent $|W|$ symbols in the stream. In this setting, our goal consists in reporting an estimate of the $LIS(\sigma(i-|W|+1)\cdots\sigma(i))$ at every time i limiting our focus on the last $|W|$ symbols rather than considering the whole stream.

6.1 Binary Tree Extension

We first propose an extension of the binary decomposition of the stream to solve this problem, where now the binary tree indexes the blocks in the part of the stream contained in the window W. It is important to notice that if we keep a binary decomposition for every window we will incur a large privacy and computational cost. In fact, removing any element in the stream $\sigma(i)$ affects $O(|W|)$ binary trees that are overlapping on that element. Hence at most $O(|W|\log|W|)$ computations can be affected for all the blocks of the binary decompositions in the sliding windows.

To reduce this privacy cost we use the technique proposed by Bolot et al. [4] to answer counting query over sliding windows. In our case, we partition the stream into non-overlapping regions $W_i = \sigma(i|W|), \cdots, \sigma((i+1)|W|-1)$ of length $|W|$, where for each W_i we use a binary decompo-

Algorithm 4 Sliding Windows

procedure SLIDING WINDOWS(W, α, σ)
Input: window length W; privacy parameter α; event stream σ
Output: $\tilde{l}(W)$ released longest increasing subsequence over the window

  for ($i = 0, 1, \dots$) **do**
    Determine the range W_{i+1} containing $\sigma(i)$
    Update the current active blocks
    $k \leftarrow \lfloor i/|W| \rfloor |W| + i \mod |W|$
    $W_{pref} = \sigma(i - |W| + 1) \cdots \sigma(k)$
    $W_{suff} = \sigma(k + 1) \cdots \sigma(i)$
    Combine the noisy LIS from W_{pref} and W_{suff} in $\tilde{l}(W)$
    return $\tilde{l}(W)$
  end for
end procedure

sition. For those windows that span between two regions W_i and W_{i+1}, the answer can be computed combining the blocks shared in those regions. An example is reported in Figure 5.

In particular, for any window query $W = \sigma(i - |W| + 1) \cdots \sigma(i)$ at time i in the stream, the answer for W is obtained by combining the length of the LIS from multiple blocks in two binary decompositions. Let W_{pref} and W_{suff} denote the prefix and suffix part of W respectively, where $W_{pref} = \sigma(i - |W| + 1) \cdots \sigma(k)$ and $W_{suff} = \sigma(k + 1) \cdots \sigma(i)$ with $k = \lfloor i/|W| \rfloor |W| + i \mod |W|$. Furthermore, let W_{i+1} be the range containing the element $\sigma(i)$. Then the answer for the query window W is obtained by combining the LIS computed for W_{pref} and W_{suff} using the binary decompositions W_i and W_{i+1} respectively. An overview of the procedure for the query window is reported in Algorithm 4.

Example 16. Consider the situation as illustrated in Figure 5, where we report the LIS for the window W. The window query $W = 4, 1, 5, 3, 6, 8, 10, 14$ is divided into two parts $W_{pref} = 4, 1, 5, 3, 6, 8$ and $W_{suff} = 10, 14$ that belong to W_i and W_{i+1} respectively. The length of the LIS over the window W is obtained by combining the longest increasing subsequence in the binary block decomposition for W_{pref} and W_{suff}. The prefix part is decomposed into block $B_1 = 4, 1$ and $B_2 = 5, 3, 6, 8$, while the suffix part is represented with the block $B_3 = 10, 14$.

Privacy Analysis. We can observe that removing any element in the stream affects only $O(\log |W|)$ values of LIS in the binary decomposition of the window range W. Therefore, it is sufficient to add noise proportional to $Lap(\frac{\log |W|}{\alpha})$ to each LIS in the blocks to satisfy the privacy requirement.

Utility Analysis. The utility analysis is similar to the binary decomposition approach. Notice that in this case there could be at most two blocks with the same length that are combined. Although this differs from the case we saw in the previous section, the worst case guarantee does not change.

Theorem 17 (Window Query utility)**.** *The window query decomposition algorithm for computing the longest increasing subsequence over a sliding window W is* $(\log |W| + \frac{\log^{3/2} |W|}{LIS(W)\alpha} \ln \frac{1}{\delta}, \delta)$*-useful.*

Proof. At every time step i, for a window query W the number of blocks involved in reporting the LIS is at most $O(\log |W|)$. The additive error due to the privacy mechanism is computed using Corollary 7, where the total error is due to the sum of $O(\log |W|)$ i.i.d. Laplace random variables of parameter $\log |W|/\alpha$. Therefore, with probability at least $1 - \delta$ the error due to the privacy mechanism is $O(\frac{\log^{3/2} |W|}{\alpha} \ln \frac{1}{\delta})$. Furthermore, using Lemma 10 the error due to combining the

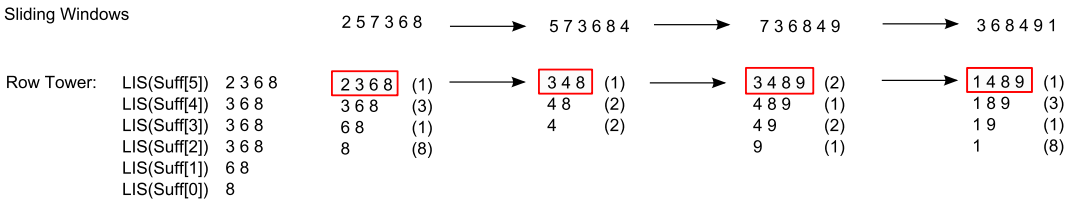


Figure 6: Example for the input stream $\sigma = 257368491$ as in [2], with sliding window W of length 6.

LIS from the blocks in the binary decompositions is bounded by $O(\log |W| \cdot LIS(W))$. Therefore combining these two errors and normalizing by $LIS(W)$ we obtain the final utility guarantees. □

Space Analysis. To analyze the space requirements for this approach we can observe that at every time we need to store only the information related to two binary decompositions. Therefore, the total memory requirements for this technique is $O(|W|)$.

Time Analysis. The update time for this approach follows directly from the time analysis for the binary decomposition. Therefore, the time for each update is $O(\log |W| \log LIS(W))$.

In restricting the focus of the LIS query on the most recent $|W|$ symbols, we can observe that the impact of the perturbation noise can be reduced to a quantity that only depends on the length of the window W.

6.2 Hierarchy Extension

The hierarchy mechanism can be adapted to compute the length of the LIS over a sliding window W by introducing two modifications in the original algorithm. First, we restrict the number of layers m to a quantity that is $\Theta(\ln \frac{|W|}{b})$, so that the maximum length of the LIS is bounded within each window W. Second, for any position i in the stream, we need to keep track of all the LIS for the suffixes starting at positions: $i - |W| + 1, i - |W| + 2, ..., i$. In fact, when a window is shifted from position i to position $i+1$, we can compute the new LIS by extending some of the LIS for the previous suffixes. A naive implementation of this observation requires to keep multiple hierarchy structures, one for each suffix yielding to an overall space requirement of $O(|W|LIS(W))$. In [2], the authors proposed a compact representation for storing the information regarding the length of the LIS for all the suffixes in the sliding window. In their representation, the information for the Patience Sorting algorithm is implicitly preserved in a structure (row tower) where only the *principal row tower* (i.e. LIS for the longest suffix) is preserved while the LIS for the other suffixes can be computed by constructing the respective rows from the principal one. In fact, it can be shown that each row other than the first one in the row tower is either the same as the one above, or can be obtained from it by deleting a single element. Therefore, the duplicate rows can be removed and record the multiplicity of the original rows.

Example 18. Consider the situation in Figure 6, where $|W| = 6$. On the left, we report the full Patience Sorting representation for all the subsequences associated with the suffixes in the sliding window W. The duplicated rows are removed and only a single copy is preserved together with their multiplicity. The principal row tower (circled in red) is on the top of the structure and it is used to keep track of the length of the LIS in the current window.

In our solution, we adapt this structure where each row is kept by using our hierarchy representation. In this way, we can preserve the information of the LIS for all the subsequences in the window

Table 1: Summary of results for LIS query over entire stream.

Method	Error	Memory	Update Time
Baseline	$O(\frac{\beta\sqrt{T}}{\alpha} \ln \frac{1}{\delta})$	$O(LIS(\sigma))$	$O(\log \frac{\sqrt{T}}{\beta})$
Binary	$O((\log T - 1) + \frac{\beta \log^{3/2} T}{\alpha\sqrt{T}} \ln \frac{1}{\delta})$ where $\beta \geq 1$ $O(\log \beta\sqrt{T} + \frac{\beta \log^{3/2} T}{\alpha\sqrt{T}} \ln \frac{1}{\delta})$ where $\beta \in [1/\sqrt{T}, 1)$	$O(LIS(\sigma) \ln(\beta^2 LIS(\sigma)))$	$O(\log T \log \frac{\sqrt{T}}{\beta})$
Hierarchy	$O((1 - \frac{T-b}{T+b}) + \frac{b\beta}{\sqrt{T}\alpha} \log^{3/2}(\frac{T}{b}) \log(\frac{2}{\delta}))$	$O(LIS(\sigma))$	$O(\log b \log \frac{T}{b})$
Binary Extension	$O(\log \lvert W\rvert + \frac{\log^{3/2} \lvert W\rvert}{LIS(W)\alpha} \ln \frac{1}{\delta})$	$O(\lvert W\rvert)$	$O(\log \lvert W\rvert \log LIS(W))$

and perturb the length of the principal row tower in reporting the current length of the LIS. This preliminary investigation for extending our approach to sliding windows queries provide us with useful insights about the compact representation of the stream that will be further investigated in future works.

7 Summary of Results

Table 1 summarizes the utility results of our proposed solutions. The first three entries refer to our solutions for computing the length of the LIS over the entire stream, while the last entry represents our extension for sliding windows.

Regarding the whole stream, we observe that both our binary and hierarchy solutions outperform the baseline approach in many perspectives. We notice that the baseline approach incurs a large perturbation error which could dramatically compromise the utility. Specifically, the additive error in the baseline strategy grows linearly with the length of the stream. For the binary decomposition instead, we provide output-sensitive utility results showing the benefits of this technique for different lengths of LIS. Due to the use of disjoint blocks, this approach incurs a considerably smaller perturbation error with respect to the baseline solution. In fact, the dependency of the error with respect to the perturbation noise is only polylogarithmic in this case. Furthermore, we can observe that the decomposition framework has small space requirements and update time. In principle, the space and time complexity of this solution could be further improved by using more sophisticated algorithms (e.g. [15, 24]) as an internal procedure instead of relying on the Patience Sorting. For count based statistic the binary decomposition has been shown very effective; however due to the nature of the LIS, this strategy incurs an approximation error. Our hierarchy approach specifically addresses the LIS problem by directly simulating the Patience Sorting algorithm. This procedure incurs a small running time and it has small memory requirements. Comparing the worst case performance of this technique with the binary decomposition, we can observe that the decomposition framework is still superior leading to a smaller additive error with the same approximation ratio. This result is due to the fact that the hierarchy strategy suffers when the LIS constitutes the initial part of the stream. In fact, as the execution proceeds the elements in the sketch are moved in higher level increasing the approximation error over the stream. However, we can notice that in real scenarios such situation is unlikely to occur because in many applications the stream presents trends over time.

In the case of sliding windows, we have a considerable improvement in the performances. Comparing the binary solution on the whole stream with its extension on sliding windows, we observe an error reduction for the computed LIS. In fact, the error dependency is only on the size of the window W rather than the length of the stream T. We have as well a reduction in the memory complexity and update time. These advantages are due to the use of bounded windows that limits the impact of the privacy mechanism on the output statistics. In fact, in this case the sensitivity of the output is only proportional to W rather than the whole stream. This reduces the perturbation noise, hence we achieve higher utility.

Table 2: Data Streams Characteristics

Data Stream	Length	$\mathbf{l_{max}}$	$\mathbf{l_{avg}}$
finance	365	0.027	0.075
gasoline	1315	0.127	0.14

Table 3: Default Algorithmic Parameter Values

Parameter	Description	Value
α	Privacy Parameter	1
γ	Number of buckets in HM	0.01
$\|W\|$	Window size	32

8 Experiments

In this section, we report the empirical evaluation of our proposed solutions for privately computing the length of the LIS on two real-world data streams. In each setting, we evaluate the performance of the baseline approach (base), Binary decomposition framework (binary) and Hierarchy mechanism (HM) described in Section 3.2, Section 4 and Section 5 respectively. Furthermore, we compare these solutions with the non-private approach based on the Patience Sorting algorithm to understand the impact of the privacy and approximation on the final utility.

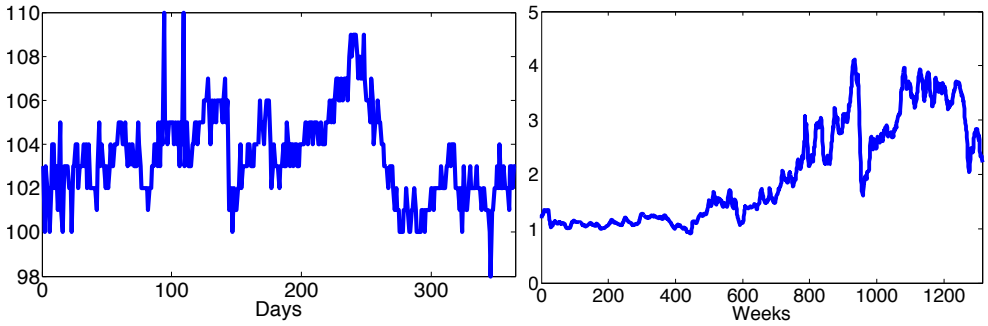
Data Stream Description. In our experiments, we use two real-world data streams: finance and gasoline. The first data stream represents the unit price variation for the stock labeled with the symbol C in the TRACE database over a period of one year. The complete dataset contains a set of financial transactions ordered by time reporting the unit stock price and the quantity purchased. This data stream is available from the Wharton Research Data Services (WRDS) [1] for academic use. In our settings, we extracted the daily price changes of the stock over a fixed year; therefore, the overall selected stream contains 365 data points. The second stream represents the weekly retail gasoline price (dollar per gallon) in U.S. from August 1990 to October 2015 [2]. Figure 7 illustrates the data streams considered in our experiments.

From the graphs is Figure 7, we observe that these two data streams have different characteristics. In the finance stream, the are many price fluctuations that indicate the high variability in the stock price. On the other hand, in the gasoline case, the gas price shows a constant increment over the last 6 years (excluding the economic downturn in December 2008-January 2009, where the gas price had a considerable drop). These differences are reflected also in the lengths of the LISs in these streams. In Table 2, we report for each dataset, the overall length of the stream T, the relative maximum length of the LIS ($l_{max} = \frac{LIS_{max}}{T}$), and the relative average length of the LIS ($l_{avg} = \frac{1}{T}\sum_{i=1}^{T}\frac{LIS(\sigma[0,i])}{i}$). Intuitively, l_{max} measures the longest LIS in the stream compared to overall length of the time-series. On the other hand, the quantity l_{avg} measures the average length of the LIS at each time stamp in the stream. From these values, we observe that the gasoline stream has considerably longer LISs compared to the finance stream confirming the differences in behavior of these streams.

Algorithm Setup. The default parameter values for our algorithms are reported in Table 3. Note that the number of layers for the HM approach are computed as $\log \frac{T}{b}$, where $b = \gamma \cdot T$ is the number

[1] https://wrds-web.wharton.upenn.edu/wrds/

[2] http://www.eia.gov/dnav/pet/hist/LeafHandler.ashx?n=PET&s=EMM_EPMR_PTE_NUS_DPG&f=W



(a) `finance` Data stream: daily stock price

(b) `gasoline` Data stream: weekly gasoline price ($ per gallon)

Figure 7: Data Streams

of buckets and T is the length of the overall stream. Since the absolute length of the LIS in these data stream is quite small, we use a default privacy budget $\alpha = 1$ to obtain meaningful results.

8.1 Metrics

Mean Relative Error. In our evaluations to measure the utility of the released statistics we consider the Mean Relative Error (MRE) between the released perturbed length of the LIS series and the original series computed by the non-private Patience Sorting algorithm. In particular, let $\hat{l}_i$ be the privately computed length of the LIS up to time i in the data stream and let l_i denote the real length of the LIS till this time. Then, the MRE metric provides an indication on how well the released sequence $\{\hat{l}_i\}_{i=1}^{T}$ follows the original sequence $\{l_i\}_{i=1}^{T}$. The MRE is a generic metric to evaluate data accuracy, disregarding the actual, domain-specific applications. More formally, we define the MRE error as follows:

$$MRE = \frac{1}{T}\sum_{i=1}^{T}\frac{|l_i - \hat{l}_i|}{l_i} \tag{10}$$

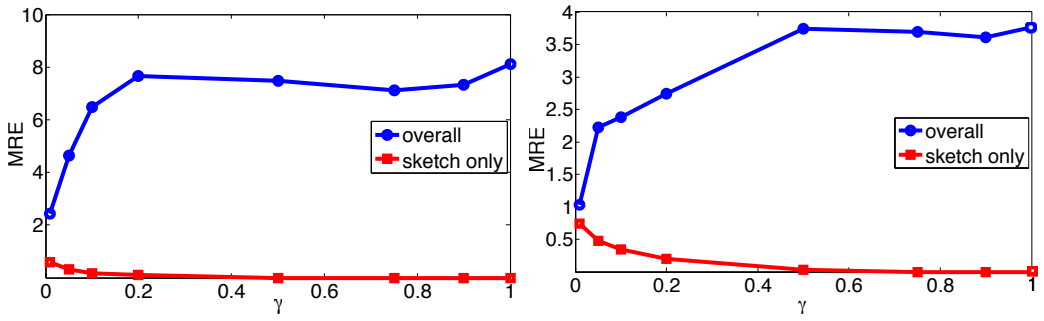
From the definition above, we observe that smaller values of MRE imply higher similarity between the released and the original series, hence higher utility.

Mean Absolute Error. In addition to the previous metric, we also measure the Mean Absolute Error (MAE) between the released perturbed length of the LIS series and the original series computed by the non-private Patience Sorting algorithm. Using the notation introduced in the previous section, we define the MAE error as follows:

$$MAE = \frac{1}{T}\sum_{i=1}^{T}|l_i - \hat{l}_i| \tag{11}$$

Similarly to the the MRE, smaller values of MAE imply higher similarity between the released and the original series, hence higher utility.

Both these utility metrics have been extensively used to measure the quality of the released statistic over time in many privacy preserving works, such as in [18].



(a) `finance` Data stream: MRE vs b. (b) `gasoline` Data stream: MRE vs b.

Figure 8: Impact of the number of buckets in `HM` mechanism.

8.2 Evaluation on the Entire Stream

8.2.1 Utility Evaluation

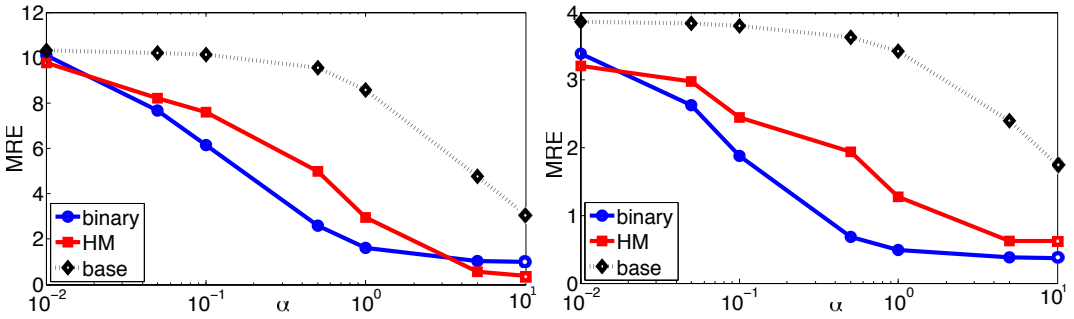
In this section, we study the utility of our privacy preserving solutions in releasing the length of the LIS over the entire data streams. We provide experimental results to understand the ability of our approaches in preserving the length of the LIS in terms of mean relative error and mean absolute error.

Impact of the number of buckets. We evaluate the impact of the number of buckets b used in the hierarchy mechanism on the quality of the length of the LIS returned by the hierarchy solution in terms of MRE (Figure 8). Specifically, we consider b as a fraction of the original length of the stream $b = \gamma \cdot T$, where γ is a parameter in the range $[0.01, 0.4]$.

For each data stream, we report the MRE obtained by the sketching approach without considering the privacy perturbation mechanism, denoted as `sketch only`, and the overall MRE obtained by perturbing the length of the LIS in each layer, denoted as `overall`.

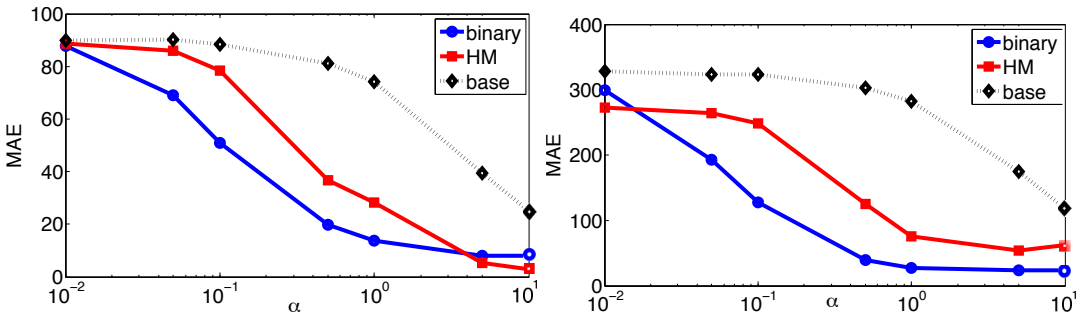
We observe that in both data streams the error obtained by the `overall` mechanism is considerably higher than the one produced by the only use of the hierarchy data structure. In fact, as b increases a more accurate length of the LIS can be computed since more elements can be maintained in the lower level of the sketch. This points out that our proposed data structure is effective in summarizing the stream leading to low MRE. On the other hand, the overall performance is dominated by the perturbation noise introduced by the privacy mechanism which leads to an increment of the MRE in our final solutions. The overall performances of `HM` approach are superior in the `gasoline` stream since in this case the LIS tends to be relatively longer compared to the `finance` stream. In fact, in the `finance` stream we observe several price fluctuations and no monotonic subsequences are evident. In general the length of the LIS is relatively small compared to the length of the streams considered in input, hence even a small amount of noise may introduce a large relative error in the returned statistics.

Impact of the privacy level on MRE. In Figure 9, for all three methods, we vary the privacy budget (α) and we report their MRE values in computing the length of the LIS. We observe that for every approach the MRE drops as the privacy budget increases (i.e. less privacy). In fact, as the privacy budget α increases a smaller perturbation error is introduced by the differential privacy mechanism leading to higher utility. The baseline approach `base` which directly releases perturbed length of the LIS incurs the highest MRE error in every privacy setting. Both `binary` and `HM` provide higher utility results than the baseline solution. In fact, despite the approximation introduce in `binary`



(a) `finance` Data stream: MRE vs α. (b) `gasoline` Data stream: MRE vs α.

Figure 9: Impact of the privacy parameter on the MRE.



(a) `finance` Data stream: MAE vs α. (b) `gasoline` Data stream: MAE vs α.

Figure 10: Impact of the privacy parameter on the MAE.

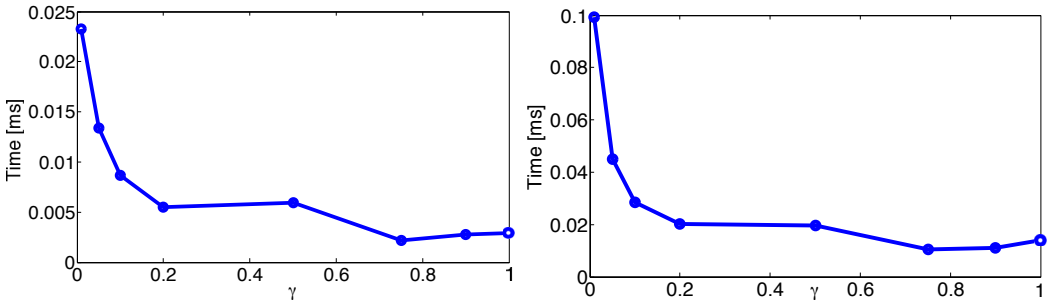
and `HM`, these approaches benefit from the perturbation noise reduction.

Impact of the privacy level on MAE. In Figure 10, we report the MAE error of all three approaches for different values of privacy budget α. Similarly, for the case of MRE, as privacy level decreases (i.e. higher values of α), a smaller amount of noise is injected leading to smaller MAE value, hence higher utility. Also in this case, the binary and hierarchy solutions provide superior results compared to the baseline solution.

8.2.2 Running Time Evaluation

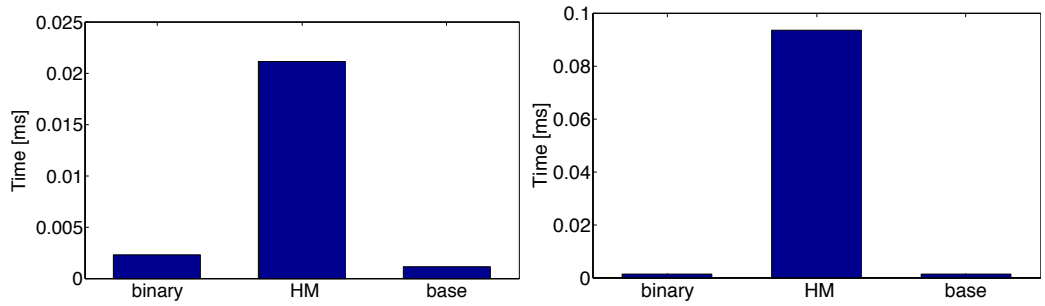
In this section, we evaluate the running time of the three proposed algorithms. Such a time is measured as the overall running time normalized by the stream length. Intuitively, this represents the average update time required each time a new element appears in the stream. We first study the impact of the number of buckets b on the performance of our `HM` approach and successively we provide an overview of the performance of our solutions.

Impact of the number of buckets on the running time. Figure 11 illustrates the average running time of the `HM` approach on both data streams for different values of $b = \gamma \cdot T$. We observe as the number of buckets increases in the data structure the running time tends to decrease in both streams. In fact, by using more buckets we considerably reduce the number layers in the data structure leading to a smaller number of operations required to update the layers when elements expire. Although a finer granularity could help both from the running time and utility perspective, the privacy cost for



(a) `finance` Data stream: time vs b (b) `gasoline` Data stream: time vs b.

Figure 11: Time vs number of buckets b.



(a) `finance` Data stream: update time (b) `gasoline` Data stream: update time.

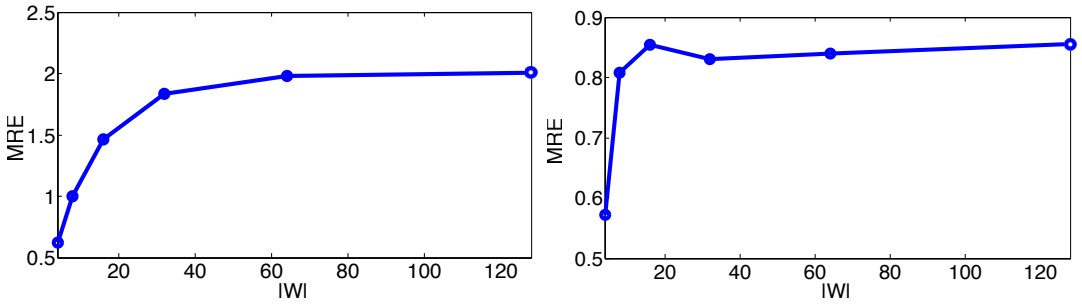
Figure 12: Update time.

protecting the elements reduces the overall utility of our approach as we showed in Figure 8. Hence, as default setup for this approach we use $\gamma = 0.01$.

Running time comparison. In Figure 12, we report the average update time required by all the three approaches. The baseline approach `base` incurs the smallest update time due to the fact that only one pile of the Patience Sorting algorithm may be updated each time a new element arrives in the stream. Also the binary decomposition approach `binary` has a small update time since at each time only a logarithmic number of blocks are active. On the other hand, `HM` incurs the highest update time since all the buckets in all the layers are tested for expiration. Contrary to the other two approaches, in this case the computational cost of these operations is the same regardless of the current length of the LIS but it rather depends on the size of the data structures (i.e. number of buckets and number of layers). Therefore, this approach may not take advantage of short LIS in terms of running time. This phenomenon is more evident when comparing the running time in the two data streams (Figure 12a and Figure 12b) over in the `gasoline` stream. In this case, the `HM` has a considerably higher increment of running time due to the larger data structure size compared to the other two approaches.

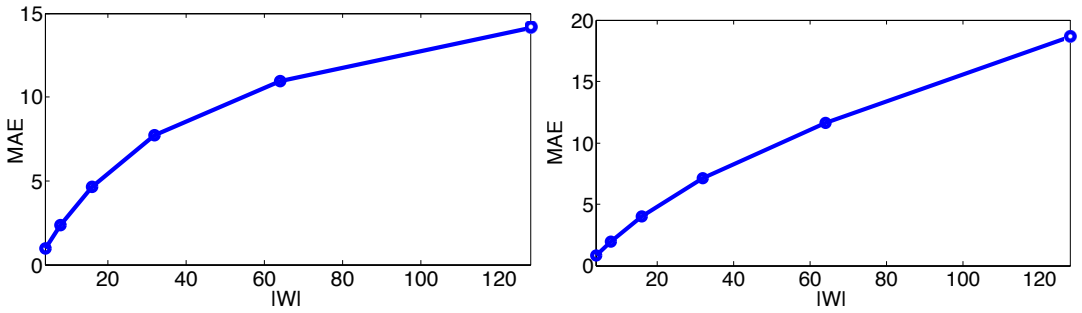
8.3 Evaluation on Sliding Windows

In this section, we focus on evaluating the extension of the `binary` approach for computing the length of the LIS over sliding windows. We chose this approach for two main reasons: (1) from the previous section it provides slightly superior performance than the `HM` solution and (2) the whole



(a) `finance` Data stream: MRE vs $|W|$. (b) `gasoline` Data stream: MRE vs $|W|$.

Figure 13: Impact of the window size on the MRE.



(a) `finance` Data stream: MAE vs $|W|$. (b) `gasoline` Data stream: MAE vs $|W|$.

Figure 14: Impact of the window size on the MAE.

stream solution can be easily extended to handle the computation of the LIS over windows as we discussed in Section 6.1.

8.3.1 Utility Evaluation

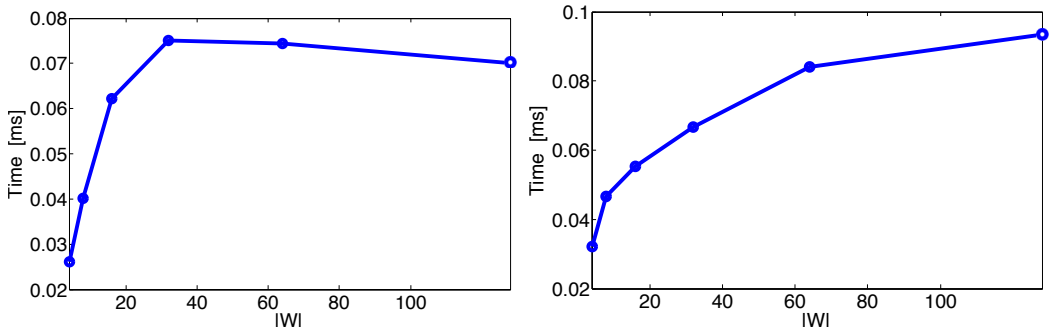
In this setting, we fix the privacy parameter $\alpha = 1.0$ and we vary the length of the sliding window W to understand how this parameter impacts the utility of our solution.

Impact of the window size on the MRE. In Figure 13, we report the MRE values for increasing window size. We observe that as $|W|$ increases the MRE increases in both settings. When we increase the window W, the impact of each element on the final output increases and since the privacy parameter is fixed, we incur a larger perturbation noise, hence lower utility. Nevertheless, in comparison with the results in Figure 9, where the entire stream is considered, we observe a great improvement in the performance due to the beneficial effect of introducing a sliding window which limits the privacy perturbation noise.

Impact of the window size on the MAE. Similarly to the behavior observed for the MRE, the MAE increases as the window is enlarged. Also in this case larger window sizes incur a larger perturbation noise as shown in Figure 14.

8.3.2 Running Time Evaluation

Here we measure the average running time of our solution with respect to different window sizes.



(a) `finance` Data stream: time vs $|W|$ (b) `gasoline` Data stream: time vs $|W|$.

Figure 15: Time vs window sizes $|W|$.

Impact of the window size on the running time. Figure 15 reports the running time for increasing window sizes. As more elements are considered in the window a larger binary structure is constructed, hence we observe an increasing update time. In this case, the LIS is computed over sliding windows of length $|W|$ and it is sufficient to divide the stream into non-overlapping regions of size $|W|$ (i.e. roughly $\Theta(|T|/|W|)$ binary structures) to answer all the sliding windows query. Therefore, the overall time grows linearly with the size of the stream, while for the solution that handle the whole stream this dependency is only logarithmic. Nevertheless, the update time is only logarithmic with the window size.

8.3.3 Detecting Surprising Trends

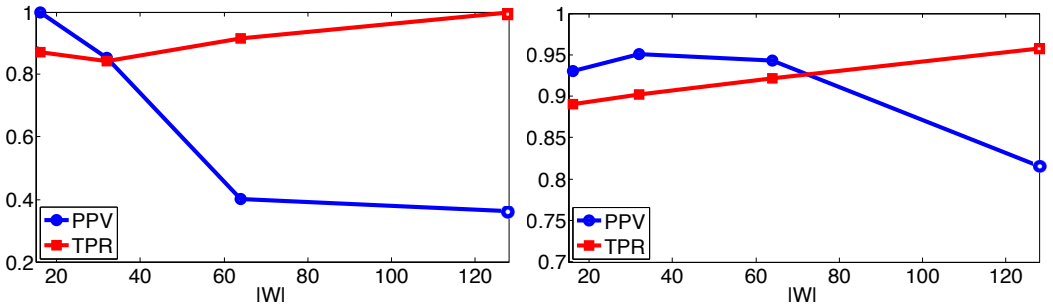
As we mentioned before, the computation of the length of the LIS over a data stream can be often associated to time-series monitoring activities. Therefore, the ability of privately computing such statistics in an online manner is an important indication of the usability of our solutions. In our case, we monitor the length of the LIS to detect *surprising trends* over sliding windows. Specifically, we adapt the definition of surprising patterns introduced in [19] to our setting. In our evaluation, we say that a surprising trend occurs at time i in the input stream σ if and only if $LIS(\sigma[i-|W|+1, i]) \geq th$, where th is a user specified threshold that indicates the level of *"surpriseness"*. In our case, this definition allows us to detect windows of the original stream containing unexpected long increasing subsequence indicating unusual price increments, for example.

To measure the ability of our solution in privately detecting such surprising trends we report the *precision* (PPV) and *recall* (TPR). More formally, these quantities are defined as follows:

$$PPV = \frac{TP}{TP + FP} \tag{12}$$

$$TPR = \frac{TP}{TP + FN} \tag{13}$$

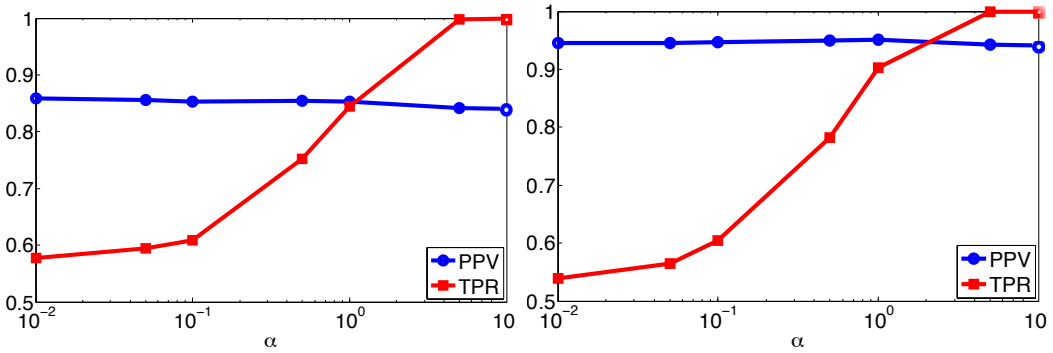
where TP represents the true positive (i.e. the number of position i correctly detected), FP represents the false positive (i.e. the number of position i incorrectly detected) and FN denotes the false negative (i.e. the number of position i incorrectly missed). From the definition above, higher values of precision indicate the ability of the solution in reporting a high fraction of the detected position that are associated to surprising trends. On the other hand, higher values of recall imply the ability of retrieving a high percentage of all relevant positions that are associated to surprising trends.



(a) `finance` Data stream: PPV and TPR vs $|W|$, $\theta = 0.1$.

(b) `gasoline` Data stream: PPV and TPR vs $|W|$, $\theta = 0.15$.

Figure 16: Precision and Recall vs window size $|W|$.



(a) `finance` Data stream: PPV and TPR vs α

(b) `gasoline` Data stream: PPV and TPR vs α.

Figure 17: Precision and Recall vs privacy parameter α.

Impact of the threshold. We first examine the accuracy in detecting trends by considering different window lengths. Specifically, we define a threshold value $th(W) = \theta \cdot |W|$ which is window dependent. We proceed in this way because using a threshold that depends on the window allows us to treat all the LIS in the same way regardless of the specific length of the window considered. Our results are reported in Figure 16. As $|W|$ increases we observe similar utility performance on the two data streams. As the window sizes increase the precision PPV of our approach tends to decrease. We notice that in the `gasoline` data stream, the precision is considerably higher than in the `finance` data stream. We believe that this is due to the fact that the former stream contains longer LISs than the latter which facilitates the detection of surprising trends. In terms of recall TPR, we observe that its value tends to increase after an initial drop. For the successive experiments, we select a window size $|W|$ of 32 symbols which, from Figure 16, yields a good trade-off between precision and recall.

Impact of the privacy level. Figure 17 reports the precision and recall of our privacy solution varying privacy levels. As the privacy level increases, less perturbation noise is injected and we observe a reduction of false negative FN and an increment of true positive TP. Overall, we have an improvement on the recall as the privacy level decreases while the precision remains stable.

9 Conclusions

In this paper, we considered the problem of privately detecting trends in stream data. Specifically, we addressed the problem of computing the length of the LIS while protecting the presence of single event in the stream. We developed two different solutions that provide formal guarantee of privacy. The first approach approximates the length of the LIS by assembling local information computed on segments of the stream. The second approach constructs a small sketch of the stream by exploiting the structure of the problem. Using a rigorous analysis, we showed that these strategies provided significant benefits over the baseline approach. Furthermore, we demonstrated the ability of our solutions in computing the length of the LIS and detecting surprising patterns on real-world data streams.

For the future, we consider to broaden the range of applications (e.g. activity monitoring) that can benefit from our solutions as well as developing enhanced and small-memory impact algorithms. We also aim to further improve our hierarchy solution to solve sliding window query by using some of the techniques in [2].

Acknowledgments

We thank the anonymous reviewers for the precious comments and suggestions that greatly helped to improve the quality of the paper. This material is based upon work supported by the National Science Foundation under Grant No. 1117763.

References

[1] Miklós Ajtai, T. S. Jayram, Ravi Kumar, and D. Sivakumar. Approximate counting of inversions in a data stream. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, STOC '02, pages 370–379, New York, NY, USA, 2002. ACM.

[2] Michael H Albert, Alexander Golynski, Angle M Hamel, Alejandro Lpez-Ortiz, S.Srinivasa Rao, and Mohammad Ali Safari. Longest increasing subsequences in sliding windows. *Theoretical Computer Science*, 321(23):405 – 414, 2004.

[3] David Aldous and Persi Diaconis. Longest increasing subsequences: From patience sorting to the baik-deift-johansson theorem. *Bull. Amer. Math. Soc*, 36:413–432, 1999.

[4] Jean Bolot, Nadia Fawaz, S. Muthukrishnan, Aleksandar Nikolov, and Nina Taft. Private decayed predicate sums on streams. In *Proceedings of the 16th International Conference on Database Theory*, ICDT '13, pages 284–295, New York, NY, USA, 2013. ACM.

[5] T.-H. Hubert Chan, Elaine Shi, and Dawn Song. Private and continual release of statistics. *ACM Trans. Inf. Syst. Secur.*, 14(3):26:1–26:24, November 2011.

[6] T.-H.Hubert Chan, Elaine Shi, and Dawn Song. Privacy-preserving stream aggregation with fault tolerance. In AngelosD. Keromytis, editor, *Financial Cryptography and Data Security*, volume 7397 of *Lecture Notes in Computer Science*, pages 200–214. Springer Berlin Heidelberg, 2012.

[7] Graham Cormode, S. Muthukrishnan, and Süleyman Cenk Sahinalp. Permutation editing and matching via embeddings. In *Proceedings of the 28th International Colloquium on Automata, Languages and Programming,*, ICALP '01, pages 481–492, London, UK, UK, 2001. Springer-Verlag.

[8] Cynthia Dwork. Differential privacy. In *ICALP*, pages 1–12, 2006.

[9] Cynthia Dwork. Differential privacy in new settings. In *SODA*, pages 174–183, 2010.

[10] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006*, pages 265–284, 2006.

[11] Cynthia Dwork, Moni Naor, Toniann Pitassi, and Guy N. Rothblum. Differential privacy under continual observation. In *Proceedings of the Forty-second ACM Symposium on Theory of Computing*, STOC '10, pages 715–724, New York, NY, USA, 2010. ACM.

[12] Cynthia Dwork, Moni Naor, Toniann Pitassi, Guy N. Rothblum, and Sergey Yekhanin. Pan-private streaming algorithms. In *ICS*, pages 66–80, 2010.

[13] Funda Ergun and Hossein Jowhari. On distance to monotonicity and longest increasing subsequence of a data stream. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '08, pages 730–736, Philadelphia, PA, USA, 2008. Society for Industrial and Applied Mathematics.

[14] Liyue Fan and Li Xiong. An adaptive approach to real-time aggregate monitoring with differential privacy. *Knowledge and Data Engineering, IEEE Transactions on*, 26(9):2094–2106, Sept 2014.

[15] Parikshit Gopalan, T. S. Jayram, Robert Krauthgamer, and Ravi Kumar. Estimating the sortedness of a data stream. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '07, pages 318–327, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[16] Anupam Gupta and Francis X. Zane. Counting inversions in lists. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '03, pages 253–254, Philadelphia, PA, USA, 2003. Society for Industrial and Applied Mathematics.

[17] J. M. Hammersley. A few seedlings of research. In *Proceedings of the Sixth Berkeley Symposium on Mathematical Statistics and Probability*, pages 345–394, Berkeley, Calif., 1972. University of California Press.

[18] Georgios Kellaris, Stavros Papadopoulos, Xiaokui Xiao, and Dimitris Papadias. Differentially private event sequences over infinite streams. *Proc. VLDB Endow.*, 7(12):1155–1166, August 2014.

[19] Eamonn Keogh, Stefano Lonardi, and Bill'Yuan-chi' Chiu. Finding surprising patterns in a time series database in linear time and space. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 550–556. ACM, 2002.

[20] David Liben-Nowell, Erik Vee, and An Zhu. Finding longest increasing and common subsequences in streaming data. In *Proceedings of the 11th annual international conference on Computing and Combinatorics*, COCOON'05, pages 263–272, Berlin, Heidelberg, 2005. Springer-Verlag.

[21] Frank D. McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *SIGMOD '09*, SIGMOD '09, pages 19–30, New York, NY, USA, 2009.

[22] Darakhshan Mir, S. Muthukrishnan, Aleksandar Nikolov, and Rebecca N. Wright. Pan-private algorithms via statistics on sketches. In *Proceedings of the Thirtieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '11, pages 37–48, New York, NY, USA, 2011. ACM.

[23] Vibhor Rastogi and Suman Nath. Differentially private aggregation of distributed time-series with transformation and encryption. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, pages 735–746, New York, NY, USA, 2010. ACM.

[24] Michael Saks and C. Seshadhri. Space efficient streaming algorithms for the distance to monotonicity and asymmetric edit distance. In *SODA*, pages 1698–1709, 2013.

[25] Elaine Shi, Richard Chow, T h. Hubert Chan, Dawn Song, and Eleanor Rieffel. Privacy-preserving aggregation of time-series data. In *In NDSS*, 2011.