Karlsruhe Reports in Informatics 2019,2
Edited by Karlsruhe Institute of Technology,
Faculty of Informatics
ISSN 2190-4782

Minimizing Bias in Estimation of Mutual Information from Data Streams

Vadim Arzamasov, Klemens Böhm, Ignaz Rutter

2019

KIT – University of the State of Baden-Wuerttemberg and National Research Center of the Helmholtz Association

Minimizing Bias in Estimation of Mutual Information from Data Streams

Vadim Arzamasov
Karlsruhe Institute of Technology
Karlsruhe, Germany
vadim.arzamasov@kit.edu

Klemens Böhm
Karlsruhe Institute of Technology
Karlsruhe, Germany
klemens.boehm@kit.edu

Ignaz Rutter*
University of Passau
Passau, Germany
Ignaz.Rutter@uni-passau.de

ABSTRACT

Mutual information is a measure for both linear and non-linear associations between variables. There exist several estimators of mutual information for static data. In the dynamic case, one needs to apply these estimators to samples of points from data streams. The sampling should be such that more detailed information on the recent past is available. We formulate a list of natural requirements an estimator of mutual information on data streams should fulfill, and we propose two approaches which do meet all of them. Finally, we compare our algorithms to an existing method both theoretically and experimentally. Our findings include that our approaches are faster and have lower bias and better memory complexity.

KEYWORDS

mutual information, data streams, KSG estimator, biased sampling

1 INTRODUCTION

Capturing associations in data is an important task in virtually any domain. Nowadays, data often comes in the form of streams. An example is when measurements take place continuously. An approach to estimate the dependence between a pair of streams should meet all of the following requirements:

Generality – Measure. Uses a general (non-parametric) association measure.
Efficiency. Fast and memory efficient.
Generality – Queries. Supports many query types.
High Quality. Returns good estimates of associations.

We now discuss these requirements one by one.

Generality – Measure. Various dependence measures exist, for example, Pearson and Spearman correlation coefficients, Kendall's τ and mutual information (MI). Most of them rely on parametric assumptions. For example, the first three measures only quantify monotonic dependencies. MI in turn is a very general measure, as it captures both linear and non-linear as well as monotonic and non-monotonic relationships. For two continuous random variables X, Y with joint probability density function (pdf) $f(x, y)$ and marginal pdfs $f(x)$ and $f(y)$, it is defined as:

$$I(X;Y) = \iint f(x,y) \log \left(\frac{f(x,y)}{f(x)f(y)} \right) dxdy$$

Efficiency. Various methods have been developed to estimate MI from data samples, see for example [5, 14, 16]. These methods are for static samples, and their extension to streams is not straightforward. The crucial point is that none of them allows to get rid of the individual data points — they are necessary to estimate the data distribution. However, storing all points is impossible due to the infinite nature of data streams. Since storing individual points is unavoidable, one needs to sample from a stream, to reduce complexity of the algorithm, both regarding speed and memory consumption.

Generality – Queries. Denoting the current time as t_c, a query in this article is one that asks for an MI estimate in a window $[t_1, t_2]$, $t_2 \leq t_c$ for a pair of one-dimensional real-valued streams x_t and y_t. Most existing sampling schemes that are memory-efficient, i.e., maintain a sample of a fixed size, only facilitate restricted classes of queries, as explained in Section 2 and summarized in Table 1. In general, one is interested in MI changes at any time in the past and requires a fine granularity for MI estimates in the recent past, while accepting larger windows for queries in the more remote past. In this case, one often samples non-uniformly so that recent points are more likely to be stored than older ones.

High Quality. Samples obtained from streams in a non-uniform fashion cannot be considered random. Since MI estimation is based on the estimate of the joint distribution of X and Y, non-uniform samples may lead to incorrect estimates of $f(x, y)$ and consequently of the MI.

In the rest of the paper we assume that a static MI estimator applied to the data which consists of all points of the stream in the query window outputs accurate MI estimates. This estimate is the reference. We say that an MI estimate from a stream is of high quality if it is close to the reference.

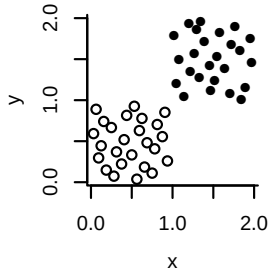

Figure 1: 50 points $p = (x, y)$ of a data stream.

Example 1. *Think of a two-dimensional data stream consisting of 50 points $p_t = (x_t, y_t)$, as shown in Figure 1. The first 25 points $p_1, \ldots, p_{25}$ come from the lower region (empty circles), and the last 25 points $p_{26}, \ldots, p_{50}$ are from the upper region (filled circles). Assume that one wants to estimate the MI in window $t = [1, 50]$. Clearly, there is an association between X and Y in this window. It manifests itself when all points $p_1, \ldots, p_{50}$ are used to estimate the MI. However, if*

*This work originated while the author was affiliated with Karlsruhe Institute of Technology

Table 1: Compliance of the approaches with query types

Approaches	Rd	Rs	Sw	Sw*
DEMI, ADEMI, DIMID				✓
Sliding window			✓	✓
Reservoir sampling		✓		
MISE	✓	✓	✓	✓
Our approaches	✓	✓	✓	✓

recent points are stored with higher probability, one will underestimate the MI. To see this, assume that only points $p_{26}, \ldots, p_{50}$ are stored — they do not resemble any association between X and Y. Hence, the MI estimate will be close to 0, i.e., no association.

As the next section will show, no currently existing approach fulfills all requirements in combination. In this paper we propose such an approach. Specifically, our contributions are as follows. (1) We formally define the notion of *uniformity on average*, which is required to deal with non-uniformity issues. (2) We design two approaches to estimate MI for streams. The first one, called point-based (PBA), is entirely new. The second one, dubbed structure-based (SBA), is a modification of an existing one, MISE [9], to comply with all requirements. (3) We provide formal guarantees on the memory consumption and the calculation effort of our approaches. As accuracy depends on the true MI value [6], attaining a general model of it is difficult to impossible; we nevertheless provide some useful qualitative results. (4) We compare the performance of our methods with MISE experimentally.

Our theoretical results are that our two methods have better upper bounds for asymptotic memory complexity and calculation effort, with PBA being better. Our experiments confirm this and show that they also yield a higher accuracy on the data sets considered. Moreover, PBA scales linearly with the number of data streams, as opposed to the quadratic complexity of MISE and SBA.

Paper outline: Both Section 2 and Section 3 cover related work. Section 2 reviews existing approaches for MI estimation from streams. Section 3 describes the static estimator of MI we use and the non-uniformity problem. Section 4 describes PBA. Section 5 describes MISE and explains why it has non-uniformities and how SBA overcomes them. Section 6 investigates PBA theoretically. Section 7 features experiments. Section 8 concludes.

2 ESTIMATION OF MI FROM STREAMS

[18] introduces two methods, DEMI and ADEMI, which speed up calculations significantly, compared to the direct usage of a static estimator. [4] provides the fast method DIMID, but according to [18] its accuracy is low. [9] proposes MISE and uses reservoir sampling and sliding window sampling as baselines.

These approaches rely on different sampling schemes and thus allow for different types of queries. Let N_{min} be the minimal number of points necessary to answer a query and t_c the current time. In the following we review common sampling schemes and respective query types.

Sliding window sampling with window size W only facilitates queries of type "Sw", i.e., $t_1 \geq t_c - W$. DEMI, ADEMI and DIMID were designed for the special case of queries of type "Sw*", with $t_2 = t_c$, $t_1 = t_2 - W$. Reservoir sampling with reservoir size R can only deal with queries of type "Rs", with $(t_2 - t_1)/t_c \geq N_{min}/R$. MISE samples according to a reciprocal function, which is an example of a bias function under biased reservoir sampling [1]. This function can cope with short windows in the recent past as well as with larger windows in the remote past. Thus, it allows for a more general query type, namely "Rd", comprising "Sw" and "Rs". To define this formally, for any query, we can compute $\Delta = (t_c - t_2)/(t_2 - t_1)$. For a query window of a fixed width, Δ increases when shifting the window further in the past. Then "Rd" is a query with $\Delta \leq \Delta_{max}$, where Δ_{max} is an exogenous parameter of MISE. MISE however does not account for non-uniform sampling and may provide low-quality MI estimates, as we will show. – Table 1 lists the compliance of the different approaches with the query types.

3 ESTIMATION OF MI FROM SAMPLES

In this section we model the process of estimating the MI of two variables from a sample. We introduce *uniform on average* sampling, necessary for reliable MI estimates from streams. For a specific static MI estimator we use, we formalize the notions of internal and external (non-) uniformities, which are deviations from *uniform on average*. These non-uniformities influence the bias of MI estimation on streams.

3.1 KSG Estimator

There are several static estimators for Mutual Information. The KSG estimator [13], which is based on nearest-neighbor information, often outperforms its competitors [10, 11, 15, 19]. In particular, it provides good accuracy also for small datasets and hence is well-suited for samples. We build our approaches on it. This subsection describes the KSG estimator. There are two versions of it; they both explore the neighborhood of a data point $p = (x, y)$ with respect to the distance function induced by the maximum norm:

$$\left\|p - p_j\right\| = \max(|x - x_j|, |y - y_j|) \tag{1}$$

In the following we stick to the first version, since it is used in MISE, allowing for a fair comparison to it. Nevertheless, the approaches we consider can also be used together with the second version. See Section E of Appendix[1].

Let D be a data set. For a data point $p_t \in D$, let $kNNdist(t)$ be the distance from p_t to its k-th nearest neighbor.

Definition 1. *Let MC_x^t be the number of points $p_{t_i} \in D$ with $|x_t - x_{t_i}| < kNNdist(t)$. We call p_{t_i} an x-marginal point of p_t in D. We call MC_x^t the x-marginal count of p_t in D. We define y-marginal points (count) analogously.*

Figure 2 illustrates this. We take $k = 1$ and consider the point p_5. The nearest neighbor of p_5 is p_3; p_7 and p_3 are x-marginal points, and p_2 and p_6 are y-marginal points. The marginal counts of p_5 are $MC_x^5 = 2$ and $MC_y^5 = 2$.

Let $\psi(\cdot)$ be the digamma function. We define:

$$I(p_t, D) = \psi(k) - \psi(MC_x^t + 1) - \psi(MC_y^t + 1) + \psi(|D|) \tag{2}$$

[1] Sections having a letter as identifier are part of the Appendix

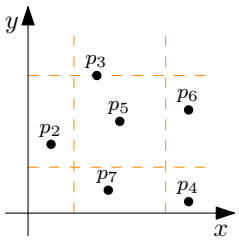

Figure 2: $kNNdist(t)$, MC_x^t **and** MC_y^t **for** $k = 1$ **and** $t = 5$.

The actual MI estimate for D is:

$$I(D) = \frac{1}{|D|} \sum_{p_t \in D} I(p_t, D) \tag{3}$$

i.e., it is obtained by averaging $I(p_t, D)$ over all points $p_t \in D$. The KSG estimator measures MI in nats: one nat equals $\log_2 e$ bits. The value k is an exogenous parameter of the estimator. The recommendation in [13] is to keep its value small, e.g., $k = 2$.

3.2 Approximation for Samples

Let *Sample* be a subset of D. From now on we assume that D is part of the data stream, i.e., contains all points of the stream in some window $[t_1, t_2]$, t_1,$t_2 \in \mathbb{N}$, ordered by the timestamp t. We assume that timestamps t are sequential natural numbers. For a point $s \in Sample$, let $Sample_s$ be a sample of D. In general, $Sample_s$ is different for different s. The main idea, used for instance in [9], is to use $I(s, Sample_s)$ as an approximation of $I(s, D)$ and

$$I(Sample) = \frac{1}{|Sample|} \sum_{s \in Sample} I(s, Sample_s) \tag{4}$$

as an approximation of $I(D)$.

Poor choices of *Sample* and $Sample_s$, as in Example 1, may yield estimations far from the reference MI value — i.e., the one which one would have obtained for the whole set D with (3). To avoid this, we require the points from different parts of D to be equally represented in *Sample* or $Sample_s$. We call this property *uniformity on average*.

Definition 2. *$Sample_u$ is formed uniformly on average from D if $\exists P \in \mathbb{R}$, $w \in \mathbb{N}$, $w \leq (|D| + 1)/|Sample_u|$: $\forall \tau_1 \geq t_1$, $\tau_2 \leq t_2$, $\tau_2 - \tau_1 = w$:*

$$\sum_{t \in [\tau_1, \tau_2]} Pr(p_t \in Sample_u) = P$$

where Pr stands for 'probability'.

Observe that this definition does not address any property of the data stream. It describes the process of obtaining $Sample_u$ from D. We need this process to leave aside the position of points in the window $[t_1, t_2]$.

Example 2. *Consider $D = \{p_1, \ldots, p_{50}\}$ from Example 1. The set D is formed uniformly on average ($w = 1$, $P = 1$). The sample consisting of all points from D with even timestamps $\{p_2, p_4, \ldots, p_{50}\}$ is formed uniformly on average ($w = 2$, $P = 1$). The sample $\{p_{26}, \ldots, p_{50}\}$ is not formed uniformly on average.*

Definition 3. *An MI estimation is internally uniform if all $Sample_s$ in (4) are formed uniformly on average from D.*

Definition 4. *An MI estimation is externally uniform if Sample in (4) is formed uniformly on average from D.*

Intuitively, 'uniform on average' sampling implies that pdf $f(x, y)$ is estimated from the entire dataset D and from subset $Sample_u$ most likely will not differ much even when it evolves over time. When the uniformity conditions are not satisfied, non-uniformities occur, making MI estimates unreliable if pdfs evolve. See Section A for more information.

4 THE POINT-BASED APPROACH (PBA)

Our point-based approach stores the incoming points in several queues. It does this so that each queue Q contains a set of points formed uniformly on average from some interval $[t_Q, t_c]$, where t_c is the current time. Then PBA uses only one queue to answer a query so that the interval $[t_Q, t_c]$ contains the query window.

4.1 Sampling: Pyramidal Time Frame

For sampling we propose to use a pyramidal time frame [2]. It is based on the notion of the order of points.

Definition 5. *Point p_t is a point of the ord-th order ($ord \in \mathbb{N} \cup \{0\}$) if t is exactly divisible by β^{ord}, where $\beta > 1$ is a fixed integer.*

A point p_t can be of several orders. For example, if $\beta = 2$, Point p_4 is of the orders 0, 1 and 2. To apply this concept to sampling in our approach, we only keep the last α points of order *ord*. The parameters α and β are exogenous. To keep things simple, we store the points in separate queues of length α for each order. This is, we allow some points to be stored in more than one queue at a time.

We now justify our choice of the pyramidal sampling scheme. We do so by showing that (1) sampling according to a reciprocal function does provide the desired properties for our setting, and (2) pyramidal sampling resembles the behavior of sampling according to a reciprocal function.

We define a sampling function $f(t_c - t)$ as the probability of a point p_t to be stored. [9] proves the following lemma:

Lemma 1. *Sampling according to a reciprocal function $f_{rec}(t_c - t) = \alpha/(t_c - t)$ provides an equal expected number of sampling elements for queries with the same value of Δ.*

This means that any sampling function decaying faster than the reciprocal function does not satisfy the requirements on our estimator, see Section 2. On the other hand, any function that decays slower is less memory efficient. Since sampling with pyramidal time is deterministic, it is natural to replace 'probability' with 'density' of points. For example, in a region where one stores points p_t so that t is exactly divisible by 8, the density will be 1/8. In general, when sampling with pyramidal time, the density is

$$d(t_c - t, \alpha) = \frac{1}{\beta^{\lceil \log_\beta \max(1, (t_c - t)/\alpha) \rceil}} \tag{5}$$

This density function decays as fast as the reciprocal function. In other words, the following holds.

Lemma 2. *$\forall \alpha \; \exists \alpha_1, \alpha_2 : \alpha_1/(t_c - t) \leq d(t_c - t, \alpha) \leq \alpha_2/(t_c - t)$, if $t_c - t > \alpha$*

Algorithm 1 PBA – answering queries

1: **data:** Qs — list of queues storing stream points
2: **function** QueryPBA(t_1, t_2, t_c)
3: $ord = \lceil \log_\beta \max(1, (t_c - t_1 + 1)/\alpha) \rceil$
4: $Sample \leftarrow \{p_t \in Qs[ord] | t \in [t_1, t_2]\}$
5: **return** $I(Sample)$
6: **end function**

All proofs are in Section D. – The lemma means that we can see the pyramidal time frame as a deterministic replacement of a probabilistic reciprocal sampling function. In other words, the pyramidal time frame is suitable for sampling.

4.2 Answering Queries

Algorithm 1 shows how PBA answers queries. Let t_c denote the current time, i.e., the time when the query arrives. To estimate MI in the window $[t_1, t_2]$, we

(1) compute the order of the queue to be used (Line 3). This order depends on t_1 and on the current time t_c;
(2) create a sample *Sample* from all points in the window $[t_1, t_2]$ stored in this queue (Line 4). Here '$Qs[ord]$' stands for the ord-th element of Qs;
(3) estimate MI with Formula (4) assuming $Sample_s = Sample$ for each $s \in Sample$ (Line 5).

The estimate obtained in this way is internally and externally uniform. Specifically, the following holds:

Lemma 3. *If D consists of all points of the data stream in the interval $[t_1, t_2]$, the sample Sample specified above is formed uniformly on average from D.*

Finally, we observe the following: As mentioned, we focus on MI estimation between two one-dimensional data streams. However, there often are many streams, much more than two, and one may be interested in the MI of each pair. In this case, it is enough to have just one PBA instance, sampling simultaneously from each data stream.

5 MISE AND THE STRUCTURE-BASED APPROACH

In this section we review MISE. We explain why it has internal and external non-uniformities and give an intuition of how SBA overcomes these issues. Due to space limitations, the full description of SBA is in Section B.

MISE uses the KSG estimator as a basis and maintains a collection of data structures, the so-called query anchors. Each query anchor QA_t maintains the list of nearest neighbors and marginal points of p_t for all possible query windows. QA_t also includes the list of points which existed at the moment it was created.

Online processing is as follows. When a new point p_t arrives,

(1) a new query anchor QA_t is created;
(2) each existing query anchor QA_{t_i} stores this point p_t if it is a new nearest neighbor of p_{t_i};
(3) QA_t in turn stores points p_{t_i}, if there currently exists query anchor QA_{t_i}, and if p_{t_i} alter the nearest neighbors or the marginal counts of p_t. This is the case, if p_{t_i} is closer to p_t than any other point stored in QA_t so far, or if p_{t_i} is a marginal point. QA_t also stores timestamps t_i of all existing QA_{t_i};
(4) Sampling removes some existing query anchors so that the timestamps of the remaining ones follow a reciprocal distribution. This is, the probability of QA_{t_i} being stored is proportional to $1/(t - t_i)$.

To answer queries, MISE uses all available information, i.e., all existing query anchors. So it does not account for the reciprocal sampling scheme it relies on. Thus, when calculating the MI, *Sample* is formed from the points p_t where $t \in [t_1, t_2]$, and QA_t exists. The query anchors QA_t exist according to the reciprocal sampling function, i.e., with increasing probability towards t_2. This means that MI estimation with MISE is externally non-uniform.

Recall that MI estimation according to Equation (4) requires several samples, referred to as $Sample_s$. In MISE, such a sample $Sample_{s_t}$ is the list of points p_{t_i} stored in QA_t which are in the query window $[t_1, t_2]$. For $t_i < t$, QA_t considers only points p_{t_i} which have existed at time t. Since the points exist according to the reciprocal probability function, $Sample_{s_t}$ is not formed uniformly on average. This is, MI estimation with MISE also is internally non-uniform.

Our structure-based approach modifies MISE to get rid of bias in MI estimates caused by non-uniformities. To do so, we adopt the idea of [8], i.e., use weights to take external non-uniformity into account. However, this idea cannot be used to cope with internal non-uniformity, since it is unclear how to account for weights in the nearest neighbor search. To get rid of internal non-uniformity, we again deploy a pyramidal time frame. This allows to obtain samples formed uniformly on average when calculating MI, similarly to PBA. It also allows us to arrive at complexity guarantees for the memory consumption of SBA. They are better than those of MISE, as we will show.

Finally, observe that to calculate MI between n one-dimensional data streams, one needs to establish $n(n-1)/2$ MISE or SBA instances. This is because these approaches do a big share of the calculations online.

6 PERFORMANCE MODEL OF THE APPROACHES

In this section we derive the memory requirements, the asymptotic maintenance speed, i.e., the time needed to add a point, and the worst case query-answering speeds for PBA. We derive the corresponding bounds for SBA and MISE in Section C. Finally, we say how to choose values of parameters α and β for PBA to meet user requirements regarding the number of points used to answer queries.

Memory Requirements. PBA stores up to α points of each order. The maximal order of the existing points at time T is $\log_\beta T$. Thus, the memory requirements of the approach are in $O(\alpha \log_\beta T)$.

Maintenance Speed. To insert a point with PBA, one needs to calculate its order and insert it into the respective queues. In the worst case, each existing queue is updated. The number of the queues as well as the time to calculate the order is in $O(\log_\beta T)$

Table 2: Characteristics of the estimators in terms of their parameters α and β and of stream length T. Me — memory consumption; Ma — maintenance complexity; Qa — query answering speed.

	PBA	SBA	MISE
Me	$O(\alpha \log_\beta T)$	$O(\alpha^2 \log^2_\beta T)$	$O(\alpha T + \alpha^2 \log^2 T)$
Ma	$O(\log_\beta T)$	$O(\alpha \log_\beta T)$	$O(\alpha \log T)$
Qa	$O(\log\log_\beta T + \alpha \log \alpha)$	$O(\alpha^2 \log^2_\beta T)$	$O(\alpha T + \alpha^2 \log^2 T)$

Query-answering Speed. For any query window, PBA uses only one of the $\log_\beta T$ queues containing α points. To find this queue, PBA performs a binary search, which takes $O(\log\log_\beta T)$ time. This search to find points inside the window then requires $O(\log \alpha)$ operations. One can easily show that the number of returned points is $n_{query} \leq \lceil \alpha/(\Delta+1) \rceil$, $\Delta = (t_c - t_2)/(t_2 - t_1)$ for query $[t_1, t_2]$ and current time t_c. Another expensive operation in our implementation is to build a kD-tree in order to speed up the k-nearest neighbor search and the counting of marginal points [17]. Building the tree takes $O(n_{query} \log(n_{query}))$ time. Note that this result is in line with Theorem 4.3 in [18].

Table 2 is a summary of PBA, SBA and MISE.

Insights Regarding Estimation Accuracy. The approximation (4) of $I(D)$ affects the accuracy of the estimator in two ways. First, the size of *Sample* affects the variance of the estimate. In case *Sample* is small, the average in (4) is calculated over a small number of terms, which leads to a high volatility. Second, the size of $Sample_s$ mainly affects the bias of the estimate. For a rather small number of points in $Sample_s$, the expected bias is likely to be high [10, 13]. Moreover, the bias depends on the true MI. The number of points required to obtain accurate estimates grows exponentially with the true MI [6, 7]. Section E demonstrates the dependence of bias and variance of the KSG estimator on the number of points.

This means that, to attain an understanding of the accuracy one can expect, one needs to establish the dependence between the parameters of the approach and the effective sizes of *Sample* and $Sample_s$. We derive this in the following.

Lemma 4. *With PBA,* $|Sample| = |Sample_s| \geq \alpha/\beta(\Delta + 1) - 1$ *as long as* $t_c - t_1 > \alpha$.

Assume now that one is interested in queries with Δ not higher than Δ_{max} and requires the sizes of *Sample* or $Sample_s$ to be at least N_{min} to estimate MI with acceptable quality. Then one should choose the parameters so that

$$N_{min} \leq \frac{\alpha}{\beta(\Delta_{max} + 1)} - 1$$

For example, assume that one is interested in MI estimation from windows with values of Δ not greater than $\Delta_{max} = 1$. This includes any window of the width of n seconds (hours, days etc.) shifted not more than n seconds (hours, days etc.) in the past, where n is an arbitrary number. Now assume that estimates based on $N_{min} = 100$ points of the stream are sufficiently accurate in a given scenario. Then PBA with $\beta = 2$ and $\alpha = 404$ provides the required accuracy.

7 EXPERIMENTS

In the following we compare the performance of PBA and SBA with MISE and with storing the entire data stream. We provide the results of additional experiments in Section F; their results do not provide significant insights additional to the ones presented here.

So far we have a general description of the algorithms, with various parameter values to be set. Specifically, one must choose the value of k (parameter of the KSG estimator) and of α for all algorithms and of β for PBA and SBA. A further degree of freedom is the choice of the data. On the other hand, we want to compare the algorithms according to the following performance measures: memory requirements, maintenance and query answering speed, and accuracy. The last two measures further require specifying the query, in terms of, say, Δ and query-window width. A complication is that each parameter affects all performance measures, and the effect usually is nonlinear.

To reduce the number of degrees of freedom, we fix some parameters values. We set $k = 1$, as this value leads to the lowest memory consumption of SBA and MISE and to the fastest query answering time for all algorithms. Next, we set $\beta = 2$, as it provides smoother sampling and again requires less memory for a similar accuracy of PBA and SBA. Then we set parameter α to 250 or 500 for SBA and MISE, depending on the data size, and to 2000 for PBA. In the plots, the value of α will come after the name of the algorithms so that there is no ambiguity. In the following we use only queries with $\Delta = 0$. We do so as it demonstrates the worst case performance of approaches.

Table 2 suggests that PBA is better in memory consumption and worst case query answering speed than SBA, and that SBA is better than MISE. Thus, the primary target of our experiments is to confirm the derived asymptotic complexity of the algorithms and to check the quality of the MI estimates they provide. To do so, we do the following: (1) We scale axes on the plots so that some complexity functions from Table 2 are straight lines in the coordinates. (2) When absolute performance values differ much, we introduce a separate vertical axis. On these plots (Figures 3–7), one should compare the curvatures of the lines – straight is better than convex. The absolute figures for algorithm performance should be taken with care, as even the same α values do not imply the same quality.

We have implemented all estimators in C++ and perform the experiments on an Intel Core i7-3520M processor at 2.90 GHz with 8 GB RAM.

7.1 Data for the Experiments

In this section we perform experiments on one real-world dataset and two synthetic ones designed specifically to demonstrate the worst case scenarios. Our real world dataset consists of columns W, F of PAMAP data[2], rows 20000–40091, with NaN values removed (20000 points).

To generate the dataset Increasing MI (20000 points), we iterate between uniform and high dependence distributions, gradually

[2] http://www.pamap.org/demo.html

increasing from 0 to 1 the probability to have a point from the latter one. These distributions are as follows:

- Uniform: we sample x_t and y_t uniformly and independently from the interval $[0, 1000]$.
- High dependence: we sample x_t and y_t uniformly from eight equal squares $[0, 125]\times[0, 125]\cup\cdots\cup[875, 1000]\times[875, 1000]$

Finally, we generate the so-called Special dataset as follows: $x_t = t$, $y_t = 1 - 10^{-5}/t$. This dataset is of length 100000.

7.2 Memory Requirements

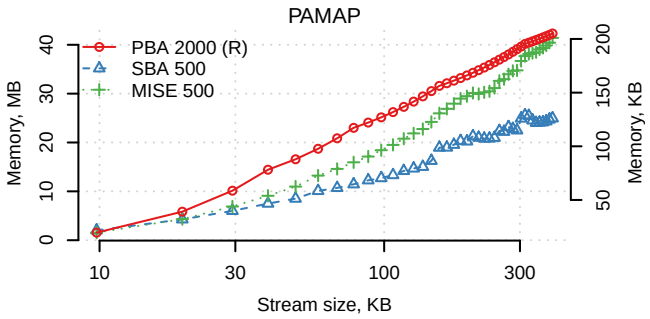

Figure 3: Memory consumption on PAMAP data.

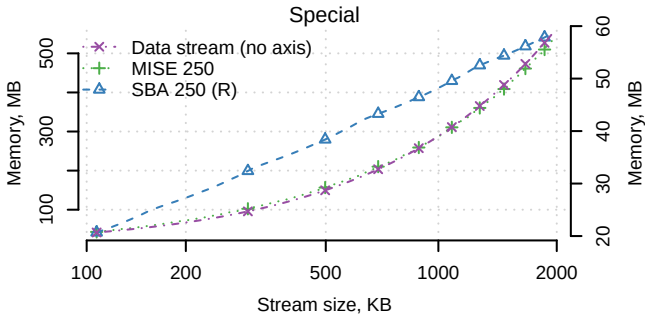

Figure 4: Memory consumption on Special data.

Figure 3 graphs the memory consumption contingent on stream size for PAMAP data. Hereafter, the lines are the median results in series of 50 experiments, and "(R)" in the legend means "right axis". We have scaled the horizontal axis logarithmically. We do this to obtain straight lines for PBA, as predicted theoretically, see Table 2.

To compare the memory consumption of MISE and SBA, we plot it as well as the one to store the entire stream for Special data (Figure 4). The horizontal axis is scaled squared logarithmically, and the SBA memory consumption is a straight line. MISE however has a convex memory-consumption curve with these axes and does not provide any benefit with Special data, compared to storing the entire stream.

One can see that experimental results are in line with those derived theoretically.

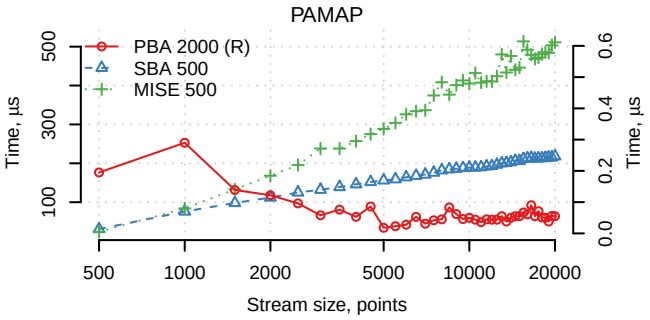

Figure 5: Maintenance speeds on PAMAP data.

7.3 Computation Speed

Figure 5 shows the maintenance speed of all three algorithms for PAMAP data. The horizontal axis is scaled logarithmically. After a stabilization period, the maintenance speeds for SBA and MISE are straight lines, as predicted by our analysis, see Table 2. This data is way too short to notice an increasing maintenance complexity of PBA. In our experiments it consumes somewhat more time in the beginning, since it needs to create new queues more frequently in an early stage. In absolute numbers, PBA with $\alpha = 2000$ is approximately 2000 times faster than SBA with $\alpha = 500$ and 5000 times than MISE with $\alpha = 500$, after 20K points.

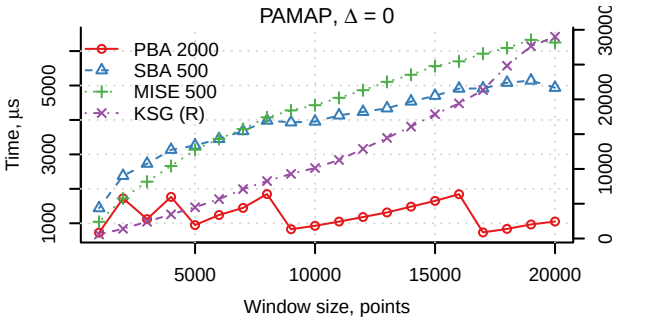

Figure 6: Query answering speeds for PAMAP data.

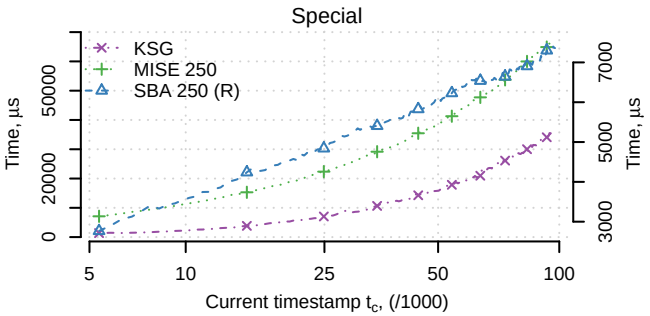

Figure 7: Query answering speeds for Special data.

Figure 6 plots query answering speeds for $\Delta = 0$ for PAMAP. The abrupt jumps in the curve for PBA reflect sudden changes in

the number of points used for MI calculation, due to changes of the queue, see Line 3 of Algorithm 1.

To compare the speed of SBA and MISE, we plot the query answering times of these methods on Special data (Figure 7). We do so for queries in the window $[1, t_c]$ ($\Delta = 0$). The horizontal axis is scaled squared logarithmically. One can see that query answering times for SBA scale almost linearly with these axes, while the curves for MISE are convex. This is in line with the last row of Table 2. Note that MISE does not provide any benefit over the static KSG estimator on the entire dataset.

7.4 Accuracy

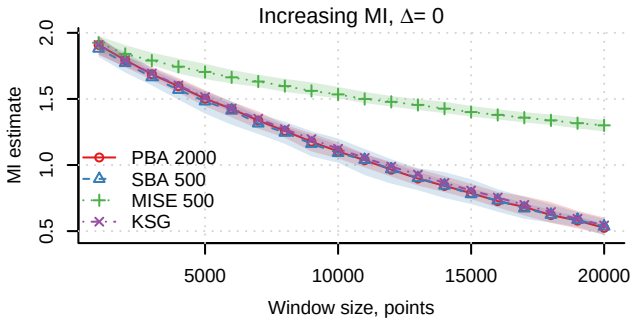

Figure 8: MI estimates on Increasing MI data.

Figure 8 shows the MI estimates obtained on the Increasing MI dataset for $\Delta = 0$ contingent on the size of the query window. Shadows indicate the areas between 10 and 90 percentiles of these estimates in a set of 50 experiments. We compare these estimates with those obtained with the KSG estimator using all points in the window — our reference point. One can see that MISE produces gross errors of up to 150%. The other estimators in turn follow the reference line quite well, not introducing bias caused by non-uniformities.

8 CONCLUSIONS

In this work, we have studied the problem of estimating mutual information on a pair of one-dimensional data streams. We have formulated a list of requirements and have shown that no existing approach satisfies them all. We have focused on the accuracy problems which occur when applying a static MI estimator to sampled data in a straightforward manner. To avoid these problems, we have defined the notion of uniformity on average, a characteristic a sample must satisfy.

We then have proposed two new approaches, PBA and SBA, to compute MI on data streams. At their core is the static KSG MI estimator, one of the best existing options. PBA and SBA satisfy uniformity on average by design. We have compared these approaches to the closest competitor, MISE, both theoretically and experimentally on synthetic and real-world datasets. In all respects, our approaches are at least comparable with MISE, which also has uniformity issues, and often are much better.

PBA is a clear winner regarding asymptotic memory consumption and worst case query answering speed. Its maintenance speed is better by up to three orders of magnitude, and it provides MI estimates close to the reference values. PBA can calculate many pairwise MI estimates and scales linearly with the number of one-dimensional streams, as opposed to the quadratic complexity of SBA or MISE, where one needs to have different SBA or MISE instances for each pair of streams.

ACKNOWLEDGMENTS

This work was partially supported by the German Research Foundation (Deutsche Forschungsgemeinschaft), Research Training Group 2153: “Energy Status Data – Informatics Methods for its Collection, Analysis and Exploitation”

REFERENCES

[1] Charu C. Aggarwal. 2006. On biased reservoir sampling in the presence of stream evolution. In *Proceedings of the 32nd international conference on Very large data bases*. VLDB Endowment, 607–618.

[2] Charu C Aggarwal, T J Watson, Resch Ctr, Jiawei Han, Jianyong Wang, and Philip S. Yu. 2003. A Framework for Clustering Evolving Data Streams. In *Proceedings of the 29th international conference on Very large data bases*, Vol. 29. 81–92. https://doi.org/10.1.1.13.8650

[3] Jose M Bernardo. 1976. Algorithm AS 103: Psi (digamma) function. *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 25, 3 (1976), 315–317.

[4] Jonathan Boidol and Andreas Hapfelmeier. 2017. Fast mutual information computation for dependency-monitoring on data streams. In *Proceedings of the Symposium on Applied Computing*. ACM, 830–835. https://doi.org/10.1145/3019612.3019669

[5] C. J. Cellucci, A. M. Albano, and P. E. Rapp. 2005. Statistical validation of mutual information calculations: Comparison of alternative numerical algorithms. *Physical Review E* 71, 6 (jun 2005), 066208. https://doi.org/10.1103/PhysRevE.71.066208

[6] Shuyang Gao, G V Steeg, and Aram Galstyan. 2015. Efficient estimation of mutual information for strongly dependent variables. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*. 277–286.

[7] Shuyang Gao, Greg Ver Steeg, and Aram Galstyan. 2015. Estimating Mutual Information by Local Gaussian Approximation. In *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence*. 278–287. arXiv:1508.00536

[8] Daniel G Horvitz and Donovan J Thompson. 1952. A generalization of sampling without replacement from a finite universe. *Journal of the American statistical Association* 47, 260 (1952), 663–685.

[9] Fabian Keller, Emmanuel Müller, and Klemens Böhm. 2015. Estimating Mutual Information on Data Streams. In *Proceedings of the 27th International Conference on Scientific and Statistical Database Management*. ACM Press, 1–12. https://doi.org/10.1145/2791347.2791348

[10] Shiraj Khan, Sharba Bandyopadhyay, Auroop R Ganguly, Sunil Saigal, David J. Erickson, Vladimir Protopopescu, and George Ostrouchov. 2007. Relative performance of mutual information estimation methods for quantifying the dependence among short and noisy data. *Physical Review E* 76, 2 (2007), 026209. https://doi.org/10.1103/PhysRevE.76.026209

[11] Justin B. Kinney and Gurinder S. Atwal. 2014. Equitability, mutual information, and the maximal information coefficient. In *Proceedings of the National Academy of Sciences*, Vol. 111. National Acad Sciences, 3354–3359. https://doi.org/10.1073/pnas.1309933111 arXiv:arXiv:1301.7745v1

[12] LF Kozachenko and Nikolai N Leonenko. 1987. Sample estimate of the entropy of a random vector. *Problemy Peredachi Informatsii* 23, 2 (1987), 9–16.

[13] Alexander Kraskov, Harald Stögbauer, and Peter Grassberger. 2004. Estimating mutual information. *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics* 69, 6 2 (2004). https://doi.org/10.1103/PhysRevE.69.066138 arXiv:cond-mat/0305641

[14] Young-Il I Moon, Balaji Rajagopalan, and Upmanu Lall. 1995. Estimation of mutual information using kernel density estimators. *Physical Review E* 52, 3 (sep 1995), 2318–2321. https://doi.org/10.1103/physreve.52.2318

[15] Angeliki Papana and Dimitris Kugiumtzis. 2008. Evaluation of mutual information estimators on nonlinear dynamic systems. *Nonlinear phenomena in complex systems* 11, 2 (2008), 225–232. https://doi.org/eprintarXiv:0809.2149 arXiv:0809.2149

[16] Marc M. Van Hulle. 2005. Edgeworth Approximation of Multivariate Differential Entropy. *Neural Computation* 17, 9 (2005), 1903–1910. https://doi.org/10.1162/0899766054323026

[17] Martin Vejmelka and Katerina Hlavackova-Schindler. 2007. Mutual Information Estimation in Higher Dimensions : A Speed-Up of a k-Nearest Neighbor Based Estimator. *Adaptive And Natural Computing Algorithms* (2007), 790–797.

[18] Michael Vollmer, Ignaz Rutter, and Klemens Böhm. 2018. On Complexity and Efficiency of Mutual Information Estimation on Static and Dynamic Data. (2018).

[19] Janett Walters-Williams and Yan Li. 2009. Estimation of Mutual Information: A Survey. In *Proceedings of the 4th International Conference on Rough Sets and Knowledge Technology*. Springer Berlin Heidelberg, 389–396. https://doi.org/10.1007/978-3-642-02962-2_49

A INFLUENCE OF NON-UNIFORMITIES

In the following, we briefly review the properties of KSG MI estimator and investigate the effects of internal and external non-uniformity separately in two studies. To demonstrate the effects of internal non-uniformity, we choose $Sample = D$ (see Section 3.2) and form $Sample_s$ without uniformity on average. Afterwards, to demonstrate external non-uniformity, we use $Sample$ formed not uniformly on average and choose $Sample_s = D$ for all $s \in Sample$.

A.1 KSG Properties

In [12], an estimator of differential entropy (an ancestor of the KSG estimator) was proven to be asymptotically consistent and unbiased if the observations are drawn independently. That is why, by using MI calculated on D as reference, we assume that D is some random sample from two-dimensional distribution, for which one wants to obtain mutual information. If this distribution evolves over time, any non-uniform on average sample from D can not be considered random anymore, violating the assumptions behind MI estimator.

A.2 The Data for the Studies

For our studies we use the distribution, shown in Figure 11. From it we generate a dataset D consisting of $4N = t_2 - t_1 - 1$ points $p_t = (x_t, y_t)$, where t_1, t_2 are as described in Section 3.2. We sample the first $2N$ points ($t_1 \le t < t_1 + 2N$) from the lower left big square, the next N points ($t_1 + 2N \le t < t_1 + 3N$) from the middle square, and the last N points ($t_1 + 3N \le t \le t_2$) from the upper right square.

A.3 Internal Non-uniformity

For this study we assume that $Sample = D$. For each point $s_t \in Sample$ we form $Sample_{s_t}$ consisting only of the points from the corresponding square, see Figure 11. Formally,

- $Sample_{s_t} = \{p_\tau | t_1 \le \tau < t_1 + 2N\}$, if $t_1 \le t < t_1 + 2N$
- $Sample_{s_t} = \{p_\tau | t_1 + 2N \le \tau < t_1 + 3N\}$, if $t_1 + 2N \le t < t_1 + 3N$
- $Sample_{s_t} = \{p_\tau | t_1 + 3N \le \tau < t_2\}$, if $t_1 + 3N \le t < t_2$

By construction, samples $Sample_{s_t}$ are not formed uniformly on average from D.

Then we calculate $I(s_t, Sample_{s_t})$ and estimate $I(Sample)$ using equation (4). We define

$$\epsilon(t) = \psi(k) - \psi(MC_x^t + 1) - \psi(MC_y^t + 1) + \psi(\eta(t)) \tag{6}$$

where MC_x^t, MC_y^t are marginal counts of s_t in $Sample_{s_t}$ and

$$\eta(t) = \begin{cases} 2N, & \text{if } t_1 \le t < t_1 + 2N \\ N, & \text{if } t_1 + 2N \le t < t_2 \end{cases} \tag{7}$$

When comparing (6), (7) to (2), one observes that $I(s_t, Sample_{s_t}) = \epsilon(t)$ in the case described. Indeed, the first three terms in right-hand sides of equations (6) and (2) coincide by definition of $I(s_t, Sample_{s_t})$; $\eta(t) = |Sample_{s_t}|$, given the description of $Sample_{s_t}$ above.

Since the mutual information of the distribution, described by any square in isolation, equals zero, we expect $\epsilon(t)$ to be around zero regardless of the timestamp t. This means that we expect the whole MI estimate $I(Sample)$ in our example to be close to zero. To demonstrate this experimentally, we set $N = 1000$, $k = 1$ and independently generate 1000 sets D. We obtain the average value $4 \cdot 10^{-4}$ of $I(Sample)$ with standard deviation 0.025.

So we have shown that internally non-uniform MI estimation can result in an MI estimate far from the real value of MI.

A.4 External Non-uniformity

To demonstrate the effect of external non-uniformity, we assume that $Sample_{s_t} = D$ for any $s_t \in Sample$. $Sample$ only consists of the points from the two upper squares (see Figure 11), i.e., is not formed uniformly on average. Formally, $Sample = \{p_\tau | t_1 + 2N \le \tau < t_2\}$.

Let s_t belong to one of the square areas in Figure 11. For high values of N it is very unlikely that the points from the other squared areas influence marginal counts of s_t in $Sample_{s_t}$. This means that $|Sample_{s_t}| = 4N$ and

$$I(s_t, Sample_{s_t}) \approx \epsilon(t) - \psi(\eta(t)) + \psi(4N),$$

where $\epsilon(t)$, $\eta(t)$ are defined in (6) and (7). We have shown previously that $\epsilon(t)$ tends to have values around zero. Using Formula 4 from [3], we can write $\lim_{N\to\infty}(\psi(4N) - \psi(\eta(t))) = \ln(4N/\eta(t))$. This means that we expect $I(s_t, Sample_{s_t})$ to be around 0.69 for $\eta = 2N$ and around 1.39 for $\eta = N$. On average, $I(s_t, Sample_{s_t})$ will be close to $1.04 = (2N \cdot 0.69 + 2N \cdot 1.39)/4N$, the true MI value for our distribution

In our case, $Sample$ does not include the points from the lower square ($t_1 \le t < t_1 + 2N$). Thus, we expect $I(Sample)$ to be close to $1.39 = 2N \cdot 1.39/2N$. To confirm this experimentally, we set $N = 1000$, $k = 1$ and generate 100 sets D independently. We obtain the average value 1.37 of $I(Sample)$ with standard deviation 0.034.

So we have shown that externally non-uniform MI estimation can yield results far from the real value.

B THE STRUCTURE-BASED APPROACH

Here we present our structure-based approach SBA inspired by MISE [9]. It relies on a new data structure called local estimator. Internally, a local estimator has procedures InsertLeft and InsertRight to collect the points of the data stream and the Query procedure to calculate an estimate of $I(p_t, D)$ (equation 2). Operations Insert and QueryMI of the SBA create new local estimators and govern the procedures of the existing ones. Finally, the Clear operation deletes certain local estimators. The only purpose of having the data structure is to store the results of preliminary calculations, which will speed up query processing.

B.1 Data Structure — Local Estimator

Our structure-based approach relies on the notion of local estimator.

Definition 6. *A local estimator (LE) is a data structure associated with some point p_t. It stores the point p_t and a sample of points required to track the history of p_t's nearest neighbors and marginal points. The procedures of a local estimator are as follows:*

(1) InsertRight(p_{t_i}), which adds a point p_{t_i} with $t_i > t$ to the sample,

Algorithm 2 InsertRight procedure in LE_t

```
1: procedure InsertRight(p_{t_i})
2: lowordR ← ⌈log_β max(1, |t − t_i|/α)⌉
3: if lowordR > maxord(p_{t_i}) then
4:     exit procedure
5: end if
6: upd ← 0
7: for all ord in {lowordR, ..., maxord(p_{t_i})} do
8:     upd ← UpdateRVD(ord, p_{t_i}) OR upd
9: end for
10: if upd = 1 then
11:     LE_t.RNN.Append(p_{t_i})
12: end if
13: UpdateRMP(LE_t)
14: end procedure
```

(2) InsertLeft(p_{t_i}), which adds a point p_{t_i} with $t_i < t$ to the sample,

(3) Query(t_1, t_2), which defines the sample subsequently referred to as $Points_t$ and returns $I(p_t, Points_t)$ calculated on it.

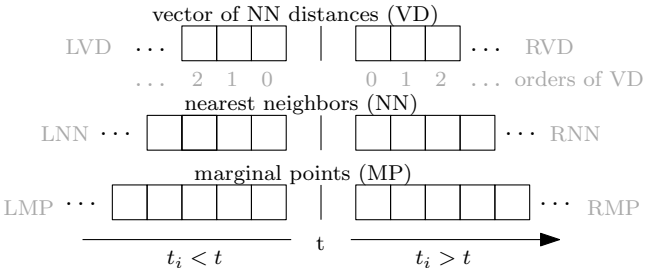

Figure 9: Structure of Local Estimator.

Figure 9 shows a local estimator for $k = 1$. It stores the nearest neighbors (NN) and marginal points (MP), in dynamic arrays. It also contains a vector of distances (VD) to the nearest neighbors of different orders. The arrays for right (forward) and left (backward) directions are separate, and we will use prefixes "R" and "L" respectively to refer to them, e.g. RMP, LMP, RVD, LVD etc.

For simplicity, we will describe SBA for $k = 1$ in the following. Its generalization for larger values of k is straightforward: one needs to change $\psi(1) \to \psi(k)$ in Query procedure (Algorithm 3) and replace vectors of distances VD with matrices of distances where the number of rows equals k.

When emphasizing that a local estimator LE is associated with point p_t, we write LE_t. We rely on the notion of order and on the pyramidal sampling to maintain local estimators (cf. Section 4.1). That is, LE_t is of order ord if p_t is of this order; and we keep only the last α estimators of each order. In the following, $maxord(p)$ denotes the maximal order of p.

B.2 Local Estimator — procedures

In this section we describe the procedures InsertRight, InsertLeft and Query of the local estimator, which collect the points of the stream and calculate an estimate of $I(p_t, D)$ (equation 2), respectively. These methods are used by operations Insert and QueryMI of the SBA, which we describe in the next section.

InsertRight (Algorithm 2) is invoked in each existing local estimator LE_t each time a new point p_{t_i} arrives in the stream. It first computes the auxiliary value $lowordR$ (Line 2) and updates the vector of distances in the right direction RVD (Lines 7–9). The value $lowordR$ controls the memory consumption of the SBA. It increases with $|t - t_i|$ so that for timestamps t far in the past, p_{t_i} can be stored in LE_t only if $maxord(p_{t_i})$ is sufficiently high. That is, the points p_{t_i}, which are far from p_t in time, will rarely lead to any update. The function UpdateRVD (Line 8) sets the ord-th element of RVD to $\|p_t - p_{t_i}\|$ and returns 1 if the distance from p_t to p_{t_i} is less than the current value of the ord-th element of RVD, or if the element is empty. Otherwise, the function does not make any update and just returns 0.

Next, if any of the elements has been updated, InsertRight appends the point p_{t_i} to the array of nearest neighbors RNN (Line 11). Finally, UpdateRMP adds the point to the array of marginal points RMP if necessary (Line 13). In particular, let $MaxDist$ denote the greatest value of elements of RVD in the positions $\{lowordR, \ldots, maxord(p_{t_i})\}$. Then the point p_{t_i} is added to the marginal points array RMP of LE_t if either $|x_t - x_{t_i}| \leq MaxDist$ or $|y_t - y_{t_i}| \leq MaxDist$.

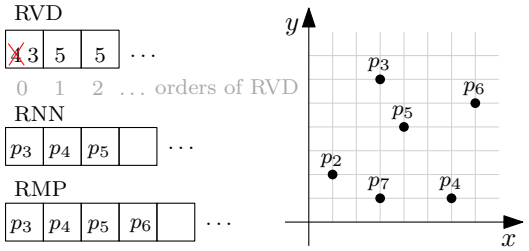

Figure 10: Procedure InsertRight at work.

The following example illustrates the process.

Example 3. *We set $k = 1$, $\alpha = 3$ and $\beta = 2$ and demonstrate how InsertRight at LE_2 associated with point p_2 fills the arrays of the local estimator as the stream progresses from $t = 3$ to $t = 8$ (Figure 10). At $t = 2$ all right arrays are empty*

(1) For p_3 $\|p_2 - p_3\| = 4$, $maxord(p_3) = 0$ and $lowordR = 0$, thus InsertRight sets the value of RVD in Position 0 to 4 and appends p_3 to RNN and RMP.

(2) $maxord(p_4) = 2$, and the distance from p_4 to p_2 is 5. InsertRight sets the values of the element of RVD in Positions 1 and 2 to 5 and adds p_4 to RNN and RMP.

(3) $maxord(p_5) = 0$. The first element of RVD is set to $\|p_2 - p_5\| = 3$, and p_5 is added to RNN and RMP.

(4) At this step, $lowordR$ increases to 1. For Point p_6, $maxord(p_6) = 1$. As $\|p_2 - p_6\| = 6$ is greater than 5, the value of RVD in Position 1, RVD and RNN are not updated. As $|y_2 - y_6| = 3 < 5$, UpdateRMP adds p_6 to the array of marginal points RMP.

(5) $maxord(p_7) = 0$, and $lowordR = 1$. The procedure exits at Line 4.

Algorithm 3 QUERY procedure in LE_t

1: **function** QUERY(t_1, t_2)
2: $ord \leftarrow \lceil \log_\beta \max(1, |t - t_1|/\alpha, |t - t_2|/\alpha) \rceil$
3: $Points_t \leftarrow \{p_{t_i} \in LE_t | t_i \in [t_1, t_2], maxord(p_{t_i}) \geq ord\}$
4: $\{MC_x, MC_y\} \leftarrow$ MARGINALCOUNTS($p_t, Points_t$)
5: $SS \leftarrow \lfloor t_2/\beta^{ord} \rfloor - \lceil t_1/\beta^{ord} \rceil + 1$
6: **if** $maxord(p_t) < ord$ **then**
7: $SS \leftarrow SS + 1$
8: **end if**
9: $LMIE_t \leftarrow \psi(1) - \psi(MC_x + 1) - \psi(MC_y + 1) + \psi(SS)$
10: **return** $LMIE_t$
11: **end function**

(6) Finally, the CLEAR operation deletes LE_2 at time $t = 8$ as SBA keeps only the last α estimators of each order (see Section B.3). For the order 1 at $t = 8$ these would be LE_4, LE_6 and LE_8.

INSERTLEFT works in the same way with arrays in the left direction. It is called in each newly created LE_t for every already existing LE_{t_i} as we show in Section B.3.

Procedure QUERY (Algorithm 3) works as follows. For a given query window with boundaries t_1 and t_2 and local estimator LE_t, it calculates the order ord of points to be used (Line 2). Then it forms the sample $Points_t$ consisting only of points of order ord, which are in the query window and are stored in LE_t (Line 3). Next, the procedure finds the nearest neighbor and computes x- and y-marginal counts of p_t in $Points_t$ (Line 4) and the number SS of points of order ord in the query window (Line 5). Finally, it calculates and returns $LMIE_t$ based on these values (Line 9), which we use as approximation of $I(p_t, D)$ (equation 2). Note that the QUERY procedure implicitly contains p_t in the sample used to calculate $LMIE_t$. That is why it increases the value of SS by one, if p_t is not the point of the order ord (Line 7).

B.3 SBA Framework

SBA internally stores a set of local estimators, and there is the CLEAR function that keeps updating them. This function simply deletes the superfluous local estimators so that only the last α ones of each order are stored. In addition, the approach provides operations INSERT and QUERYMI. The Insert operation adds a data point from the stream to the set of local estimators ($SetOfLEs$); the QUERYMI operation returns the MI estimate for a defined window. See Algorithm 4 and Algorithm 5, respectively.

INSERT first creates a new local estimator associated with the incoming point p_t. Then it performs forward and backward initialization by invoking INSERTRIGHT of existing LEs and INSERTLEFT of the new local estimator (Lines 4–5). Finally, it invokes the CLEAR operation. To process the points one by one with the procedure INSERTLEFT in LE_t we need them to be sorted by t_i in descending order. That is, the cycle (Lines 3–6) begins with larger timestamps and ends with lower ones.

The QUERYMI operation has three parameters: the boundaries of query window t_1 and t_2 to calculate Mutual Information and the current time t_c. It first determines the local estimators contained in the query window (Line 2). Then it queries each local estimator in the window to obtain the $LMIE$ (Line 5). After that the operation

Algorithm 4 INSERT operation

1: **procedure** INSERT(p_t)
2: $LE \leftarrow$ new local estimator at p_t
3: **for all** $LE_{t_i} \in SetOfLEs$ **do**
4: LE_t.INSERTLEFT(p_{t_i})
5: LE_{t_i}.INSERTRIGHT(p_t)
6: **end for**
7: $SetOfLEs \leftarrow SetOfLEs \cup \{LE_t\}$
8: CLEAR($SetOfLEs$)
9: **end procedure**

Algorithm 5 QUERYMI operation

1: **function** QUERYMI(t_1, t_2, t_c)
2: $inWindow \leftarrow \{LE_t \in SetOfLEs | t \in [t_1, t_2]\}$
3: $estimates \leftarrow \{\}$
4: **for all** $LE \in inWindow$ **do**
5: $LMIE \leftarrow LE$.QUERY(t_1, t_2)
6: $estimates$.APPEND($LMIE$)
7: **end for**
8: $weights \leftarrow$ GETWEIGHTS($inWindow, t_c$)
9: **return** WEIGHTEDMEAN($estimates, weights$)
10: **end function**

calculates weights (Line 8). Specifically, the function GETWEIGHTS calculates a weight for each $LE_t \in inWindow$ with the formula

$$weight_t = \beta^{\lceil \log_\beta \max(1, |t_c - t|/\alpha) \rceil}.$$

The next section describes the rationale behind this equation. Finally, QUERYMI returns the weighted mean of the $LMIE$s.

B.4 Uniformity of SBA

An analogue for *Sample* (4) in SBA is the set of points associated with LEs in $inWindow$ (Algorithm 5, Line 2). To avoid external non-uniformity, we could have used only LEs of specific order like in PBA. Nevertheless, we expect that using all LEs in the query window will provide more robust results, and weights will allow us to cope with external non-uniformity. Essentially, we use Horvitz–Thompson estimator [8]. The idea behind this is that local estimators of lower orders have more densely grouped timestamps. Weights of their $LMIE$ values are then inversely proportional to their density. We will demonstrate that SBA provides accurate results by means of experiments.

Given a fixed $p_t \in [t_1, t_2]$, consider sample $Points_{ord}$ consisting of all points of the data stream of order ord in $[t_1, t_2]$, where ord is as in Line 2 of Algorithm 3. Next, let MC_x^{ord} and MC_y^{ord} be the marginal counts of p_t in $Points_{ord}$. In the following we show that $Points_{ord} \cup p_t$ is the effective analogue of $Sample_s$ (4) in SBA.

LEMMA 5. *Assume that the procedure QUERY(t_1, t_2) of some LE_t calculates internally the values MC_x, MC_y and SS (Algorithm 3, Lines 4–8). Then $MC_x = MC_x^{ord}$, $MC_y = MC_y^{ord}$ and $SS = |Points_{ord} \cup \{p_t\}|$*

proof is in Section D. Since $Points_{ord}$ is formed uniformly on average ($w = \beta^{ord}$, $P = 1$), and $Sample_s$ differs from it by at

most one point. Thus, it is clear that SBA copes with internal non-uniformity.

C PERFORMANCE MODEL OF SBA AND MISE

In this section we provide upper bounds of the memory requirements for our algorithm SBA. We derive such a bound for MISE as well, since the respective publication does not feature such a result. Next, we derive the asymptotic maintenance and the worst case query-answering speeds of both algorithms. Finally, we say how to choose values of parameters α and β for SBA to meet user requirements regarding the number of points used to answer queries, in the same way we have done in Section 6,

C.1 Memory Requirements

Similarly to PBA, SBA stores up to α local estimators of each order ($O(\alpha \log_\beta T)$). Each local estimator stores internally α points of each order at most in every (right or left) direction — $O(\alpha \log_\beta T)$ in total. The memory requirements of SBA are in $O(\alpha^2 \log^2_\beta T)$.

[9] features only a partial complexity analysis of MISE. They show that the expected number of structures, the so-called query anchors, is in $O(\alpha \log T)$. Each query anchor stores internally the history of k-nearest neighbors and marginal points. It also stores the timestamps of all other query anchors which existed at creation time of the anchor. This means that MISE requires at least $O(\alpha^2 \log^2 T)$ memory to store timestamps. As the worst case, think of the stream $p_t = (t, 1/t)$. In this case, each query anchor stores all subsequent points, which become visible as marginal points after the query anchor has been saved. Then the entire algorithm requires the following amount of memory at time T:

$$\sum_{j=1}^{\alpha} j + \sum_{j=\alpha}^{T} j \cdot \frac{\alpha}{j} \sim O(\alpha T)$$

For MISE, since the sampling is probabilistic, this is an expected value. This means that there is no guarantee that MISE asymptotically ($T \to \infty$) consumes less space than the data stream itself.

C.2 Maintenance Speed

In SBA, the Clear operation function iterates through all local estimators in SBA, the number of which is $O(\alpha \log_\beta T)$. Similarly, the complexity of updates of NN and MP arrays take $O(\alpha \log_\beta T)$ time. Consider the complexity of the Insert operation in VD arrays. By construction, the InsertLeft procedure affects each position in the LVD array of the new local estimator at most α times. The number of positions in this array is restricted by the highest order of the existing local estimators, which is in $O(\log_\beta T)$. As InsertRight works symmetrically, a point of order 0 influences only the last α *LE*s and one position of RVD in each local estimator; a point of order 1 affects the last 2α *LE*s: two positions of RVD in the last α local estimators and one position in preceding *LE*s, etc. At the same time, the probability that the point is of order *ord* equals $1/\beta^{ord}$. This gives us the amortized complexity of InsertRight.

$$\sum_{i=1}^{\lfloor \log_\beta T+1 \rfloor} \frac{\alpha(1+2+\cdots+i)}{\beta^{i-1}} \in O(\alpha \log_\beta T)$$

The amortized insert complexity of a query anchor in MISE is $O(1)$, and the number of query anchors is $O(\alpha \log T)$. The Sampling function iterates through all query anchors. Thus, the MISE maintenance complexity is $O(\alpha \log T)$.

C.3 Query-answering Speed

In the same way as PBA, SBA and MISE spend $O(\log(\alpha \log T))$ time to find data structures (local estimators or query anchors respectively) inside the query window. In the worst case, the whole set of data structures is returned, and the Query procedure inside each data structure goes through all marginal points saved. The complexity of this search (in all data structures) is proportional to the memory requirements of SBA or MISE, i.e., is bounded with $O(\alpha^2 \log^2_\beta T)$ for SBA and with $O(\alpha T + \alpha^2 \log^2 T)$ for MISE.

In principle, the answering speed for SBA can be improved as follows. First, to limit the number of *LE*s in a query window for $\Delta = 0$, one can use only *LE*s of a given order (like in PBA), instead of weights. Second, one can store points of different orders in separate arrays within *LE*s. It will result in $O(\alpha \log \log_\beta T + \alpha^2)$ time for answering queries and will not affect asymptotic memory complexity or maintenance speed. We will not consider this optimization in our experiments though.

C.4 Insights Regarding Estimation Accuracy

SBA stores *LE*s in the same way as PBA stores points. It then uses all local estimators in the query window to provide the MI estimate. Thus, $\alpha/\beta(\Delta+1)-1$ is the lower bound of the size of *Sample* with SBA as well as in PBA (Lemma 4). In the following, we show that it is also the lower bound of $Sample_s$.

Lemma 6. *With SBA,* $|Sample_s| \geq \alpha/\beta(\Delta+1)-1$ *as long as* $t_c - t_1 > \alpha$.

Given this, the guideline for choosing the values α and β presented in Section 6 applies to SBA as well.

D PROOFS

Lemma 2. $\forall \alpha\ \exists \alpha_1, \alpha_2 : \alpha_1/(t_c-t) \leq d(t_c-t,\alpha) \leq \alpha_2/(t_c-t)$, *if* $t_c - t > \alpha$

Proof. For $\alpha_1 = \alpha/\beta$ and $\alpha_2 = \alpha$ and using (5):

$$\frac{\alpha_1}{t_c-t} = \frac{1}{\beta^{\log_\beta(\frac{t_c-t}{\alpha})+1}} < d \leq \frac{1}{\beta^{\log_\beta(\frac{t_c-t}{\alpha})}} = \frac{\alpha_2}{t_c-t},$$

where $d = d(t_c - t, \alpha)$ □

Lemma 3. *If D consists of all points of the data stream in the interval* $[t_1, t_2]$, *the sample Sample specified above is formed uniformly on average from D.*

Proof. Note that if *Sample* consists of all points of the order *ord* in *D*, then *Sample* is formed uniformly on average from *D* ($w = \beta^{ord}$, $P = 1$).

1) If $t_1 \geq t_c - \alpha + 1$, Sample includes points from the queue of order 0, which contains all points of the interval $[t_1, t_2]$.

2) If $t_1 < t_c - \alpha + 1$, assume that $\exists p_t \in D$, $p_t \notin Sample$, p_t is of order *ord* (Line 3 of Algorithm 1). This means that the interval

$[t_1, t_c]$ contains more than α points of order ord. That is,

$$\frac{t_c - t + 1}{\beta^{ord}} > \alpha \implies ord < \log_\beta \frac{t_c - t + 1}{\alpha}$$

which contradicts equation in Line 3 of Algorithm 1. □

Lemma 4. *With PBA,* $|Sample| = |Sample_s| \geq \alpha/\beta(\Delta + 1) - 1$ *as long as* $t_c - t_1 > \alpha$.

Proof. Equation in Line 3 of Algorithm 1 yields the order of the points to use for the estimation. The window $[t_1, t_2]$ contains

$$\lfloor \frac{t_2 - t_1}{\beta^{\lceil \log_\beta \max(1,(t_c - t_1)/\alpha) \rceil}} \rfloor \geq \frac{t_2 - t_1}{\beta^{\log_\beta((t_c - t_1)/\alpha)+1}} - 1 =$$

$$\frac{\alpha}{\beta(\Delta + 1)} - 1$$

points □

Lemma 5. *Assume that the procedure* Query(t_1, t_2) *of some* LE_t *calculates internally the values* MC_x, MC_y *and* SS *(Algorithm 3, Lines 4–8). Then* $MC_x = MC_x^{ord}$, $MC_y = MC_y^{ord}$ *and* $SS = |Points_{ord} \cup \{p_t\}|$

Proof. The number of points of the order ord in the Region $(0, t_2]$ equals $\lfloor t_2/\beta^{ord} \rfloor$; in Region $(0, t_1)$: $\lceil t_1/\beta^{ord} \rceil - 1$. Thus, the number of points of order ord in the Region $[t_1, t_2]$ equals $\lfloor t_2/\beta^{ord} \rfloor - \lceil t_1/\beta^{ord} \rceil + 1$ and $SS = |Points_{ord} \cup \{p_t\}|$ (compare to Line 5 of Algorithm 3).

Next, the InsertLeft and InsertRight procedures store any point which changes the nearest neighbors or is a marginal point of any order, defined in Algorithm 2 Line 7. Thus, to prove that $MC_x = MC_x^{ord}$ and $MC_y = MC_y^{ord}$ it is sufficient to show that Line 4 of Algorithm 2 was not executed for any point of Order ord in Region $[t_1, t_2]$. As procedures work in a symmetric way, w.l.o.g., we consider only InsertRight and the Region $(t, t_2]$. Assume, that $\exists p_{t_i}$, $t_i \in (t, t_2]$ for which Line 4 of the Algorithm 2 have been executed. This means that

$$\lceil \log_\beta \max(1, |t - t_i|/\alpha) \rceil > ord$$

Given the definition of ord (Algorithm 3 Line 2), this means that $|t - t_i| > |t - t_2|$, i.e., $t_i \notin (t, t_2]$, which contradicts our assumption □

Lemma 6. *With SBA,* $|Sample_s| \geq \alpha/\beta(\Delta + 1) - 1$ *as long as* $t_c - t_1 > \alpha$.

Proof. According to Lemma 5, $|Sample_s| \geq |Points_{ord}|$

$$|Points_{ord}| = \lfloor \frac{t_2 - t_1}{\beta^{\lceil \log_\beta \max(1, |t - t_1|/\alpha, |t - t_2|/\alpha) \rceil}} \rfloor \geq$$

$$\frac{t_2 - t_1}{\max(1, \beta(t_2 - t_1)/\alpha)} - 1 \geq \frac{\alpha}{\beta(\Delta + 1)} - 1$$

□

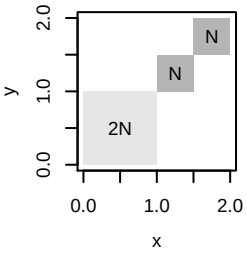

Figure 11: Data distribution for the case study. Dark regions have twice the probability density of the light region.

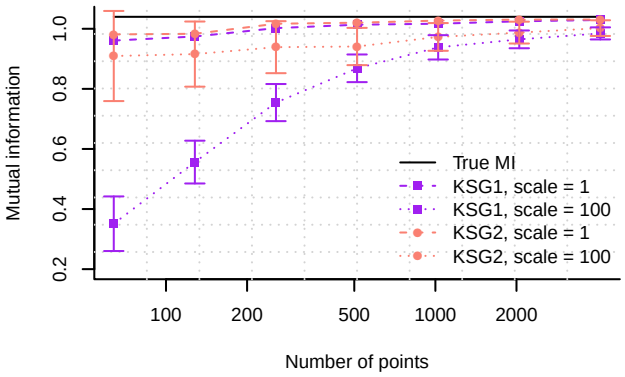

Figure 12: MI estimates obtained with the first (squares) and the second (circles) versions of the KSG estimator in dependence on a number of points.

E COMPARING KSG ESTIMATOR VERSIONS

In the following we demonstrate the relative performance of the two versions of the KSG estimator [13] depending on the relative scales of variables and number of points. To this end, we use the synthetic distribution proposed in [11], shown in Figure 11. In the figure, dark blocks represent twice the probability density of the light block. The true Mutual Information for this distribution is $I \approx 1.04$ nats.

Example 4. *We generate the dataset D from the probability distribution just described. Then we scale the x values by multiplying with 100 and calculate MI using both versions of the KSG estimator on the resulting dataset* D_{scaled}*. We perform the procedure for different numbers of points in D and for 200 independently generated datasets D of each size. In Figure 12, the horizontal axis (log scale) stands for the size of* D_{scaled} *and the vertical axis for the MI estimate. The bars on the lower lines are proportional to the standard deviation of respective estimates.*

When relative scales of variables differ significantly, the second version (KSG2) converges faster to the true value. This is because the scaling affects the values of marginal counts MC_x and MC_y in the second version only if it changes the nearest-neighbor relation. In the first version (KSG1) in turn, values may change even if the nearest neighbor does not. At the same time the first version has lower variance. These results have been featured in [13] as well.

Table 3: Description of the real world datasets

Name	Modification	Period	Length
IBMGE[3]	Revenues	02.01.1962–08.03.2016	13639
USDRUB[4]	Revenues	30.06.2014–01.03.2017	15260
PAMAP[5]	Differences	rows 20000–40091[6]	20000

Table 4: Synthetic datasets types

Name	Number of points
Uniform High dependence Periodic 100, 1000, 5000 Increasing/Decreasing MI	20000
Uniform long Special	100000

F ADDITIONAL EXPERIMENTS

In this section we provide additional experimental results. We perform our experiments for a variety of datasets, different parameter values of the approaches and queries.

F.1 Data

In our experiments, we use three real-world and various types of synthetic datasets. We convert these datasets to differences ($x_t = a_{t+1} - a_t$) or revenues ($x_t = a_{t+1} - a_t)/a_t$, where a_t is the original time-series. Table 3 is a summary of the real-world datasets.

Table 4 lists the types of synthetic data. To form the Uniform dataset, we sample x_t and y_t uniformly and independently from the interval $[0, 1000]$. For the High dependence dataset, we sample x_t and y_t uniformly from eight equal squares $[0, 125]\times[0, 125]\cup \cdots \cup [875, 1000]\times[875, 1000]$. To generate the dataset Increasing MI, we iterate between uniform and high dependence distributions, gradually increasing from 0 to 1 the probability to have a point from the latter one. The Decreasing MI dataset is just the inverse of the previous one. In Periodic X datasets we periodically increase and decrease the probability to have a point from the high dependence distribution with period $2X$, where $X = 100, 1000$, or 5000. We also generate the Uniform Long dataset from the uniform distribution, but it consists of more points.

F.2 Memory Requirements

Figure 23 graphs the memory requirements with the stream size for the short streams and all three algorithms. We have scaled the horizontal axis on the left panel logarithmically and on the middle and right panel squared logarithmically. Memory consumption

[6]Stocks IBM and GE, obtained from https://finance.yahoo.com/
[6]Pair USD/RUB and Brent, obtained from https://www.finam.ru/
[6]Subject102, columns W, F, obtained from http://www.pamap.org/demo.html
[6]excluding NaN values

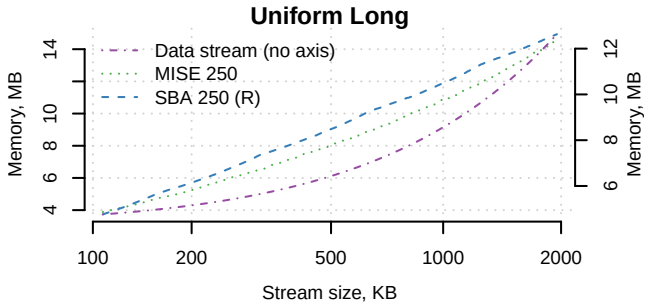

Figure 13: Memory consumption on Uniform Long data.

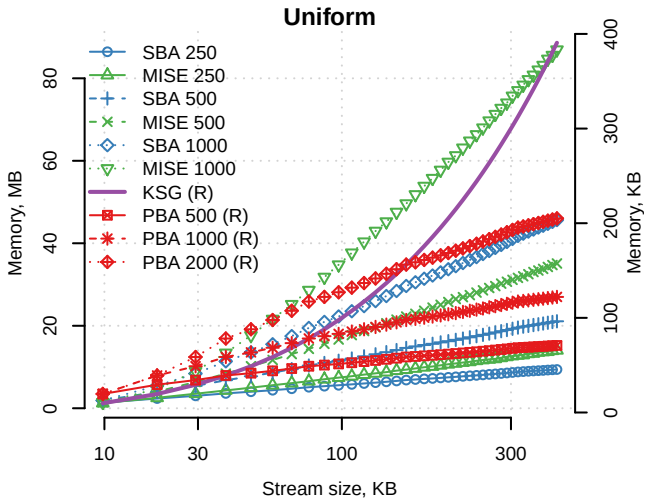

Figure 14: Memory consumption of on Uniform data.

of PBA does not depend on the data and is in $O(\log T)$. SBA and MISE consume the least memory for the High dependence data stream and require more for the Uniform or the real world data. The memory requirement of SBA is in $O(\log^2 T)$. MISE with the same α value consumes more memory than SBA and has worse memory complexity at least for some data. We also plot the memory consumption of SBA and MISE on Uniform long dataset as well as the memory requirements to store the entire dataset (Figure 13). The horizontal axis is scaled squared logarithmically, and the SBA memory consumption is a straight line here. MISE however has a convex memory consumption curve with these coordinates.

Finally, Figure 14 graphs the memory consumption of all algorithms for different α on the Uniform dataset. The horizontal axis is scaled squared logarithmically. As expected, a greater α leads to a higher memory consumption of any algorithm. For the same α, SBA consumes less memory than MISE. PBA needs more than 100 times less memory than SBA and has a better complexity.

F.3 Computation Speed

Figures 15, 16 show the maintenance speed of SBA and MISE. Horizontal axes are scaled logarithmically. The maintenance speeds are

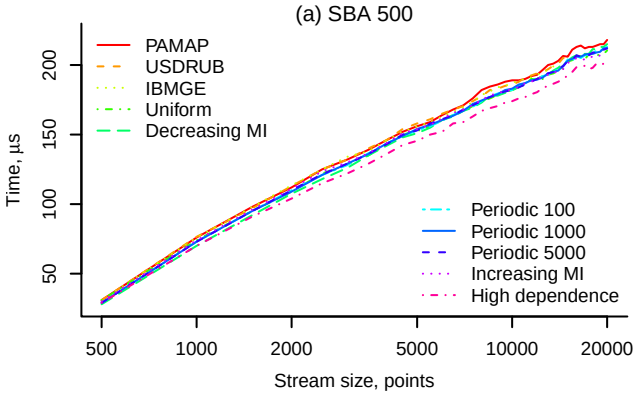

Figure 15: Insertion speed of SBA various data.

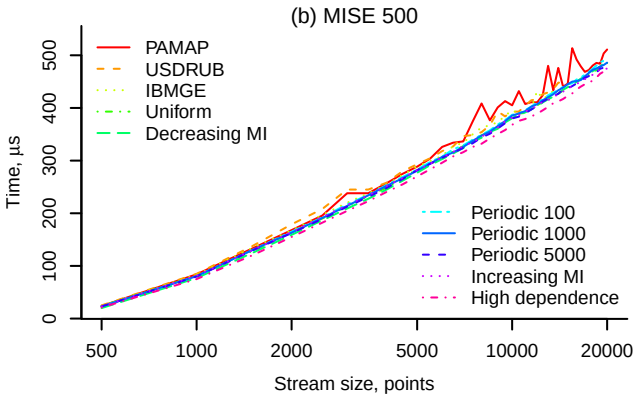

Figure 16: Insertion speed of MISE various data.

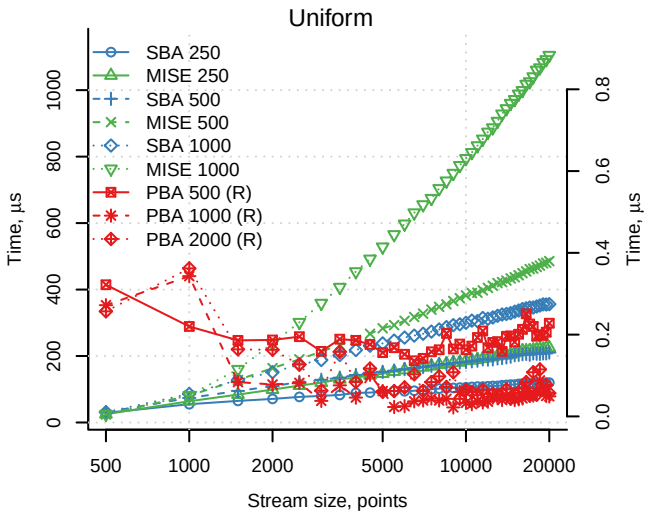

Figure 17: Insertion speed on Uniform data.

straight lines for both the synthetic and the real-world data, as predicted by our analysis, see Table 2. For the same α, SBA maintains its data structures faster than MISE. PBA insertion speed does not depend on the data. Figure 17 graphs it for different values of α, together with the MISE and SBA curves for the Uniform dataset.

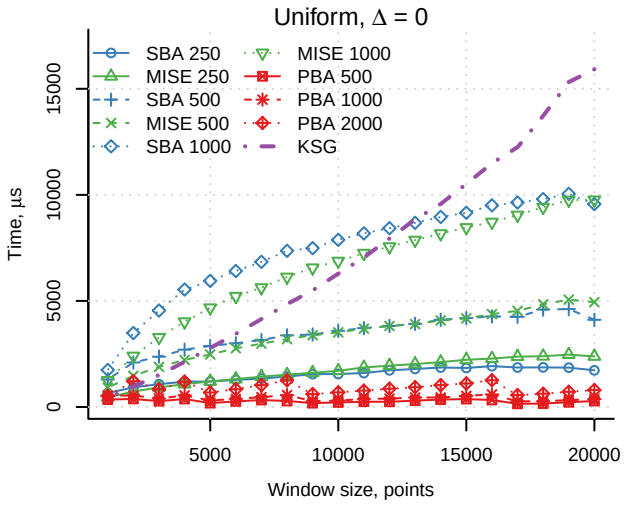

Figure 18: Query answering speed on Uniform data stream, $\Delta = 0$.

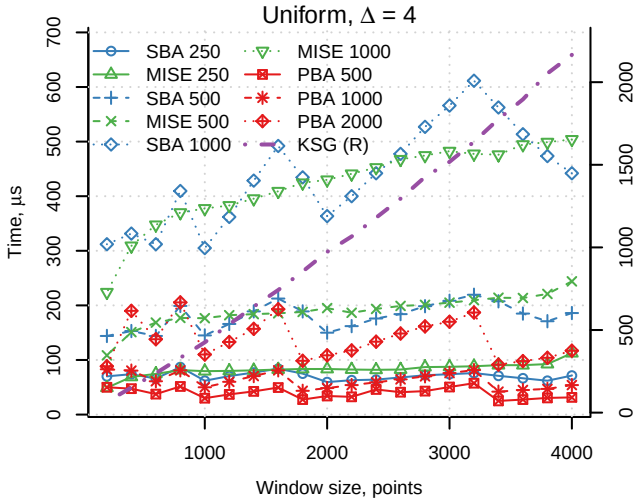

Figure 19: Query answering speed on Uniform data stream, $\Delta = 4$.

Figures 24, 25 plot query answering speeds for $\Delta = \{0, 4\}$ for short datasets. In general, PBA is faster for $\Delta = 0$ and as fast as SBA and MISE on average for Δ. MISE and SBA perform comparably in both cases, with SBA being slightly faster for Δ. As with memory requirements, SBA and MISE answer queries faster on the High dependence dataset than on the Uniform or the real world datasets.

Figures 18, 19 graph query answering times on the uniform data stream for the three algorithms for $\Delta = \{0, 4\}$ and different values of α. One can see that for a window size greater than 13K points

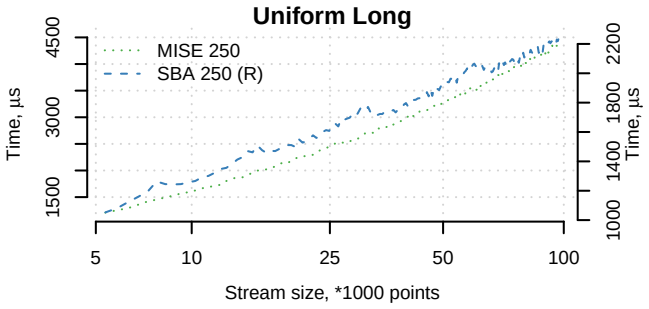

Figure 20: Query answering speed on Uniform Long data.

already, any algorithm is faster than using the static KSG estimator on the raw data.

Finally, to compare the speed complexity of SBA and MISE, we plot the query answering times for these methods on Uniform long dataset (Figure 20). We do so for queries in the window $[1, t_c]$ ($\Delta = 0$). The horizontal axis is scaled squared logarithmically. One can see that query answering time for SBA scales almost linearly with these axes, while the curves for MISE are convex. This is in line with the last row of Table 2.

F.4 Accuracy

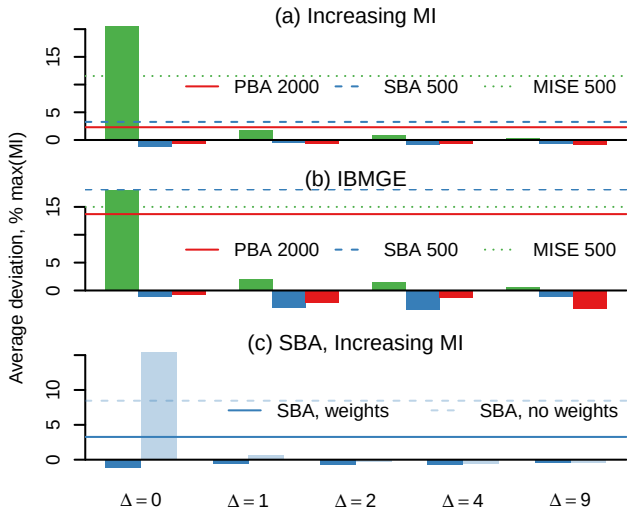

Figure 21: Accuracy on Increasing MI (a) and IBM/GE (b) data. (c) — comparison of the SBA algorithm with and without weights for Increasing MI data.

We first explain how we investigate the accuracy of the estimators. Let t_{max} be the length of a certain dataset. For a given delta we then create query windows $[t_1, t_2]$, where $t_1 = 1, \ldots, \lfloor t_{max}/1000 \rfloor \cdot 1000$, and we choose t_2 to provide the required Δ value; $t_2 = \lfloor (t_{max} + t_1\Delta)/(\Delta+1) \rfloor$. Then we obtain MI estimates for all datasets and windows with PBA, SBA and MISE as well as with the static KSG estimator, our reference point. Finally, we compute the average

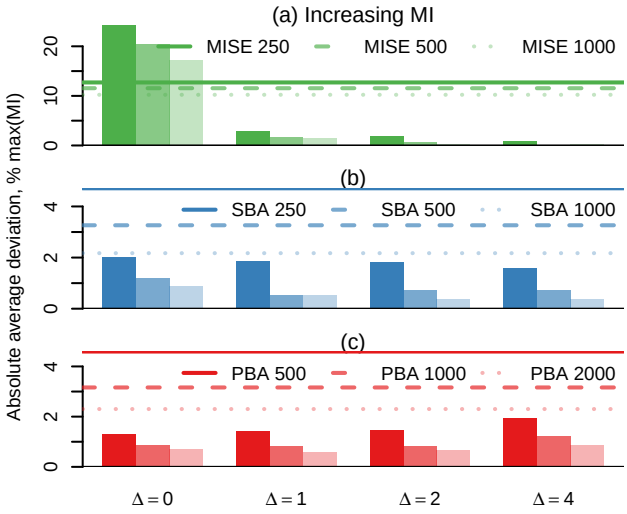

Figure 22: Accuracy on Increasing MI data for different values of α.

deviation of the result of the stream estimators from the one of KSG and standard deviations of these differences.

Figure 21(a, b) features the average deviations in relative terms, i.e., as the share of maximal MI obtained with the static KSG estimator. Lines stand for standard deviations for $\Delta = 0$. For $\Delta = 0$, one can see that MISE estimates MI with a strong and statistically significant bias. This is not the case for SBA or PBA. Figure 21(c) compares SBA with and without weights. Clearly, weights reduce the bias of this algorithm.

Bars in Figure 22 represent the decrease of the bias of the MI estimate with an increased α for all values of Δ and for each algorithm. As before, lines stand for standard deviations for $\Delta = 0$. Even though the bias with MISE decreases with the increased α, it remains statistically significant.

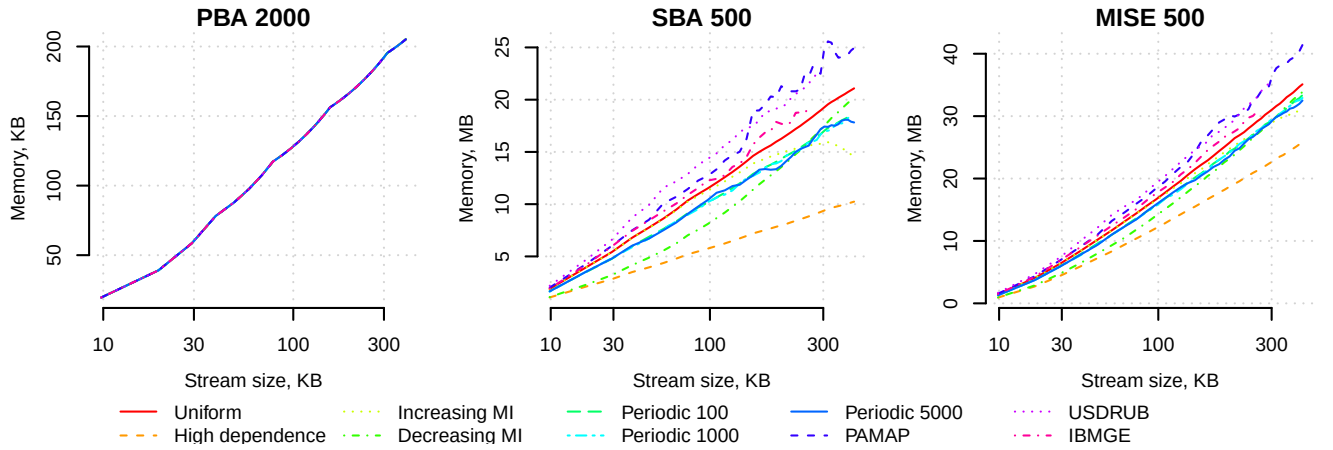

Figure 23: Memory consumption on various data.

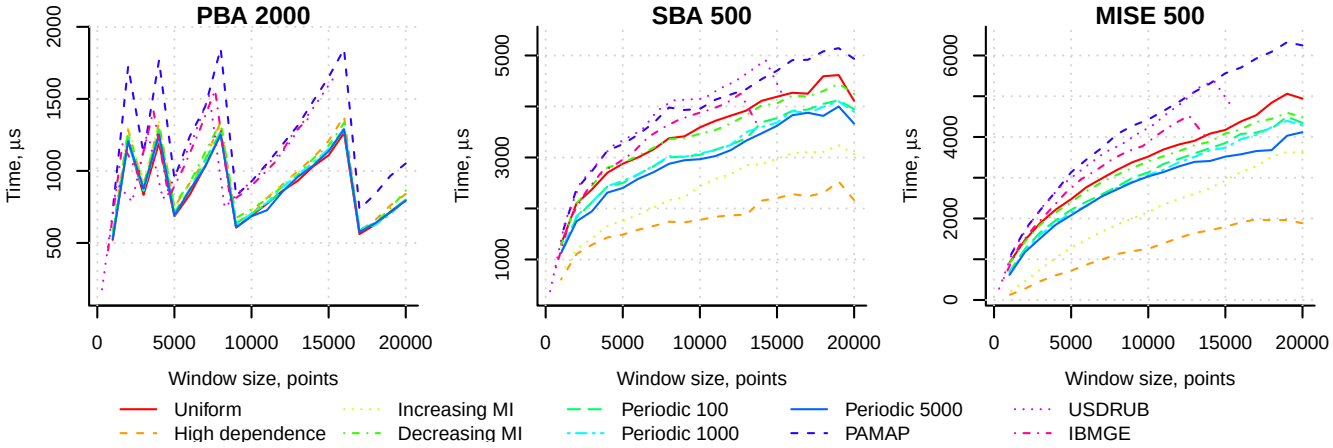

Figure 24: Query answering speed on various data for $\Delta = 0$.

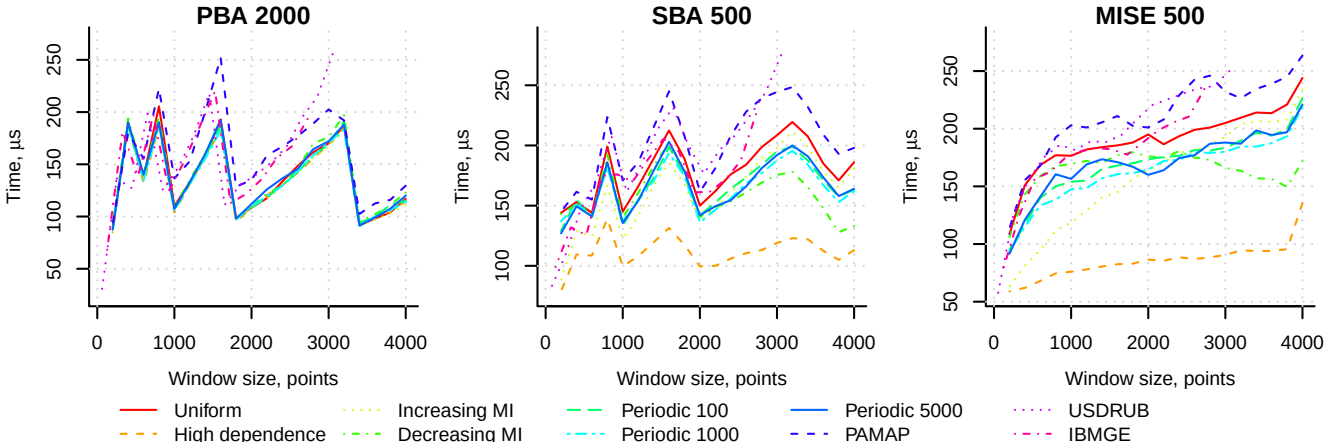

Figure 25: Query answering speed on various data for $\Delta = 4$.