On Complexity and Efficiency of Mutual Information Estimation on Static and Dynamic Data

Michael Vollmer
Karlsruhe Institute of Technology
Karlsruhe, Germany
michael.vollmer@kit.edu

Ignaz Rutter*
Eindhoven University of Technology
Eindhoven, The Netherlands
i.rutter@tue.nl

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

ABSTRACT

Mutual Information (MI) is an established measure for the dependence of two variables and is often used as a generalization of correlation measures. Existing methods to estimate MI focus on static data. However, dynamic data is ubiquitous as well, and MI estimates on it are useful for stream mining and advanced monitoring tasks. In dynamic data, small changes (e.g., insertion or deletion of a value) may often invalidate the previous estimate. In this article, we study how to efficiently adjust an existing MI estimate when such a change occurs. As a first step, we focus on the well-known nearest-neighbor based estimators for static data and derive a tight lower bound for their computational complexity, which is unknown so far. We then propose two dynamic data structures that can update existing estimates asymptotically faster than any approach that computes the estimates independently, i.e., from scratch. Next, we infer a lower bound for the computational complexity of such updates, irrespective of the data structure and the algorithm, and present an algorithm that is only a logarithmic factor slower than this bound. In absolute numbers, these solutions offer fast and accurate estimates of MI on dynamic data as well.

1 INTRODUCTION

Motivation. Finding and quantifying dependencies between variables is an essential task in data analysis. Conventional methods to detect (in)dependent attributes, like correlation coefficients and covariance matrices, are limited in the types of dependencies they detect. *Mutual Information* (MI) in turn is a notion from Information Theory that captures both linear and arbitrary non-linear dependencies. However, MI is defined on the probability density of the data. This makes exact computation impossible on samples. Nevertheless, existing MI estimators yield good results even for small samples [13]. In consequence, a wide range of applications, such as Feature Selection [22], Text Analysis [7] and Computer Vision [23], uses MI.

A popular choice are estimators based on nearest-neighbor distances [9, 16, 17]. This is because such estimators essentially are non-parametric and yield very good results [13, 14, 21, 29]. Nearest-neighbor based estimation of MI is often perceived as equivalent to the concrete estimation formula by Kraskov et al. (KSG)[17]. However, the KSG is just one estimation formula for MI using the nearest-neighbor entropy estimator by Kozachenko and Leonenko [16]. There exists at least one other MI estimator using a different formula, while relying on the same entropy estimator (3KL)[9]. In the following, the term *estimator* names concrete formulas that estimate the MI value (e.g., KSG, 3KL), and *nearest-neighbor based estimation* is the group of these estimators.

So far, algorithms to compute nearest-neighbor based MI estimates and thus their practical applications focus on static data. However, data streams are ubiquitous as well and also require suitable analysis methods. The problem studied in this article is nearest-neighbor based estimation of MI on dynamic data. In this setting, an elementary task called *update* is the incremental maintenance of an estimate when adding or deleting a point.

With dynamic data, scalability with the number of data points is crucial. A good, data-independent measure is the computational complexity of the respective algorithms. In order to evaluate the efficiency of a new solution, it also is important to know the complexity of the problem. That is, a lower bound for any algorithm that computes such estimates, independent of the concrete approach. So far, no lower bound for nearest-neighbor based MI estimation is known, be it for updating an existing estimate, be it for computing the estimate on static data.

Challenges. Designing the estimators envisioned with controlled complexity is challenging. Two reasons for this are as follows: First, while nearest-neighbor based estimation of entropy depends on distances to the nearest neighbor, this does not imply that it has the same computational complexity as nearest-neighbor search. Put differently, it may be possible to obtain the same results using different methodologies. Consequently, one must prove the complexity based only on the result and not hinge on the complexity of certain tasks that seem mandatory in the context at first sight, such as nearest-neighbor search.

Second, to design a dynamic data structure that answers certain queries faster than any static algorithm, it is necessary to identify expensive computations whose results are relatively easy to maintain in the presence of updates. This means that the time required to incrementally maintain the results after a change must be limited in all cases. But this is not obvious here. At the same time, availability of these results must significantly speed up the query.

Our Contributions. Our work focuses on the time required to maintain an estimate of MI on dynamic data. We concentrate on the computational efficiency of nearest-neighbor based estimators on static data and the implications for dynamic estimation. We present solutions for dynamic data that maintain an estimate with the same estimation quality as static estimators, but with less time required. Specifically, our contributions are as follows:

Computational complexity of nearest-neighbor based estimators. In Section 4 we provide a complexity analysis of nearest-neighbor based MI estimators. Using a proof by reduction, we establish a lower bound in the algebraic computation tree model for any algorithm estimating MI using 3KL or KSG. To our knowledge, we are the first to prove any lower bound for the time complexity of such estimators. The lower bounds we prove are tight. This means that there already exist algorithms that have this asymptotically

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

optimal running time. Additionally, we use this result to infer lower bounds for the maintenance of KSG and 3KL estimates on dynamic data.

Dynamic data structures. In this article, we present two dynamic data structures. The first one is DEMI, which estimates nearest-neighbor based MI on a dynamic data set, see Section 5. This data structure holds a set of data points and some intermediate computation results we use for the estimation. The data structure allows insertion and deletion of data points and querying the estimate using all data points stored. Both the 3KL and the KSG estimator can use this data structure. We prove that updating an estimate using DEMI is asymptotically faster than the lower bound for static estimates, i.e., computing the estimate from scratch. To our knowledge, we are first to present a way of maintaining a KSG estimate on dynamic data that requires asymptotically less time than static estimation and preserves the estimate without any approximation.

Near-optimal computation time. The second data structure we present, ADEMI, integrates existing state-of-the-art data structures and algorithms into DEMI to reduce computation time, see Section 6. While the structure does not offer a speedup when maintaining the KSG estimate, we can maintain the 3KL estimate in polylogarithmic time. In particular, we are only a logarithmic factor slower than the lower bound shown in Section 4.

Systematic experimental evaluation. Finally, we evaluate our approaches experimentally, using a broad variety of dependency types and noise levels, in Section 7. We show that both KSG and 3KL converge to the true MI values with a good rate of convergence. This is a stark contrast to another recently published estimator for MI on sliding windows [3]. Additionally, we show that our data structures perform very well when maintaining MI estimates on large samples.

2 FUNDAMENTALS

We begin by revisiting the foundations of MI and its estimation.

Mutual Information. Introduced by Shannon [26], the notion of *entropy* is a measure for the expected information from observing the value of a random variable X, noted as $H(X)$. The expected information for observed values of two random variables X and Y is the natural extension *joint entropy* $H(X, Y)$. This gives way to the notion of *Mutual Information*

$$I(X;Y) = H(X) + H(Y) - H(X,Y), \tag{1}$$

which describes the information shared between both variables. Using the definition of entropy for continuous random variables in Equation 1 yields the differential definition of MI

$$I(X;Y) = \int_Y \int_X p(x,y)\, \log\left(\frac{p(x,y)}{p(x)p(y)}\right) dx\, dy \tag{2}$$

where $p(x)$, $p(y)$ and $p(x, y)$ are the marginal and joint probability density functions of X and Y, respectively [8]. Using the natural logarithm, MI is then measured in the *natural unit of information* (nat).

Nearest-Neighbor based Estimation. Kozachenko and Leonenko [16] presented a nearest-neighbor based estimator for (joint) entropies for a given sample. They used the distance to the k-th nearest neighbor as a means to approximate the density of the distribution for that region. They also have proven that this method yields a consistent estimator for entropy independent of the choice of k. 'Consistent' means that, with increasing sample size, the estimate converges towards the true entropy value. Their method is as follows:

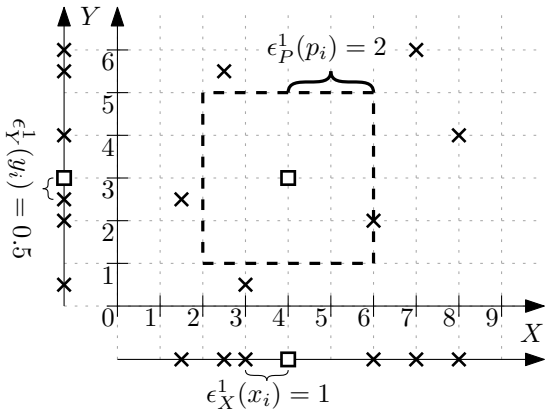

Figure 1: Illustration of the notation used for the 3KL.

Let $Q = \{q_1, \ldots, q_n\} \subseteq \mathbb{R}^d$ be a set of points in a d-dimensional euclidean space, and let $\epsilon_Q^k(q_i)$ be the distance between q_i and its k-th nearest neighbor in Q using the L_∞-norm, also known as maximum distance. Using the notation by Kraskov et al. [17], the entropy estimator by Kozachenko and Leonenko [16] is

$$\hat{H}(Q) = \psi(n) - \psi(k) + \log(2^d) + \frac{d}{n}\sum_{i=1}^{n} \log(\epsilon_Q^k(q_i)), \tag{3}$$

where ψ is the digamma function. That is, $\psi(x) = \sum_{m=1}^{x-1}(\frac{1}{m}) - C$, for $x \geq 1$ with $C \approx 0.577$ being the Euler-Mascheroni-Constant.

We now say how this entropy estimator is used to estimate MI. Let $P = \{p_1 = (x_1, y_1), \ldots, p_n = (x_n, y_n)\} \subseteq \mathbb{R}^2$ be a sample of a random variable with two attributes. Note that we use P for the sample whose MI value we are interested in. This may be the original, full data set as well as the set of the most recent points of a data stream or any other subsample. Additionally, let $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_n\}$ be the sets of all values of the respective attribute in the sample. We use $\epsilon_P^k(p_i)$, $\epsilon_X^k(x_i)$ and $\epsilon_Y^k(y_i)$ to refer to the distance of p_i, x_i and y_i to its k-th nearest neighbor in P, X and Y, respectively. Figure 1 illustrates an exemplary set P, with $k = 1$ and p_i, x_i and y_i marked as squares. Inserting Equation 3 into Equation 1 yields the MI estimator

$$\widehat{I_{3KL}}(P) = \psi(n) - \psi(k) + \frac{1}{n}\sum_{i=1}^{n} \log\left(\frac{\epsilon_X^k(x_i) \cdot \epsilon_Y^k(y_i)}{(\epsilon_P^k(p_i))^2}\right). \tag{4}$$

Because this estimator estimates each of the three entropies in Equation 1 separately with the estimator by Kozachenko and Leonenko, we call the estimator 3KL. This estimator has also been used by Evans [9] for MI estimation on static data. Additionally, because each term in Equation 1 is estimated using a consistent estimator, the 3KL also is a consistent estimator.

Varying numbers of nearest neighbors. A different approach to use Equation 3 for MI estimation was proposed by Kraskov et al. [17]. While the 3KL uses the same k when estimating $\hat{H}(X)$, $\hat{H}(Y)$ and $\hat{H}(P)$ to obtain a compact formula, Kraskov et al. adjust k for every point such that the logarithmic term is 0. The idea is to make the distances $\epsilon_P^k(p_i), \epsilon_X^k(x_i)$ and $\epsilon_Y^k(y_i)$ of a point to its nearest neighbors in X, Y and P identical. To achieve this, the parameter k for $\epsilon_X^k(x_i)$ and $\epsilon_Y^k(y_i)$ has to be set accordingly and may be different for each point. Specifically, each nearest neighbor p_j of p_i in P should result in x_j being a nearest neighbor

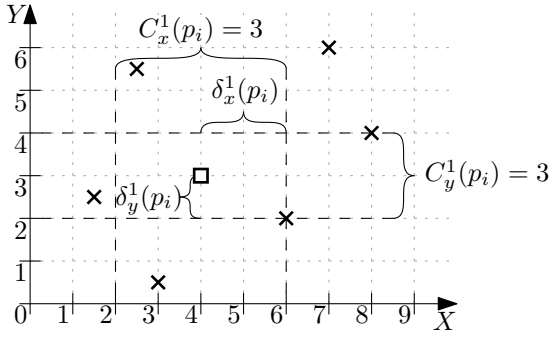

Figure 2: Illustration of the notation used for the KSG.

of x_i in X and y_j being a nearest neighbor of y_i in Y. To this end, the k-th nearest neighbor distance $\epsilon_P^k(p_i)$ is determined and afterwards k_x and k_y for $\epsilon_X^{k_x}(x_i)$ and $\epsilon_Y^{k_y}(y_i)$ is set accordingly. Figure 2 features an illustration of the notation that follows, using the same exemplary set P and k as Figure 1. As before, the set *kNN* of k nearest neighbors of a point $p_i \in P$ using the L_∞-norm is determined first. Let $\delta_x^k(p_i) = \max_{p_j \in kNN} |x_i - x_j|$ be the greatest distance between x_i and any other x-value among its k nearest neighbors. Then the *marginal count* $C_x^k(p_i)$ is the number of elements in X as close to x_i as this distance, i.e.,

$$C_x^k(p_i) = |\{x \in X \setminus \{x_i\} : |x_i - x| \leq \delta_x^k(p_i)\}|. \quad (5)$$

Another marginal count $C_y^k(p_i)$ is defined analogously using y_i and $\delta_y^k(p_i)$. When these marginal counts are used as k per point in the estimator $\hat{H}(X)$ and $\hat{H}(Y)$, the distances $\epsilon_X(x_i), \epsilon_Y(y_i)$ and $\epsilon_P(p_i)$ in Equation 4 (mostly) cancel out. However, one has to adjust the formula for using a different k, i.e., the marginal counts, for each point. The resulting estimator, called KSG due to its inventors Kraskov, Stögbauer and Grassberger, is

$$\widehat{I_{KSG}}(P) = \psi(n) + \psi(k) - \frac{1}{k} - \frac{1}{n}\sum_{i=1}^{n} \psi(C_x^k(p_i)) + \psi(C_y^k(p_i)). \quad (6)$$

While there exist many other approaches to estimate MI, we focus on nearest-neighbor based estimation due to its performance in comparative studies [13, 14, 21, 29]. Because these studies consider only the KSG, we use the same distributions in Section 7 to assess the estimation quality of the 3KL.

Another point is that it is generally recommended [13, 14, 17] to use a small k, that is $k < 10$. The choice of k has only been studied extensively for the KSG but not for the 3KL. However, because Equation 3 is consistent for any k [16], the 3KL is the sum of three consistent estimators and thus consistent for any k as well. Consequently, we assume $k < 10$ in this work, i.e., k is a constant for asymptotic considerations.

3 RELATED WORK

Data Streams. Data streams, a constantly growing form of dynamic data, are ubiquitous. Because data streams grow over time, and memory and storage is limited, it is impossible to store all data points. This means that information is lost over time.

Nevertheless, there exist space-efficient estimators for entropy of discrete distributions on streams. With Equation 1, entropy estimators can also be used to estimate MI, but with accumulating error. The estimator of Chakrabarti et al. [5] provides multiplicative approximations of entropy on insert-only streams. In contrast, the estimator by Harvey et al. [10] offers multiplicative and additive approximations of entropy on streams with insertions and deletions, but requires knowledge about the maximum length of the stream. However, both estimators are restricted to discrete distributions. Estimating MI on discrete distributions is easier, because their relative count of points is a good estimator for the probability. Estimating the density of continuous distributions in turn is not trivial.

The estimation of MI of continuous distributions on streams has received less attention. The MISE framework [12] offers estimates of MI between continuous variables for any time interval on data streams. While both MISE and our approach offer nearest-neighbor based MI estimation, the difference is as follows: Results with MISE are approximations of the KSG estimate for consecutive subsets of the data. We in turn provide exactly the estimates of KSG and 3KL on a dynamic data set. Maintaining an accurate KSG estimate for a dynamic set of data points, e.g., the last 1000 data points, would incur prohibitively high (and growing) resource consumption with MISE. This is because it cannot explicitly delete points. In consequence, the target application and the optimizations are too different to allow for a fair comparison.

Sliding Windows. A common approach to process data streams are sliding windows. Maintaining only a fixed number of points ensures a fixed problem size that allows for bounded resource consumption. By construction, this technique rules out the usage of any information outside the window, but allows for accurate computations on data within it. There already are very good general approaches for sliding-window aggregation [27]. However, no competitive MI estimator is known so far that can be aggregated and thus used with this framework. Most MI estimators have stronger relations to concrete items than to collective values, e.g., distances to the nearest neighbor instead of distances to the mean. In consequence, previous analytics tasks that use MI estimates over a sliding window [15, 24] had to recompute the estimate from scratch for each window.

There is little work regarding algorithmic optimization of the computation time for such tasks. A very recent work by Boidol and Hapfelmeier [3] has introduced an estimator that approximates the 3KL inside a sliding window. In contrast, our approach allows for arbitrary insertions and deletions, and we provide the exact results of the 3KL and KSG. To show the difference between their approximation and accurate 3KL estimates, we include their method in our experiments in Section 7.

Computational Complexity. There has been little research regarding the computational complexity of the KSG and 3KL. Several proposals to compute the KSG appear in the original KSG article [17] with the claimed time complexity $O(n)$ for their fastest, so-called "box-assisted" algorithm on smooth distributions. Vejmelka et al. [28] compare their own approach with the box-assisted algorithm and cite [25] for different conditions for a linear runtime of the box-assisted algorithm. In the end, the best universal time complexity of their presented algorithms is $\Theta(n \log n)$. The same complexity is given for the algorithm computing the 3KL by Evans [9]. In the following section we prove that this limit is not a coincidence, i.e., we prove that no algorithm computing these estimators can have a time complexity lower than $O(n \log n)$.

4 LOWER BOUNDS

In this section we present our first contribution, the lower bounds for computing and maintaining estimates using the KSG and 3KL.

All existing approaches to compute the 3KL and KSG follow the original description in the sense that they first compute the nearest neighbors of all points. In the case of the KSG, the marginal counts C_x^k and C_y^k are computed afterwards. However, it is not known if this is the only approach to compute $\widehat{I_{3KL}}(P)$ and $\widehat{I_{KSG}}(P)$, or if it is computationally optimal. For instance, there could be a different formula for either of these estimators that does not require explicit computation of the nearest neighbors. Consequently, the complexity of computing the 3KL and KSG can only be based on the result and not on intermediate steps such as determining the nearest neighbors. The problems whose complexities we want to study in general, i.e., without confinement to specific algorithms, are the following ones.

Problem 1 (3KL-Estimation)**.** *For a set $P \subseteq \mathbb{R}^2$ of points, determine $\widehat{I_{3KL}}(P)$.*

Problem 2 (KSG-Estimation)**.** *For a set $P \subseteq \mathbb{R}^2$ of points, determine $\widehat{I_{KSG}}(P)$.*

In the following, we show the complexity of Problem 1. By reducing a problem with known complexity to 3KL-Estimation, we prove that it has a lower bound of $\Omega(n \log n)$ in the algebraic computation tree model [1]. For brevity, all formal proofs in this article are available in Appendix A. We use the algebraic computation tree model because it allows us to prove bounds without assuming any statistical properties of the data. This is important because we want general-purpose estimation of MI. If knowledge regarding the data or its distribution was known, it could be used to model the density function in Equation 2.

Theorem 4.1. *The problem 3KL-Estimation has time complexity $\Omega(n \log n)$.*

Proof. The formal proof is available in Appendix A.1. □

This lower bound matches the running time of the algorithm presented by Evans [9] to solve 3KL-Estimation. Consequently, this algorithms is already asymptotically optimal, and the lower bound is tight.

Corollary 4.2. *The computational complexity of 3KL-Estimation is $\Theta(n \log n)$.*

We use the same approach to prove a lower bound for KSG-Estimation. With the algorithms presented by Vejmelka et al. [28] this lower bound is tight as well.

Theorem 4.3. *The problem KSG-Estimation has a time complexity in $\Omega(n \log n)$.*

Proof. The formal proof is available in Appendix A.2. □

Corollary 4.4. *The computational complexity of KSG-Estimation is $\Theta(n \log n)$.*

As a next step, we consider dynamic data. The distinctive feature of dynamic data is that the data changes over time. For a set P of points, all changes can be modeled using insertion of new points and deletion of existing points. For instance, moving a point from (x, y) to (x', y') can be modeled with one deletion of (x, y) and one insertion of (x', y'). To maintain an estimate of MI with the 3KL or KSG, we need to adjust the estimate according to such insertions or deletions. We see this as a problem for a dynamic data structure and thus allow storage of some auxiliary information about P, noted as *state S_P* of a dynamic data structure. The formal problem is then:

Problem 3 (3KL-Update)**.** *Let $P \subseteq \mathbb{R}^2$ be a set of points, S_P the state for P and $p \in \mathbb{R}^2$ a point. Determine $\widehat{I_{3KL}}(P \cup \{p\})$ and $S_{P \cup \{p\}}$ if p is inserted and $\widehat{I_{3KL}}(P \setminus \{p\})$ and $S_{P \setminus \{p\}}$ if p is deleted using only S_P and p.*

Problem 4 (KSG-Update)**.** *Let $P \subseteq \mathbb{R}^2$ be a set of points, S_P the state for P and $p \in \mathbb{R}^2$ a point. Determine $\widehat{I_{KSG}}(P \cup \{p\})$ and $S_{P \cup \{p\}}$ if p is inserted and $\widehat{I_{KSG}}(P \setminus \{p\})$ and $S_{P \setminus \{p\}}$ if p is deleted using only S_P and p.*

Because these problems can be used to solve 3KL-Estimation and KSG-Estimation, respectively, we can use the previous results to infer lower bounds for their time complexities. If we start with an empty set P and incrementally insert n points, the total time required cannot generally be asymptotically faster than $\Omega(n \log n)$ by Theorem 4.1 and Theorem 4.3. Because this includes n insertions, the time complexity of individual insertions is in $\Omega(\log n)$.

Corollary 4.5. *The problem 3KL-Update has a time complexity in $\Omega(\log n)$.*

Corollary 4.6. *The problem KSG-Update has a time complexity in $\Omega(\log n)$.*

In this section we have established formal problem descriptions for the tasks of estimating and maintaining MI estimates using the 3KL and KSG. Furthermore, we have proven lower bounds for the time required to solve these problems. These bounds are tight for computing estimates on static data. This means that no asymptotic speed-up is achievable. In contrast, we are not aware of any data structures or algorithms that solve the problems of maintaining 3KL or KSG estimates when points are inserted or deleted. In the following sections, we present two data structures for these tasks, evaluate their time complexity and compare them to the lower bounds presented in this section.

5 ESTIMATING MUTUAL INFORMATION ON DYNAMIC DATA

Naturally, the simplest solution to KSG-Update and 3KL-Update is storing exactly P in S_P and computing $\widehat{I_{3KL}}(\cdot)$ and $\widehat{I_{KSG}}(\cdot)$, respectively, with every change. The result from the previous section is that any such approach would require $\Omega(n \log n)$ time for 3KL-Update and KSG-Update. In the following we show that this is not optimal and present a more efficient solution.

We propose the data structure DEMI (Dynamic Estimation of Mutual Information) that focuses on updating an estimate of the 3KL or KSG for a single insertion or deletion. First, we present how this data structure works with 3KL estimates. In Section 5.2 we describe the differences when maintaining a KSG estimate. In more detail, we describe the changes to the 3KL estimate that can occur by inserting or deleting a point. Then we describe which information our data structure stores and how it determines the changes in the 3KL estimate efficiently. Lastly, we evaluate the space complexity of our data structure as well as the time complexity of adding or deleting a point.

5.1 Updating 3KL Estimates

Let $P = \{p_1 = (x_1, y_1), \ldots, p_n = (x_n, y_n)\} \subseteq \mathbb{R}^2$ be the set of points in our sample and let $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_n\}$ be the set of values per attribute. When we insert a point $p_{n+1} = (x_{n+1}, y_{n+1}) \in \mathbb{R}^2$, let $P' = P \cup \{p_{n+1}\}$, $X' = X \cup \{x_{n+1}\}$ and $Y' = Y \cup \{y_{n+1}\}$ be the sets including

Data Structure 1: DEMI

struct {
  real x, y
  real $\epsilon_P^k, \epsilon_X^k, \epsilon_Y^k$
} *DemiPoint*;

struct {
  DemiPoint[ ] P_D
  BST<DemiPoint*> T_x, T_y
  real *base*, *sum*
} *state*;

p_{n+1}, x_{n+1} and y_{n+1}, respectively. Considering Equation 4, the change from $\widehat{I_{3KL}}(P)$ to $\widehat{I_{3KL}}(P')$ consists of three partial changes:

(1) $\psi(n)$ increases to $\psi(n+1) = \psi(n) + \frac{1}{n}$,
(2) the arithmetic mean includes $n+1$ logarithms instead of n,
(3) and the nearest-neighbor distances $\epsilon_P^k(p_i)$, $\epsilon_X^k(x_i)$ and $\epsilon_Y^k(y_i)$ may change for any $i \in \{1, \dots, n\}$.

While Change (1) is trivial, Change (2) requires the computation of $\epsilon_{P'}^k(p_{n+1})$, $\epsilon_{X'}^k(x_{n+1})$ and $\epsilon_{Y'}^k(y_{n+1})$. However, Change (3) could require the re-evaluation of all nearest-neighbor distances. Clearly, these changes apply analogously if p_1 is removed from P instead of inserting p_{n+1}. Following these observations, we propose a dynamic data structure that determines these changes efficiently and evaluate its computation complexity.

Overview. Our data structure, DEMI, is given in Data Structure 1. For each point $p_i \in P$ of our sample, we store its attributes x_i, y_i and k-th nearest-neighbor distances $\epsilon_P^k(p_i)$, $\epsilon_X^k(x_i)$ and $\epsilon_Y^k(y_i)$ as a *DemiPoint*. In addition, we store references to all DemiPoints, ordered by the x-component and y-component of the point, in binary search trees (BST) T_x and T_y, respectively. Using self-balancing BST like red-black-trees, we can insert, delete and search items in logarithmic time. Additionally, we also maintain the values $base = \psi(|P|) - \psi(k)$ and $sum = \sum_{i=1}^{n} \log\left(\frac{\epsilon_X^k(x_i) \cdot \epsilon_Y^k(y_i)}{(\epsilon_P^k(p_i))^2}\right)$. The collection of all stored data is the state S_P of our data structure for the sample P. Because we store a constant amount of information per point, the space complexity of DEMI is $\Theta(n)$. Given State S_P, one can query the 3KL estimate on the set P in constant time as $\widehat{I_{3KL}} = base + \frac{sum}{|P|}$. However, this data structure requires adjustment of S_P after every change of P.

Insertion Algorithm. To insert a point p_{n+1} into a state S_P, illustrated in Algorithm 2, we distinguish two phases of the update. First (Lines 1–6), we add p_{n+1} as a DemiPoint to P_D and update *base* and *sum* accordingly. Second (Lines 7–18), we determine which nearest-neighbor distances change and adjust *sum* according to the changes. We now describe these steps in more detail, together with the computational complexity of elementary operations, to allow for an easier evaluation. We discuss possible improvements in Section 6.

To add p_{n+1} to S_P, we first compute its k-th nearest neighbor in P' by linear search and derive the k-th nearest-neighbor distance $\epsilon_{P'}^k(p_{n+1})$ ($O(n)$, Line 1). To determine the k-th nearest neighbor distances $\epsilon_{X'}^k(x_{n+1})$ and $\epsilon_{Y'}^k(y_{n+1})$ we can use the binary search tree and evaluate the distance to the next k and preceding k elements ($O(k \cdot \log n)$, Line 2). With this information we construct the DemiPoint for p_{n+1} and insert it into P_D ($O(1)$,

Algorithm 2: INSERT(S_P, p_{n+1})

1	Compute $\epsilon_{P'}^k(p_{n+1})$	$O(n)$
2	Compute $\epsilon_{X'}^k(x_{n+1})$ and $\epsilon_{Y'}^k(y_{n+1})$	$O(k \cdot \log n)$
3	Insert p_{n+1} into P_D	$O(1)$
4	Reference p_{n+1} in T_x, T_y	$O(\log n)$
5	$base \leftarrow base + \frac{1}{n}$	$O(1)$
6	$sum \leftarrow sum + \log\left(\frac{\epsilon_{X'}^k(x_{n+1}) \cdot \epsilon_{Y'}^k(y_{n+1})}{(\epsilon_{P'}^k(p_{n+1}))^2}\right)$	$O(1)$
7	$A \leftarrow \{p_i \in P\colon \max(\lvert x_i - x_{n+1}\rvert, \lvert y_i - y_{n+1}\rvert) < \epsilon_P^k(p_i)\}$	$O(n)$
8	$B \leftarrow \{p_i \in P\colon \lvert x_i - x_{n+1}\rvert < \epsilon_X^k(x_i)\}$	$O(n)$
9	$C \leftarrow \{p_i \in P\colon \lvert y_i - y_{n+1}\rvert < \epsilon_Y^k(y_i)\}$	$O(n)$
10	**forall** $p_i \in A$ **do**	
11	Compute $\epsilon_{P'}^k(p_i)$	$O(\lvert A\rvert \cdot n)$
12	$sum \leftarrow sum + \log((\epsilon_P^k(p_i))^2) - \log((\epsilon_{P'}^k(p_i))^2)$	$O(\lvert A\rvert)$
13	**forall** $p_i \in B$ **do**	
14	Compute $\epsilon_{X'}^k(x_i)$	$O(\lvert B\rvert \cdot k \cdot \log n)$
15	$sum \leftarrow sum - \log(\epsilon_X^k(x_i)) + \log(\epsilon_{X'}^k(x_i))$	$O(\lvert B\rvert)$
16	**forall** $p_i \in C$ **do**	
17	Compute $\epsilon_{Y'}^k(y_i)$	$O(\lvert C\rvert \cdot k \cdot \log n)$
18	$sum \leftarrow sum - \log(\epsilon_Y^k(y_i)) + \log(\epsilon_{Y'}^k(y_i))$	$O(\lvert C\rvert)$

Line 3). References to this point are then inserted into T_x and T_y ($O(\log n)$, Line 4). Then, we add the appropriate terms to *base* and *sum* ($O(1)$, Lines 6 and 7), respectively.

Next, we find all previous nearest-neighbor distances that changed, by linear search. For each $i \in \{1, \dots, n\}$ we test whether p_{n+1}, x_{n+1} and y_{n+1} is closer than $\epsilon_P^k(p_i)$, $\epsilon_X^k(x_i)$ and $\epsilon_Y^k(y_i)$, respectively. This takes time in $O(n)$ and yields the sets A, B and C (Lines 7–9), respectively. For each point $p_i \in A$ we compute $\epsilon_{P'}^k(p_i)$ analogously to $\epsilon_{P'}^k(p_{n+1})$, which takes $O(n)$ each. Then we adjust *sum* accordingly ($O(1)$, Line 12). The sets A and B are handled in an analogous way, using $\epsilon_{X'}^k(x_i)$ and $\epsilon_{Y'}^k(y_i)$, respectively, instead (Lines 13–18). Note that these distances can be computed in time $O(k \cdot \log n)$ each, instead of $O(n)$, analogous to $\epsilon_{X'}^k(x_{n+1})$ and $\epsilon_{Y'}^k(y_{n+1})$.

Computational Complexity. The total runtime for inserting a point into our structure therefore is in $O(k \cdot n + |A| \cdot n + (|B| + |C|) \cdot k \cdot \log n)$. In the following theorem we show that $|A|$,$|B|$ and $|C|$ are in $O(k)$, because there are at most $8 \cdot k$ points for which p_{n+1} is one of the k nearest neighbors. Consequently, our insertion time is in $O(k \cdot n + k^2 \cdot \log n)$. Since k is suggested to be a small constant, e.g. less than 10, in the literature, we can assume k to be constant. This means that an insertion is in $O(n)$. This results in the total time complexity of $O(n)$. Because deleting a point changes the estimate analogously, we can use an analogous algorithm with the same complexity, i.e., $O(n)$.

THEOREM 5.1. *Let $P \subseteq \mathbb{R}^2$ be a set of points. For any point $p \in P$ there exist at most $8k$ points $q \in P$ such that p is one of the k nearest neighbors of q using the L_∞-norm.*

PROOF. The formal proof is available in Appendix A.3. □

As context for the update time of $O(n)$, Theorem 4.1 proves that any algorithm requires time in $\Omega(n \log n)$ to compute the 3KL from scratch. As a result, updating an estimate using DEMI is asymptotically faster than recomputing it, independently of the method used. In Section 6 we show how the time for updates

Data Structure 3: DEMI-KSG

```
struct {
    real x, y
    real ε_P^k, δ_x^k, δ_y^k
    int C_x^k, C_y^k
} DemiPointKSG;

struct {
    DemiPointKSG[ ] P_D
    BST<DemiPointKSG*> T_x, T_y
    real base, sum
} state;
```

on the 3KL can be improved even further. However, we will first discuss how we use the same approach to update KSG estimates.

5.2 Updating KSG Estimates

In this section, we describe how we achieve the same results, that is linear space and linear time for updates, using KSG estimates instead of 3KL estimates. As with the 3KL, we decompose the KSG estimate into $\widehat{I_{KSG}} = base + \frac{sum}{|P|}$. Comparing Equation 4 and Equation 6, it follows that *base* and *sum* need to maintain different values when maintaining 3KL or KSG estimates. The change for *base*, that is $base = \psi(|P|) + \psi(k) - \frac{1}{k}$ instead of $base = \psi(|P|) - \psi(k)$, does not have any influence on the overall procedure. However, the change from $sum = \sum_{i=1}^{n} \log\left(\frac{\epsilon_X^k(x_i)\cdot\epsilon_Y^k(y_i)}{(\epsilon_P^k(p_i))^2}\right)$ to $sum = -\sum_{i=1}^{n} \psi(C_x^k(p_i)) + \psi(C_y^k(p_i))$ has stronger implications. Most notably, we do not require explicit nearest neighbor distances per point but need marginal counts. We need to update a marginal count $C_x^k(p_i)$ if and only if the nearest neighbors of p_i in P changes, or a point (x, y) with $|x - x_i| \leq \delta_x^k(p_i)$ is inserted or deleted, see Figure 2. As a consequence, per point p_i we do not store $\epsilon_X^k(x_i)$ and $\epsilon_Y^k(y_i)$ but the distances to the furthest x- and y-values among the k nearest neighbors in P, i.e., $\delta_x^k(p_i)$ and $\delta_y^k(p_i)$. Additionally we track the marginal counts $C_x^k(p_i)$ and $C_y^k(p_i)$. These slight changes are displayed in Data Structure 3. Furthermore, this means that we still store a constant amount of information per point, and the space complexity of the data structure remains $\Theta(n)$.

Updating the data structure follows the same principles as before, that is, we include the new point into the data structure and evaluate its impact on other marginal counts afterwards. In the following we describe the changes in specific steps between the update algorithm for 3KL estimates and KSG estimates, that is, Algorithm 2 and Algorithm 4.

Tracking marginal counts, instead of nearest-neighbor distances, per attribute allows for faster updates, because the counts only need increments and decrements ($O(1)$ each, Lines 16 and 18), instead of recomputation. However, a change of nearest neighbors does also invalidate the marginal counts and requires computing them and correct adjustment of *sum* (Lines 11–14). Computing marginal counts from scratch can be done with linear search ($O(n)$ each, Lines 2 and 13).

Regarding the time complexity of Algorithm 4, it is important to note that B and C are not sets of points with changed nearest neighbors. As a consequence, only the size of A has an upper bound of $8 \cdot k$ by Theorem 5.1. In the worst case, B and C contain all points, that is, $|B| \leq n$ and $|C| \leq n$. The total time complexity

Algorithm 4: INSERT-KSG(S_P, p_{n+1})

```
1  Compute δ_x^k(p_{n+1}), δ_y^k(p_{n+1}) and ε_{P'}^k(p_{n+1})                O(n)
2  Compute C_x^k(p_{n+1}) and C_y^k(p_{n+1})                                  O(n)
3  Insert p_{n+1} into P_D                                                    O(1)
4  Reference p_{n+1} in T_x, T_y                                              O(log n)
5  base ← base + 1/n                                                          O(1)
6  sum ← sum − psi(C_x^k(p_{n+1})) − psi(C_y^k(p_{n+1}))                      O(1)
7  A ← {p_i ∈ P: max(|x_i − x_{n+1}|, |y_i − y_{n+1}|) < ε_P^k(p_i)}          O(n)
8  B ← {p_i ∈ P: |x_i − x_{n+1}| < δ_x^k(p_i)}                                O(n)
9  C ← {p_i ∈ P: |y_i − y_{n+1}| < δ_y^k(p_i)}                                O(n)
10 forall p_i ∈ A do
11     sum ← sum + psi(C_x^k(p_i)) + psi(C_y^k(p_i))                          O(|A|)
12     Compute δ_x^k(p_i), δ_y^k(p_i) and ε_{P'}^k(p_i)                       O(|A| · n)
13     Compute C_x^k(p_i) and C_y^k(p_i)                                      O(|A| · n)
14     sum ← sum − psi(C_x^k(p_i)) − psi(C_y^k(p_i))                          O(|A|)
15 forall p_i ∈ B do
16     sum ← sum − 1/C_x^k(p_i);  C_x^k(p_i) ← C_x^k(p_i) + 1                 O(|B|)
17 forall p_i ∈ C do
18     sum ← sum − 1/C_y^k(p_i);  C_y^k(p_i) ← C_y^k(p_i) + 1                 O(|C|)
```

therefore is $O(n + |A| \cdot n) = O(k \cdot n)$. As before, k is taken as constant, which yields the time complexity $O(n)$. This is asymptotically faster than recomputing the estimate by Theorem 4.3.

6 POLYLOGARITHMIC UPDATES

Because DEMI relies only on simple algorithms like linear search and binary search trees during insertions and deletions, faster solutions might exist. In this section we determine which parts of our insertion algorithm have a high computational cost and present solutions for these tasks. There are two factors that lead to the linear time complexity of Algorithm 2.

(1) Computing the nearest neighbors, with linear search
(2) Finding the points whose nearest neighbors changed by linear search

6.1 Geometric Structures

Computing the nearest neighbors. Computing the k nearest neighbors of a point is a classic problem of computational geometry, which has received a lot of research. While there exist many solutions, most of them are built for static data and are not compatible with the incremental changes in dynamic data. But there also exist solutions that allow for insertions and deletions. Chan [6] proposed a dynamic data structure that computes nearest neighbors in two-dimensional spaces with sub-linear times for insertion, deletion and queries. However, the computational complexity of deletions is $O(\log^6 n)$, which is quite high. Kapoor and Smid [11] provide an alternative based on dynamic range trees [30]. With dynamic fractional cascading [20] the time complexities for insertions, deletions and querying the nearest neighbor of a point are in $O(\log n \log\log n)$. To query two nearest neighbors, we can query one nearest neighbor, delete this point from the tree, query the new nearest neighbor and insert the deleted point. Querying the k nearest neighbors can thus easily be achieved through a sequence of k queries, $k-1$ deletions, and $k-1$ insertions, with total time in $O(k \cdot \log n \log\log n)$.

Finding the points whose nearest neighbors have changed. Finding all points whose nearest neighbors have changed is also a geometric problem, that is, finding the reverse nearest neighbors of the inserted or deleted point. For each point $p = (x, y)$ with nearest neighbor distance ϵ, all nearest neighbors of p (using the L_∞-norm) are within the square $[x-\epsilon, x+\epsilon] \times [y-\epsilon, y+\epsilon] \subseteq \mathbb{R}^2$. To find all points whose nearest neighbors contain a point p', the task is to determine which squares contain p'. One data structure to solve this problem is the segment tree by Bentley [2]. The technique of dynamic fractional cascading is also applicable for segment trees [20] and yields the time complexities for insertions and deletions in $O(\log n \log\log n)$. Queries require time in $O(\log n \log\log n + m)$, where m is the number of squares returned.

6.2 Improving DEMI

To achieve sublinear time complexity for updates, we integrate a two-dimensional dynamic range tree and a two-dimensional dynamic segment tree into DEMI. We call this the augmented version of DEMI (ADEMI). The insertion algorithm is nearly identical to Algorithm 2, except for changes in time complexities and insertions and deletions to the integrated tree structures. In consequence, we only mention the changes relative to Algorithm 2 in this section. The full data structure and insert algorithm can be found in Appendix B.

Using the dynamic range tree, Line 1 requires only time in $O(k \cdot \log n \log\log n)$, and Line 11 requires time in $O(|A| \cdot k \cdot \log n \log\log n)$. Using the dynamic segment tree, Line 7 can be done in time $O(\log n \log\log n + |A|)$. Additionally, B and C can only contain elements that are at most k positions before and after x_{n+1} and y_{n+1} in T_x and T_y, respectively. Consequently, Lines 8–9 can also be done using the binary search trees in time $O(k \cdot \log n)$.

Additionally, we need to maintain the integrated tree structures. Specifically, we insert p_{n+1} into the dynamic range tree and insert the square of its nearest neighbors, that is,

$$\begin{aligned} \mathit{square}(p_{n+1}, P') = [x_{n+1} - \epsilon^k_{P'}(p_{n+1}), x_{n+1} + \epsilon^k_{P'}(p_{n+1})] \\ \times [y_{n+1} - \epsilon^k_{P'}(p_{n+1}), y_{n+1} + \epsilon^k_{P'}(p_{n+1})], \end{aligned} \quad (7)$$

into the dynamic segment tree. The dashed lines in Figure 1 illustrate this square. Both insertions require time in $O(\log n \log\log n)$. Finally, for each point $p_i \in A$ we delete its old square of nearest neighbors $\mathit{square}(p_i, P)$ from the dynamic segment tree and insert the new square $\mathit{square}(p_i, P')$. This requires time in $O(|A| \cdot \log n \log\log n)$.

For an overview of the new time complexity, the updated insertion algorithm can be found in Appendix B. Because $|A|, |B|, |C| \in O(k)$, the total time complexity of an insertion is $O(k^2 \cdot \log n \cdot \log\log n)$. As before, k can be assumed to be a small constant, which leads to an insertion time of $O(\log n \log\log n)$. Deleting a point is completely analogous to insertions in (A)DEMI, and the used tree structures have the same complexity for insertions and deletions. Consequently, deletions in ADEMI also have a deletion time of $O(\log n \log\log n)$. Since the time complexity of queries is in $O(1)$, ADEMI solves problem 3KL-Update in time $O(\log n \log\log n)$. This means that ADEMI is a nearly optimal, since its time complexity is only a factor $\log\log n$ higher than the lower bound from Corollary 4.6.

The drawback of ADEMI is an increased space complexity. The space complexity of the two-dimensional range tree and segment tree are $O(n \log n)$ and $O(n \log^2 n)$, respectively. Additionally, the improvements to the time complexity cannot be used when maintaining KSG estimates. This is because the number of points whose marginal counts change during an update has no bound lower than n. Additionally, the impact of incrementing or decrementing a marginal count on the overall estimate depends on the current count, which can be any value between k and n. As a consequence, it remains unclear whether any dynamic data structure can solve KSG-Update in sublinear time, or whether there exists a stronger lower bound.

7 EXPERIMENTS

In this section we empirically validate the estimation quality and time efficiency of our approach. To this end, we use data with known MI values and show that the 3KL converges to these values even with small samples. We also do so for the KSG. For brevity we only present the results for $k = 4$, since this value offers good rates of convergence for both the KSG and 3KL and follows the general recommendation of small values for k. Additionally, we compare the runtimes for maintaining 3KL estimates using ADEMI, DEMI and repeated estimation from scratch (REFS). For REFS we compute Equation 4 repeatedly with a state-of-the-art static approach [9], i.e., using sorting and space-partitioning trees for nearest-neighbor searches. While we have already proven a clear hierarchy regarding their asymptotic scalability, the complexity classes neglect constant factors. So it remains interesting how their concrete runtimes compare.

Setup. All approaches are implemented in C++ and compiled using the Gnu Compiler (v. 5.4) with optimization (-O3) enabled. We use the non-commercial ALGLIB[1] implementation of KD-Trees as space-partitioning trees in REFS. We conduct all experiments on Ubuntu 16.04.2 LTS using a single core of an AMD Opteron™ Processor 6212 clocked at 2.6 GHz and 128GB RAM.

7.1 Data

For our evaluation, we use synthetic and real data sets. In particular, we use the dependent distributions with noise used for comparing MI estimators [13]. These distributions have a noise parameter σ_r, which we vary from 0.1 to 1.0. Thus, we use 10 distributions for each of these dependency types. Additionally, we use the uniform distributions used to compare MI with the maximal information coefficient [14] as well as independent uniform and normal distributions. As real data sets, we use sensor data of randomly charged and discharged batteries [4] and time series of household power consumption [18]. Monitoring MI on such data could be useful to monitor the condition of battery cells for maintenance or to infer knowledge about the behavior of the households inhabitants. In the following, we briefly describe the different distributions and data sets.

Linear. To construct the point $p_i \in P$, we draw the value x_i from the normal distribution $N(0, 1)$. Additionally, we draw some noise r_i from the normal distribution $N(0, \sigma_r)$, where σ_r is the noise parameter of the distribution. This yields the point $p_i = (x_i, x_i + r_i)$.

Quadratic. This distribution is generated analogously to the linear distribution, except that the point is $p_i = (x_i, x_i^2 + r_i)$.

Periodic. For each point $p_i \in P$, we draw the value x_i from the uniform distribution $U[-\pi, \pi]$. Additionally, we draw some noise r_i from the normal distribution $N(0, \sigma_r)$, where σ_r is the noise parameter. This yields the point $p_i = (x_i, \sin(x_i) + r_i)$.

[1] ALGLIB (www.alglib.net), Sergey Bochkanov

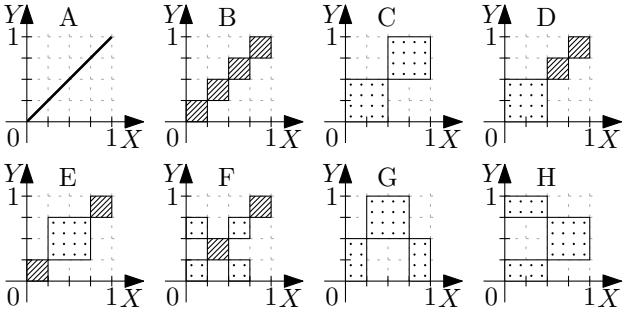

Figure 3: An overview of the uniform distributions used.

Chaotic. This distribution uses the classical Hénon Map, that is,

$$h_{x_{i+1}} = 1 - \alpha \cdot h_{x_i}^2 + h_{y_i}$$
$$h_{y_{i+1}} = \beta \cdot h_{x_i},$$

with $\alpha = 1.4, \beta = 0.3$ and $(h_{x_0}, h_{y_0}) = (0, 0)$. For a point p_i we additionally independently draw noise r_{xi}, r_{yi} from the distribution $N(0, \sigma_r)$, where σ_r is the noise parameter. Each point $p_i \in P$ is then $p_i = (h_{x_i} + r_{xi}, h_{y_i} + r_{yi})$.

Uniform. The uniform distributions A to H we use are illustrated in Figure 3. Note that the striped areas contain twice as many points as the dotted areas. For these distributions, each striped area with size $0.25 \cdot 0.25$ contains 25% of all points, while dotted areas of the same size contain 12.5% of all points. The distribution A simply draws values v_i from U[0,1] and constructs the points $p_i = (v_i, v_i)$.

Independent. Lastly, we use the distributions U_{IND} and N_{IND}, where each point consists of two values drawn independently and identically distributed from $U[0, 1]$ and $N(0, 1)$, respectively.

Battery Data. This data set, available at the NASA Prognostics Center of Excellence [4], monitors voltage, current and temperature of battery cells during random loads. We use the data corresponding to battery cell "RW9" and use each combination of the attributes as bivariate sample.

Power Consumption. This data set, available at the UCI Machine Learning Repository [18], monitors the power consumption of a household in France. We use each combination of *global active power*, *global reactive power* and *voltage* as a bivariate sample.

Data Precision. The nearest-neighbor based entropy estimator, and by consequence the 3KL and KSG, expects samples from continuous distributions and require samples without duplicate values. Because of the limited precision of the battery and the power consumption data, we add noise to the sample. Kraskov et al. also have observed this issue and recommend the addition of low intensity noise, e.g., a normal distribution with variance 10^{-10}, to eliminate duplicate points [17]. However, we think that filling the missing precision with uniform noise is a better compensation for rounded or imprecise data. Figure 4 illustrates both approaches with the number of duplicates per value of an imprecise data set in parentheses. For our experiments we use the second approach.

7.2 Quality of Estimation

To evaluate the quality of estimation, we use all data sets with well-defined MI values. That is, all synthetic data sets except the chaotic distributions, whose probability densities are unknown,

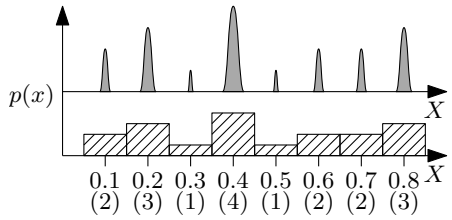

Figure 4: Avoiding duplicates in a sample by adding minimal noise (top) or filling the missing precision uniformly (bottom).

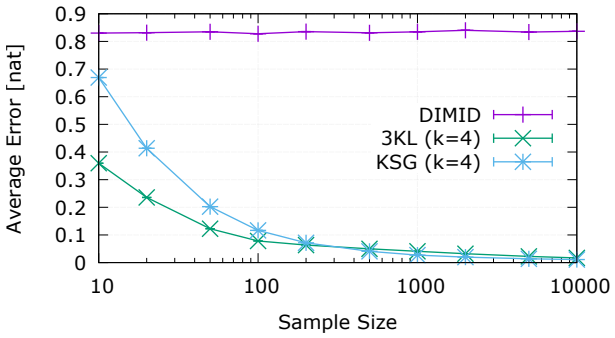

Figure 5: Average difference of estimates to true MI values depending on sample size.

and the uniform distribution A, whose MI is infinite. We use these distributions to evaluate the consistency and the rate of convergence of the KSG, 3KL and the estimator used by DIMID [3]. Specifically, we are interested in the difference between the estimated MI and true value for the distribution as well as the variance of estimates for samples of the same distribution. Since the behavior has turned out to be very homogeneous across the different distributions, we restrict our presentation to selected results.

Development with sample size. For each distribution we created samples with sample sizes between 10 and 10000 and 1000 repeats per size. Figure 5 graphs the average difference between the estimate and the true MI value of the respective distribution. Additionally, Figure 6 shows the standard deviation of estimates of the same distribution and sample size, averaged across all distributions. One can see in these diagrams, that both the 3KL and the KSG converge quickly to the true values and have only small variance. In contrast, the approximate estimator in DIMID has a strong variance and difference. We think the reason is the random projection used by that estimator. It may retain enough information such that estimates are comparable to each other, as shown in their work [3]. However, we think that the projection loses too much information regarding the joint probability density to allow for good MI estimates.

Different dependency types. We also studied whether the quality of estimation changes for different dependency types. As we have seen in the previous paragraph, both the 3KL and KSG are very consistent even with moderate sample sizes. As a result we will use a small sample size, i.e. 100, to highlight differences. Figure 7 shows the average estimation error and standard deviation of estimates using 3KL, KSG and DIMID for each dependency type. While the variance of both KSG and 3KL are comparable

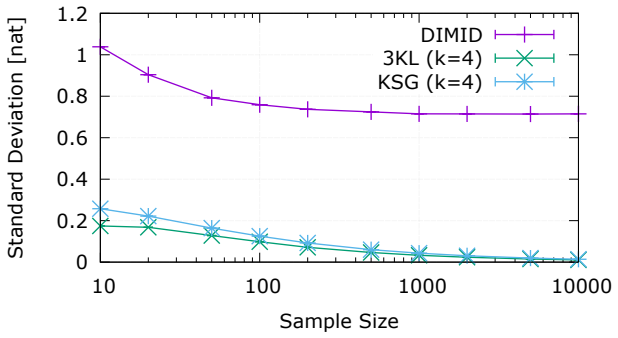

Figure 6: Standard deviation of estimates depending on sample size.

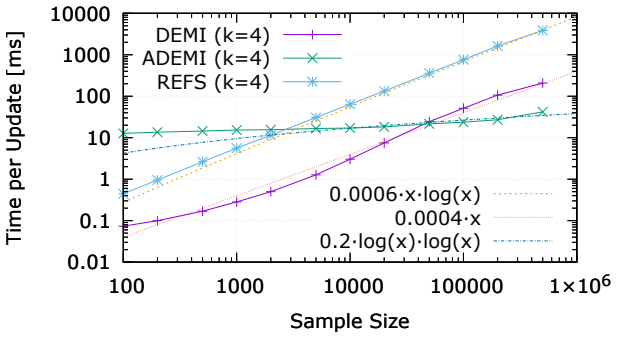

Figure 8: Average time for an update depending on sample size for the synthetic distributions.

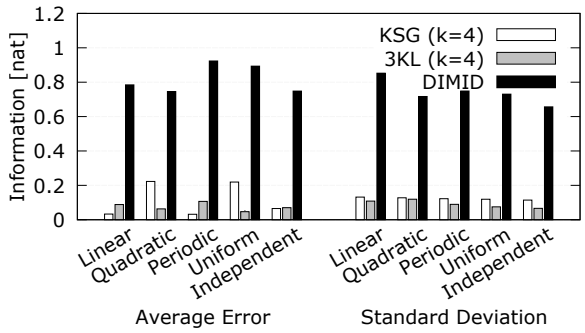

Figure 7: Average difference (left) and standard deviation (right) of estimates to true MI values on distribution type.

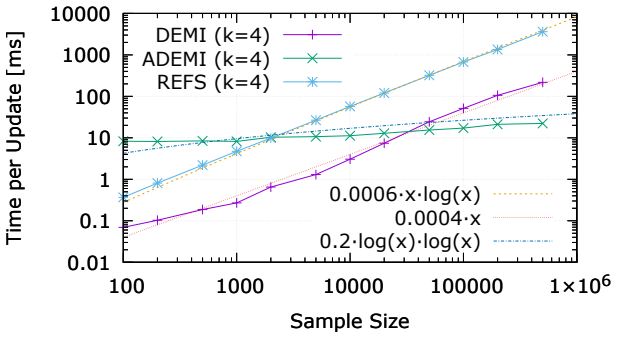

Figure 9: Average time for an update depending on sample size for the used real data set.

for all dependency types, the difference to the true value is imbalanced for the KSG but not the 3KL. Unfortunately, we do not have any explanation for this difference. As before, we notice strong differences between the DIMID approximation and the results of KSG and 3KL.

7.3 Runtime Analysis

We have benchmarked runtimes of our data structures for all data sets. Because we are not aware of any competitor that offers good MI estimates on dynamic data, we compare our performance to naïve recomputation of the estimate when an update occurs. We compare the runtime to maintain 3KL estimates using DEMI and ADEMI as well as repeated recomputation (REFS). We use a slight simplification of the ADEMI trees, compared to the description in Section 6. Specifically, we did not implement dynamic fractional cascading and relied only on the technique of Willard [30] for insertion and deletion of nodes. The reason is that dynamic fractional cascading provides a small asymptotic benefit, i.e. reducing a factor $\log n$ to $\log \log n$, but requires a lot of overhead. As a result, the structure labeled ADEMI in this section has insertion and deletion time in $O(\log^2 n)$ instead of $O(\log n \log \log n)$.

By design, both DEMI and ADEMI require only constant time for querying the current MI value, but require more time to update the data structure during insertions and deletions. The repeated static estimation REFS has inverse properties, i.e., constant time insertions and deletions but expensive queries. To provide a good overview we use the task of monitoring the MI of a changing data set of fixed size. That is, each update consists of deleting one point, inserting a different point and querying the current MI estimate. For these experiments we averaged the time required per update using 1000 updates per distribution and sample size.

Figure 8 shows the average update time required across all synthetic distributions per sample size. The same graph based on the real data sets instead of the synthetic distributions is Figure 9. As expected, the time complexity of each approach translates directly to asymptotic scaling with sample sizes, that is, steepness of the curve in the double log plot. To highlight this, the graphs include different asymptotic functions with dashed and dotted lines. An interesting result is that ADEMI has by far the worst performance for small windows and by far the best performance for large ones. Our explanation is as follows: The maintenance of the range trees and even more so the segment trees is expensive, even if it scales favorably. For instance, when inserting a square into a two-dimensional segment tree, $8 \cdot (1 + \log n)$ nodes are created in the tree. This is a lot even for small n but does not increase significantly for large n.

7.4 Discussion

To summarize this section, we confirmed the estimation quality of 3KL and KSG across all dependency types tested. Additionally, we compared the performance of DEMI, ADEMI and REFS both on synthetic and real data. As expected, DEMI consistently outperforms the SE. The evaluation of ADEMI depends on the context and application. While it is slow for small window sizes, it barely slows down for larger sizes. On the one hand, this means that it is often recommendable to use DEMI if the data size is small. On the other hand, ADEMI can be used for very data-intensive

tasks such as monitoring high-throughput streams. A problem with stream monitoring often is the multiplicative cost of high temporal resolutions: A stream with frequent items permits less time to process a new item, and a window with fixed time length contains more items. This leads to increased time to process a new item. As we have seen, the second factor is nearly negligible when using ADEMI.

8 CONCLUSIONS

In this work we have studied the efficiency of estimating mutual information using nearest-neighbor distances. We have considered the estimator by Kraskov et al. [17](KSG) and the direct application of the entropy estimator [9, 16](3KL). We have investigated the computational complexity of these estimators on static data and have proven a tight lower bound for both in the algebraic computation tree model. Next, we have turned to the maintenance of 3KL and KSG estimates on dynamic data and have examined possible optimizations and limitations. We have inferred a lower bound for the computational complexity of this task. We also have presented two dynamic data structures DEMI and ADEMI that maintain 3KL and KSG estimates. We have proven that both data structures require asymptotically less time to update their estimate than the lower bound to recompute it. Additionally, for maintenance of 3KL estimates, the time complexity of ADEMI is near optimal. Finally, we have validated the performance of our approach empirically. We have shown that the 3KL has a good rate of convergence for various dependencies. We also have benchmarked our data structure using both synthetic and real data and have shown that ADEMI is very fast for large data sets.

Future Work. In this work we have focused on exact computations of nearest-neighbor based MI estimators for dynamic data. It remains open whether our approach offers the best trade-off between estimation quality and computation time.

For one, it would be interesting which results one could achieve by binning the data and using estimators for discrete distributions [5, 10]. However, it is unclear how the bin width should be chosen, given the evolving nature of a stream. If the bin width needs adjustment, this is computationally expensive or reduces the quality of estimation if there is no adjustment.

It would also be interesting to study which provable quality one might achieve with approximations. While there exist approximations of static estimators [3, 12], there are no bounds for additive or multiplicative errors. But these would be very important because there exists a lot of work comparing static estimators. In addition, empirical assessments of new estimators often cover only some of the dependencies that MI quantifies.

A FORMAL PROOFS

A.1 Proof of Theorem 4.1

Theorem 4.1. *The problem 3KL-Estimation has time complexity* $\Omega(n \log n)$.

Proof. The proof is by reduction from the problem IntegerElementDistinctness. Given a multiset $A = \{a_1, \ldots, a_n\}$ of integers, are there two indices $i \neq j$ such that $a_i = a_j$ are duplicates. The problem IntegerElementDistinctness has a known lower bound of $\Omega(n \log n)$ in the algebraic computation tree model [19]. For an instance A of IntegerElementDistinctness, we construct an instance of 3KL-Estimation P as follows. For $a_i \in A$, the set P contains two points $p_i = (i, a_i + \frac{1}{4+i})$ and

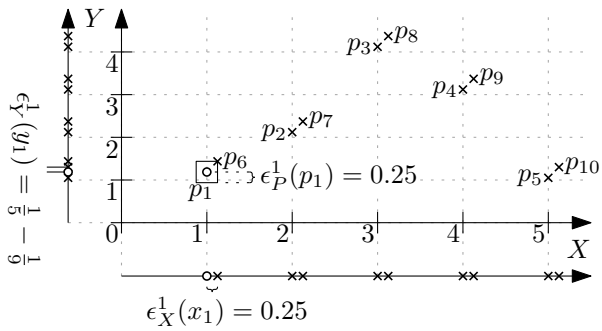

Figure 10: An illustration of the construction of P in Appendix A.1.

$p_{n+i} = p_i + (0.25, 0.25)$. Note that p_i and p_{n+i} are closer than any other pair, because i and a_i are integers. Additionally, we add the offset $\frac{1}{4+i}$ to the y-coordinates, because duplicates in A would otherwise lead to a nearest-neighbor distance of 0 and thus $\log(0)$ in Equation 4. Figure 10 features an example for this construction for the IntegerElementDistinctness instance $A = \{1, 2, 4, 3, 1\}$. The point p_1 is highlighted as circle and its nearest-neighbor distances are highlighted.

Claim 1. *A contains a duplicate if and only if $\widehat{I_{3KL}}(P) \neq \psi(|P|) - \psi(1)$ for $k = 1$.*

Subproof. Let $i, j \in \{1, \ldots, n\}$ be two integers with $i \neq j$. Based on the construction of P, it follows that $|x_i - x_j| = |x_{n+i} - x_{n+j}| \geq 1$. Additionally, it is $|x_i - x_{n+i}| = |y_i - y_{n+i}| = 0.25$. Using the reverse triangle inequality, it is $|x_i - x_{n+j}| \geq |x_i - x_j| - |x_j - x_{n+j}| \geq 0.75$. This holds for any $i \neq j$, which means that p_i is the nearest neighbor of p_{n+i} and vice versa, because we use the L_∞ norm. As a consequence the nearest neighbor distances are $\epsilon_P^1(p) = 0.25$ for all $p \in P$ and $\epsilon_X^1(x) = 0.25$ for all $x \in X$. Note that this means that the nearest neighbor distances in P and X are independent of the existence of duplicates in A.

If A does not contain any duplicates, it follows that $|y_i - y_j| = |y_{n+i} - y_{n+j}| \geq \frac{4}{5}$, since A only contains integers and the difference between $\frac{1}{4+i}$ and $\frac{1}{4+j}$ is less than 0.2. By the same arguments as above it follows that $|y_i - y_{n+j}| \geq 0.55$ and that $\epsilon_Y^1(y) = 0.25$ for all $y \in Y$. We can then use these values in Equation 4, which yields:

$$\widehat{I_{3KL}}(P) = \psi(|P|) - \psi(1) + \frac{1}{|P|} \sum_{m=1}^{|P|} \log\left(\frac{0.25 \cdot 0.25}{(0.25)^2}\right) = \psi(|P|) - \psi(1). \tag{8}$$

Conversely, if A contains the duplicates $a_i = a_j$, it is $|y_i - y_j| = |\frac{1}{4+i} - \frac{1}{4+j}| \leq 0.2$ and $|y_i - y_{n+j}| \geq |0.25 - \frac{1}{4+j}|$ and thus $\epsilon_Y^1(y_i) \leq 0.2$. Additionally, because of $j \neq i$ it also is $\epsilon_Y^1(y_i) > 0$. It follows that

$$\log\left(\frac{\epsilon_Y^1(y_i) \cdot \epsilon_X^1(x_i))}{(\epsilon_P^1(p_i))^2}\right) < 0 \Rightarrow \frac{1}{|P|} \sum_{m=1}^{|P|} \log\left(\frac{\epsilon_X^1(x_m) \cdot \epsilon_Y^1(y_m)}{\epsilon_P^1(p_m))^2}\right) < 0 \tag{9}$$

and analogously to Equation 8 we obtain $\widehat{I_{3KL}}(P) < \psi(|P|) - \psi(1)$. This concludes the subproof. ∎

It is clear that P can be constructed in time $O(|A|)$, which means $|P| \in O(|A|)$. After computing $\widehat{I_{3KL}}(P)$, the result is only compared to a sum over $|P|$ numbers, because $\psi(|P|) - \psi(1) = \sum_{m=1}^{|P|-1} \frac{1}{m}$ by definition of the digamma function. Note that this

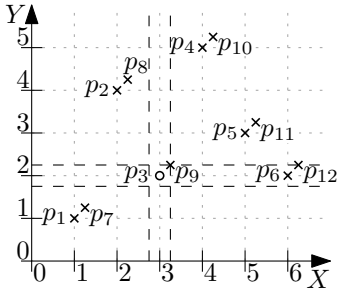

Figure 11: An illustration of the construction of P in Appendix A.2.

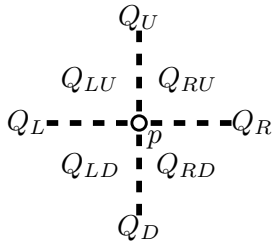

Figure 12: The partioning of Q in Appendix A.3.

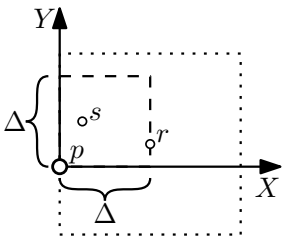

Figure 13: An example for Q_{RU}.

reduction works analogously for any fixed $k > 0$ by placing $k-1$ points evenly spaced on the diagonal between each pair p_i and p_{n+i}. Because k is fixed, the size of P increases only by a constant factor. Therefore, the complexity of the reduction is in $O(n)$. This means that determining $\widehat{I_{3KL}}(P)$ has a lower bound of $\Omega(n \log n)$. □

A.2 Proof of Theorem 4.3

Theorem 4.3. *The problem KSG-Estimation has a time complexity in $\Omega(n \log n)$.*

Proof. Similarly to the Proof of Theorem 4.1, see Appendix A.1, we reduce the problem to IntegerElementDistinctness. For any instance A of IntegerElementDistinctness, we construct an instance of KSG-Estimation P as follows. For $a_i \in A$, the set P contains two points $p_i = (i, a_i)$ and $p_{n+i} = (i + 0.25, a_i + 0.25)$. We use 0.25 because it means that this pair of points is closer than any other pair, because i and a_i are integers. Figure 11 features an example for this construction for the IntegerElementDistinctness instance $A = \{1, 4, 2, 5, 3, 2\}$. The dashed lines in the figure illustrate the areas of the marginal counts $C_x^1(p_3)$ and $C_y^1(p_3)$.

Claim 2. *A contains a duplicate if and only if $\widehat{I_{KSG}}(P) \neq \sum_{m=1}^{|P|-1}(\frac{1}{m}) - 1$ for $k = 1$.*

Subproof. Let $i, j \in \{1, \ldots, n\}$ be two integers with $i \neq j$. Based on the construction of P, it follows that $|x_i - x_j| = |x_{n+i} - x_{n+j}| \geq 1$. Additionally, it is $|x_i - x_{n+i}| = |y_i - y_{n+i}| = 0.25$. Using the reverse triangle inequality, it is $|x_i - x_{n+j}| \geq |x_i - x_j| - |x_j - x_{n+j}| \geq 0.75$. This holds for any $i \neq j$, which means that p_i is the nearest neighbor of p_{n+i} and vice versa, because we use the L_∞ norm. As a consequence, the marginal counts $C_x^1(p)$ are 1 for all $p \in P$, independent of the existence of duplicates in A.

If A does not contain any duplicates, it follows that $|y_i - y_j| = |y_{n+i} - y_{n+j}| \geq 1$, since A only contains integers. By the same arguments as above it follows that $|y_i - y_{n+j}| \geq 0.75$ and that $C_y^1(p) = 1$ for all $p \in P$. We can then use these values in Equation 6 and because of $\psi(x) = \sum_{m=1}^{x-1}(\frac{1}{m}) - C$ it is:

$$\widehat{I_{KSG}}(P) = \psi(1)+\psi(|P|)-\frac{1}{1}-\frac{1}{|P|}\sum_{m=1}^{|P|}\psi(1)+\psi(1) = \sum_{m=1}^{|P|-1}\left(\frac{1}{m}\right)-1 \tag{10}$$

Conversely, if A contains the duplicates $a_i = a_j$, it is $|y_i - y_j| = 0$ and $|y_i - y_{n+j}| = 0.25$ and thus $C_y^1(p_i) \geq 3$. Because of $\psi(x+1) = \psi(x) + \frac{1}{x} > \psi(x)$ for all $x \geq 0$, it is, analogously to Equation 10, $\widehat{I_{KSG}}(P) < \sum_{m=1}^{|P|-1}(\frac{1}{m}) - 1$. This concludes the subproof. ■

It is clear that P can be constructed in time $O(|A|)$, which means $|P| \in O(|A|)$. Note that this reduction works analogously for any fixed $k > 0$ by placing $k-1$ points evenly spaced on the diagonal between each pair p_i and p_{n+i}. Because k is fixed, the size of P increases only by a constant factor. After computing $\widehat{I_{KSG}}(P)$, the result is only compared to a sum over $|P|$ numbers. Therefore the complexity of the reduction is in $O(n)$. This means that determining $\widehat{I_{KSG}}(P)$ has a lower bound of $\Omega(n \log n)$. □

A.3 Proof of Theorem 5.1

Theorem 5.1. *Let $P \subseteq \mathbb{R}^2$ be a set of points. For any point $p \in P$ there exist at most $8k$ points $q \in P$ such that p is one of the k nearest neighbors of q using the L_∞-norm.*

Proof. Let $p = (x, y) \in P$ be a point and $Q \subseteq P$ be the set of points such that for each point $q \in Q$, p is one of the k nearest neighbors of q. We separate Q into eight sets based on their relative location to p, as illustrated in Figure 12. There are four axis-aligned rays $Q_L, Q_R, Q_U, Q_D \subseteq Q$ centered at p such that points on any of these rays share one component with p and differ in the other one. Additionally, there are four quadrants $Q_{RU}, Q_{LU}, Q_{LD}, Q_{RD} \subseteq Q$ centered at p excluding the axis-aligned rays. Because p cannot be its own nearest neighbor, these eight sets partition Q. To prove the lemma we proceed to show that each of these eight sets contains at most k points.

Let $r = (x_r, y_r)$ be the most distant point to p in the axis-aligned ray Q_R, that is, $|x - x_r| = \max_{(x_i, y_i) \in Q_R} |x - x_i|$. Then all other points in Q_R are on the line between p and r and thus closer to r than p. This means that Q_R cannot contain more than k points, because p would not be a nearest neighbor of r otherwise. By symmetry, this result also holds for the sets Q_L, Q_U, Q_D.

Similarly, Let $r = (x_r, y_r)$ be the most distant point to p in the quadrant Q_{RU} and let Δ be that distance. More formally, it is

$$\Delta = \max(|x - x_r|, |y - y_r|) = \max_{(x_i, y_i) \in Q_{RU}} \max(|x - x_i|, |y - y_i|).$$

An exemplary illustration can be found in Figure 13 with the set $Q_{RU} = \{s, r\}$. For any other point $q_i = (x_i, y_i) \in Q_{RU}$ with $q_i \neq r$ it is $x < x_i \leq x + \Delta$ and $y < y_i \leq y + \Delta$, because r is the point most distant to p. Figure 13 illustrates this by delimiting the area in which all points of Q_{RU} lie with dashed lines. Because of $x_r > x$ and $y_r > y$, it follows that $|x_r - x_i| < \Delta$ and $|y_r - y_i| < \Delta$. This means that q_i is a nearest neighbor of r. Figure 13 shows this by highlighting the area of nearest neighbors of r with dotted lines. Analogously to the axis-aligned rays, Q_{RU} cannot contain more than k points, because p would not be a nearest neighbor of r otherwise. By symmetry, this result also holds for the sets Q_{LU}, Q_{LD}, Q_{RD}. □

B ADEMI

Data Structure 5: ADEMI

struct {
  real x, y
  real $\epsilon_P^k, \epsilon_X^k, \epsilon_Y^k$
} *DemiPoint*;

struct {
  DemiPoint[ ] P_D
  BST<DemiPoint*> T_x, T_y
  real *base*, *sum*
  2D dynamic range tree T_{range}
  2D dynamic segment tree T_{seg}
} *state*;

Algorithm 6: ADEMI-INSERT(S_P, p_{n+1})

Compute $\epsilon_{P'}^k(p_{n+1})$	$O(k \cdot \log n \log\log n)$
Compute $\epsilon_{X'}^k(x_{n+1})$ and $\epsilon_{Y'}^k(y_{n+1})$	$O(\log n)$
Insert p_{n+1} into P_D	$O(1)$
Reference p_{n+1} in T_x, T_y	$O(\log n)$
Insert p_{n+1} into T_{range}	$O(\log n \log\log n)$
Insert $square(p_{n+1}, P')$ into T_{seg}	$O(\log n \log\log n)$
$base \leftarrow base + \frac{1}{n}$	$O(1)$
$sum \leftarrow sum + \log\left(\frac{\epsilon_{X'}^k(x_{n+1}) \cdot \epsilon_{Y'}^k(y_{n+1})}{(\epsilon_{P'}^k(p_{n+1}))^2}\right)$	$O(1)$
$A \leftarrow \{p_i \in P\colon \max(\lvert x_i - x_{n+1}\rvert, \lvert y_i - y_{n+1}\rvert) < \epsilon_P^k(p_i)\}$	$O(\log n \log\log n + \lvert A\rvert)$
$B \leftarrow \{p_i \in P\colon \lvert x_i - x_{n+1}\rvert < \epsilon_X^k(x_i)\}$	$O(k \cdot \log n)$
$C \leftarrow \{p_i \in P\colon \lvert y_i - y_{n+1}\rvert < \epsilon_Y^k(y_i)\}$	$O(k \cdot \log n)$
forall $p_i \in A$ **do**	
Delete $square(p_i, P)$ from T_{seg}	$O(\lvert A\rvert \cdot \log n \log\log n)$
Compute $\epsilon_{P'}^k(p_i)$	$O(\lvert A\rvert \cdot k \cdot \log n \log\log n)$
Insert $square(p_i, P')$ into T_{seg}	$O(\lvert A\rvert \cdot \log n \log\log n)$
$sum \leftarrow sum + \log((\epsilon_P^k(p_i))^2) - \log((\epsilon_{P'}^k(p_i))^2)$	$O(\lvert A\rvert)$
forall $p_i \in B$ **do**	
Compute $\epsilon_{X'}^k(x_i)$	$O(\lvert B\rvert \cdot k \cdot \log n)$
$sum \leftarrow sum - \log(\epsilon_X^k(x_i)) + \log(\epsilon_{X'}^k(x_i))$	$O(\lvert B\rvert)$
forall $p_i \in C$ **do**	
Compute $\epsilon_{Y'}^k(y_i)$	$O(\lvert C\rvert \cdot k \cdot \log n)$
$sum \leftarrow sum - \log(\epsilon_Y^k(y_i)) + \log(\epsilon_{Y'}^k(y_i))$	$O(\lvert C\rvert)$

ACKNOWLEDGMENTS

This work was partially supported by the DFG Research Training Group 2153: "Energy Status Data – Informatics Methods for its Collection, Analysis and Exploitation"

REFERENCES

[1] Michael Ben-Or. 1983. Lower bounds for algebraic computation trees. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing*. 80–86.
[2] Jon Louis Bentley. 1977. *Algorithms for Klee's rectangle problems.* Technical Report. Technical Report, Computer.
[3] 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.
[4] Brian Bole, Chetan S Kulkarni, and Matthew Daigle. 2014. Adaptation of an electrochemistry-based li-ion battery model to account for deterioration observed under randomized use. In *Proceedings of Annual Conference of the Prognostics and Health Management Society*, Vol. 29. https://ti.arc.nasa.gov/c/25/
[5] Amit Chakrabarti, Graham Cormode, and Andrew McGregor. 2007. A near-optimal algorithm for computing the entropy of a stream. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '07)*. Society for Industrial and Applied Mathematics, 328–335.
[6] Timothy M Chan. 2006. A dynamic data structure for 3-d convex hulls and 2-d nearest neighbor queries. In *Proceedings of the 17th annual ACM-SIAM Symposium on Discrete Algorithm (SODA'06)*. 1196–1202.
[7] Kenneth Ward Church and Patrick Hanks. 1990. Word association norms, mutual information, and lexicography. *Computational Linguistics* 16, 1 (1990), 22–29.
[8] Thomas M. Cover and Joy A. Thomas. 2006. *Elements of information theory* (2. ed. ed.). Wiley-Interscience, Hoboken, NJ.
[9] Dafydd Evans. 2008. A computationally efficient estimator for mutual information. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, Vol. 464. The Royal Society, 1203–1215.
[10] Nicholas JA Harvey, Jelani Nelson, and Krzysztof Onak. 2008. Sketching and streaming entropy via approximation theory. In *IEEE 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS '08)*. IEEE, 489–498.
[11] Sanjiv Kapoor and Michiel Smid. 1996. New techniques for exact and approximate dynamic closest-point problems. *SIAM J. Comput.* 25, 4 (1996), 775–796.
[12] 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 (SSDBM'15)*. ACM.
[13] 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. *Phys. Rev. E* 76 (2007), 15. Issue 2.
[14] Justin B Kinney and Gurinder S Atwal. 2014. Equitability, mutual information, and the maximal information coefficient. *Proceedings of the National Academy of Sciences* 111, 9 (2014), 3354–3359.
[15] Yuliya Kopylova, Duncan A Buell, Chin-Tser Huang, and Jeff Janies. 2008. Mutual information applied to anomaly detection. *Journal of Communications and Networks* 10, 1 (2008), 89–97.
[16] LF Kozachenko and Nikolai N Leonenko. 1987. Sample estimate of the entropy of a random vector. *Problemy Peredachi Informatsii* 23, 2 (1987), 9–16.
[17] Alexander Kraskov, Harald Stögbauer, and Peter Grassberger. 2004. Estimating mutual information. *Phys. Rev. E* 69 (2004), 16. Issue 6.
[18] M. Lichman. 2013. UCI Machine Learning Repository. (2013). https://archive.ics.uci.edu/ml/machine-learning-databases/00235/
[19] Anna Lubiw and András Rácz. 1991. A lower bound for the integer element distinctness problem. *Information and Computation* 94, 1 (1991), 83–92.
[20] Kurt Mehlhorn and Stefan Näher. 1990. Dynamic fractional cascading. *Algorithmica* 5, 1 (1990), 215–241.
[21] Angeliki Papana and Dimitris Kugiumtzis. 2009. Evaluation of mutual information estimators for time series. *International Journal of Bifurcation and Chaos* 19, 12 (2009), 4197–4215.
[22] Hanchuan Peng, Fuhui Long, and Chris Ding. 2005. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 27, 8 (2005), 1226–1238.
[23] Josien PW Pluim, JB Antoine Maintz, and Max A Viergever. 2003. Mutual-information-based registration of medical images: a survey. *IEEE Transactions on Medical Imaging* 22, 8 (2003), 986–1004.
[24] Peng Qiu, Andrew J Gentles, and Sylvia K Plevritis. 2010. Reducing the computational complexity of information theoretic approaches for reconstructing gene regulatory networks. *Journal of Computational Biology* 17, 2 (2010), 169–176.
[25] Thomas Schreiber. 1995. Efficient neighbor searching in nonlinear time series analysis. *International Journal of Bifurcation and Chaos* 5, 02 (1995), 349–358.
[26] Claude Elwood Shannon. 1948. A mathematical theory of communication. *The Bell System Technical Journal* 27 (1948), 379–423, 623–656.
[27] Kanat Tangwongsan, Martin Hirzel, Scott Schneider, and Kun-Lung Wu. 2015. General incremental sliding-window aggregation. *Proceedings of the VLDB Endowment* 8, 7 (2015), 702–713.
[28] Martin Vejmelka and Kateřina Hlaváčková-Schindler. 2007. Mutual information estimation in higher dimensions: A speed-up of a k-nearest neighbor based estimator. In *International Conference on Adaptive and Natural Computing Algorithms (ICANNGA'07)*. 790–797.
[29] Janett Walters-Williams and Yan Li. 2009. Estimation of mutual information: A survey. In *International Conference on Rough Sets and Knowledge Technology (RSKT'08)*. 389–396.
[30] Dan E Willard. 1985. New data structures for orthogonal range queries. *SIAM J. Comput.* 14, 1 (1985), 232–253.