K-Anonymity and L-Diversity on Streamed Big Data by Philip Claesson, Moritz Meister

K-Anonymity and L-Diversity on Streamed Big Data

Title:	K-Anonymity and L-Diversity on Streamed Big Data
Authors:	Philip Claesson, Moritz Meister
Version:	1.0
Date:	10-March-2018
Download page:	https://github.com/moritzmeister/FlinkAnonymity
Copyright:	Copyright © 2018, Philip Claesson, Moritz Meister – All rights reserved

Contents

List of Figures

List of Tables

1 Introduction

Nowadays, more and more public but also private organizations wish to publish their data in order to allow other parties to perform research, such that all parties can benefit from the synergies and the resulting insights or in order to outsource a task related to the data to an external service provider. However, in a time when personal data that is handled by both private and public actors is getting larger and more sophisticated, the threat against individual privacy has become more and more significant, as the motivation to breach privacy increases. Therefore, special care needs to be taken in the handling and especially publishing of data sets containing personal data, such that individuals can not be uniquely identified.

On first sight it seems sufficient to simply remove or replace attributes from the data, that are directly identifying the individual. However, this pseudonymised published data may be de-anonymised by re-identifying individuals through quasi-identifiers (QID) and combining the data with external data bases or background knowledge (Zhou et al., 2009). Let us illustrate such an attack by means of an example:

Example 1 (Re-identification attacks)**.** *Suppose there is a table T with attributes (postcode, birthplace, birthdate, health) that is being publicly available. An attacker might possess background information in the form of a table B = (name, postcode, birthplace, birthdate) about the individuals. Then a simple join of T and B on the common attributes might reveal the individuals identities and the attacker gains access to the sensitive attribute "health". The common attributes between T and B are called quasi-identifier (QID) in this case.*

There has been extensive research in the field of privacy preserving data publishing in order to protect static data sets from these kinds of attacks. Among many, two major methods/algorithms evolved namely *k-anonymity* (Sweeney, 2002) and *l-diversity* (Machanavajjhala, Gehrke, Kifer, & Venkitasubramaniam, 2006). However, with the amount of data steadily growing, a new field with need for research emerged, that is the field of *continuous privacy preserving data publishing*. The challenge in a continuous setting is that there is no single static data set with a fixed number of records, but a stream of records arriving at varying velocities and speed, which is possibly never ending. Hence, the traditional *k-anonymity* and *l-diversity* can't be applied in a straightforward way (Zhou et al., 2009). Next to information loss it is essential for publishing a data stream, to publish the records in a timely way after it arrives, if possible synchronous to the arrival. That is because often the data loses it's value if it takes to long to be published. Therefore, there is a need for an anonymisation technique with low latency and low information loss.

Additionally, in the past few years a great variety of frameworks emerged that allow for distributed, high-performing, always available and accurate data streaming applications, such a framework is the by the Apache Foundation published open source project *Apache Flink*. In this study we leveraged the Flink functionality to the widest extent possible in order to implement a variation of the *k-anonymity* and *l-diversity* algorithms, which is running in parallel, applicable to data streams and promises a minimal information loss.

This report is structured as follows: Firstly, we will provide a short overview over existing research in the field and further define the problem and the existing algorithms *k-anonymity* and *l-diversity*. Subsequently, the paper describes how we adapted the algorithms to the streaming setting and define in detail the new algorithm. This is followed by a brief description of the architecture of the solution with respect to *Apache Flink* functionality. The third section contains the results of an evaluation study of the algorithm mainly with respect to latency. The report will conclude with a discussion on strengths and weaknesses of the approach and further possibilities for research.

2 Background

A Static data sets

The possibility to de-anonymise data was introduced and demonstrated by Sweeney (2000). In the study, Sweeney used public 1990 U.S. Census summary data to investigate the potential of identifying unique individuals. It was found that 87% of the population in the United States had reported characteristics that likely made them unique based on the quasi-identifier attributes 5-digit ZIP2, gender, date of birth (Sweeney, 2000).
In order to develop a method to protect released data against re-identification attacks, Sweeney (2002) introduced the method *k-anonymity*:

Definition 1 (k-Anonymity)**.** Let RT$(A_1, ..., A_n)$ be a table of n attributes and QID_{RT} be the quasi-identifier associated with it, that is, a subset of the attrib. RT is said to satisfy k-anonyutesmity if and only if each sequence of values in RT[QID_{RT}] appears with at least k occurrences in RT[QID_{RT}]. (Sweeney, 2002)

The quasi-identifier QID_{RT} is the minimal set of attributes $QID_{RT} \subset (A_1, ..., A_n)$ that can be used in an re-identification attack to join the data with external information in order to uniquely identify individuals. That means that k-Anonymity guarantees privacy protection by making each combination of values of the quasi-identifier indistinguishable from at least $k - 1$ other individuals. In traditional database domains, k-Anonymity has been proved efficient for privacy protection (Li, Chin Ooi, & Wang, 2008).

Example 2 (3-Anonymization)**.** *Let us consider an example for the QID {postcode, occupation, age} and sensitive attribute {income}:*

postcode	occupation	age	income
10230	high school teacher	32	65 k
21032	physician	42	95 k
21030	dentist	40	110 k
10210	primary school teacher	38	58 k
21030	surgeon	46	150 k
10285	sessional instructor	28	42 k

Table 1: Table T of micro data

The following generalization of the table is 3-anonymous and therefore an attacker with background information can't reliably identify the individuals:

postcode	occupation	age	income
102**	teacher	(27-38]	65 k
2103*	medical doctor	(38-46]	95 k
2103*	medical doctor	(38-46]	110 k
102**	teacher	(27-38]	58 k
2103*	medical doctor	(38-46]	150 k
102**	teacher	(27-38]	42 k

Table 2: 3-anonymous Table T' on QID

In order to achieve k-Anonymity, generalization needs to be used. As can be seen from example 2, this is, the process of replacing values by a more general value, for example by an interval, shortening the value, a more general description or in the worst case it might be necessary to completely suppress the value and replace it entirely by an asterisk.

Even though Sweeney (2002) proves that this measure protects privacy against a variety of attacks, however Machanavajjhala et al. (2006) show that it has two major shortcomings. The first problem that they point out is the lack of diversity within the sensitive attribute. They reason that the data set can be k-anonymous, but if the k tuples of an equivalence class all have the same sensitive value, the attacker can still make inferences about an individual with the appropriate background information, they call this a **homogeneity attack**. The second shortcoming that Machanavajjhala et al. (2006) point out is that if an attacker has additional background knowledge he can detect the sensitive information. Referring to example 2, assume a friend of yours, that means having this knowledge, is a high school teacher in the postcode area 10230 with age 32. Additionally, you know that any high school teacher starts with a salary higher than 60 k, now even if you possess only the generalized table T', you can still infer your friend's exact salary. Machanavajjhala et al. (2006) call this a **background knowledge attack**.

By showing that the ideal definition of privacy called *bayes-optimal*, where both, the data publisher and the adversary, have full (and identical) background knowledge, can hardly be guaranteed in a real-world context, Machanavajjhala et al. (2006) arrive at a practical principle, they call l-diversity:

Definition 2 (l-Diversity principle)**.** A equivalence class, that is a group of tuples with the same generalized QID, is l-diverse if it contains at least l "well-represented" values for the sensitive attribute S. A table is l-diverse if every equivalence class is l-diverse. (Machanavajjhala et al., 2006)

Based on this principle Machanavajjhala et al. (2006) define five instantiations of l-Diversity. With these definitions, they prove that l-diversity provides privacy even if the data publisher lacks information about the knowledge possessed by the attacker. Furthermore, they show that existing algorithms for k-anonymity can be easily adapted to l-diversity, and with experiments they underline that their method is practical and implementable in an efficient way.

B Streamed data

The optimization of *k-anonymity* as well as *l-diversity* was proven to be NP-hard (Bayardo & Agrawal, 2005; Xiao, Yi, & Tao, 2010). In a streamed environment the methods may imply a delay of the publication of all or some tuples. This is especially critical when the utility of the the data decreases with the time that is passed in between data generation, that is, event occurrence, and the time the data is analyzed. A possible application could be a credit card provider, who wishes to outsource the task of credit risk predictions to an external organization, and therefore needs to the stream the transactional data to the service provider, if possible in real-time.

A natural extension of the two methods introduced in the previous section would be an incremental approach. That means waiting until a certain amount of data has arrived and subsequently treating this data as if it were a static data set, applying l-diversity or k-anonymity as described before. Xiao and Tao (2007) studied a situation that is closely related to the incremental approach, namely the implications of the previous section's approaches in the context of the re-publication of micro data sets that have been updated with insertions or deletions. Byun, Sohn, Bertino, and Li (2006) studied privacy attacks on re-publication of datasets with insertions only and developed a solution to effectively prevent those attacks. Xiao and Tao (2007) start from this solution and extend it by the possibility of deletions in order to preserve privacy in fully dynamic datasets. The problem inherent in re-publication is illustrated by the following example:

Example 3 (The re-publication problem of varying diversity)**.** *Consider the following equivalence class is part of a published and generalized dataset:*

postcode	occupation	age	income
2103*	medical doctor	(38-46]	95 k
2103*	medical doctor	(38-46]	110 k

Table 3: Equivalence class in first release

Now consider a republication of the dataset where the previous equivalence class becomes the following:

postcode	occupation	age	income
2103*	medical doctor	(38-46]	95 k
2103*	medical doctor	(38-46]	65 k

Table 4: Equivalence class in second release

An adversary having the following knowledge about an individual {21031, dentist, 40}, and knowing that the individual had to be present in both releases, can now infer that this individual's income must be 95 k, due to the fact that the diversity has changed.

To tackle the shortcoming illustrated in the previous example, Xiao and Tao (2007) introduced two new methods, first of all m-variance and a new generalization technique called *counterfeit generalization*. The idea behind these approaches is to add so called counterfeit tuples to equivalence classes where necessary, in order to preserve the original diversity, and add accompanying statistics about in which equivalence class these counterfeit tuples were added in order to enhance the effectiveness of data analysis. They find that the a key characteristic for preserving privacy in re-publication is to ensure a certain "in-variance" in all the equivalence classes. While this approach works well for re-publication in the case of minor insertions and incremental enhancements to data sets, for a data stream it is very impractical for various reasons. One of the reasons is that streams are characterized by high velocities and frequencies at which the data arrives, these large volumes will result in problems in the case of re-publication, since the data set will always be growing, and the previously mentioned methods will require several scans through the table to be published. Furthermore, since the utility of the data decreases with the time that has passed, there is no reason to republish the data again. The goal is to publish the data as fast as possible, one it arrived. Another reason is the characteristic of a data stream that as soon as the anonymized data stream is output, it can not be revisited again. Therefore, the previously mentioned problem of an adversary being able to analyse two releases jointly is not applicable to the streaming scenario.

Li et al. (2008) were the first researchers to adapt the k-anonymity approach entirely to a streaming setting. They propose an algorithm that guarantees k-anonymity and furthermore their algorithm poses a constraint on the degree of how much "out-of-order" the output stream is compared to the input. Additionally, the goal of the algorithm is to retain as much information in the outputted stream. Experiments show that the algorithm is effective and efficient (Li et al., 2008). Zhou et al. (2009) go further and developed an approach that considers both the distribution of the data values to be released and the statistical distribution of the data stream, which resulted in various anonymization quality measures and a group of randomized methods. With these approaches Zhou et al. (2009) tackle the problem of heavy information loss in the case of outliers, that the previous methods can't cope with. Their algorithm makes the decision whether a set of elements should be anonymized and released at a certain point of time, based on the current information loss compared to a later release, where the expected future information loss is inferred from the statistical distribution of the data in the stream.

Having discussed the pitfalls and advantages of the previously conducted research in the field, we are now ready to introduce our novel approach in the next section, that is based on a combination of the SKY-algorithm by Li et al. (2008) and the findings of Zhou et al. (2009), but extends it and applies it in a parallel computation environment. There will be more details about the two approaches provided in the description of our algorithm.

3 Approach and Algorithm

A Problem definition

A *data stream* T is an infinite time sequence of tuples $T = t_1, t_2, t_3, ...$, with an incremental order such that for two tuples t_i and t_j and $i < j$, denotes that tuple t_i arrives before t_j. A tuple t_i consists of the following attributes $t_i = \{ID, TS, A_1, ..., A_m\}$, where ID is an identity attribute to uniquely identify an individual and TS is a timestamp indicating the arrival time of the tuple. In this study the arrival time will be interchangeably used with the term ingestion time, that is the time that a tuple enters the anonymization procedure. Furthermore, let a subset $QID \subset \{A_1, ..., A_m\}$ be the quasi-identifier in the tuple.

There are a few assumptions involved in our approach: 1. In line with Zhou et al. (2009), we assume that the attributes ID and TS are not going to be published. 2. The attributes $t_i - QID - \{ID, TS\}$ are not involved in the procedure and therefore can be omitted for the sake of simplicity, however they can just be outputted with the anonymized attributes. 3. While Zhou et al. (2009) assume that there possibly are identical ID values, that is, several tuples from the same object, we will assume that each tuple comes from a unique individual. Please note that we had to take this assumption due to technical reasons in the implementation with the Apache Flink framework, since there was no straight forward implementation of non-unique IDs in the processing, however in general our approach can be extended by this feature and for now this assumption does not change the outcome of this study. 4. We assume that the utility of data in stream T is time sensitive, that is, usability of the data decreases with time passed between arrival and release. Without this assumption, that is, if the latency between data arrival and data publishing would be irrelevant, the problem would be trivial. One could simply wait until a batch of data that is large enough to be anonymized (i.e. k tuples) has arrived and thereafter periodically apply a static method to anonymize the data in batches. Furthermore, as described in the previous section, incremental approaches are not appropriate in this problem setting either, because re-publication of the entire data that has arrived up to a certain point in time is highly inefficient and will decrease the utility of the data immensely due to large latency.

The k-anonymity and l-diversity problem, that our algorithm is going to solve can now be defined as follows:

Definition 3 (k-anonymity problem for data streams)**.** Considering a data stream T, an anonymized output stream T' needs to be produced, where every equivalence class that is being released to T' needs to contain at least k indistinguishable tuples with respect to the QID.

Definition 4 (l-diversity problem for data streams)**.** Considering a data stream T, an anonymized output stream T' needs to be produced, where every equivalence class that is being released to T' needs to contain at least k indistinguishable tuples with respect to the QID and additionally in the sensitive attribute need to be l different values present.

With these definition, an attacker cannot re-identify an individual with a confidence higher than $1/k$ and furthermore she can not reliably infer the sensitive value.

B Algorithm

For the anonymization process we need to define a generalization procedure. We are adopting the approach of Li et al. (2008), and let the data publisher define a generalization hierarchy for each of the suspected attributes of the QID. Figure 1 illustrates such a generalization hierarchy for the attributes *Age* and *Education*. These trees can be defined by the user with varying depths of the tree in a file and subsequently one can choose to which level/granularity the attributes should be generalized. In the example of figure 1 for the attribute *age*, the user would have three generalization levels to choose from: In terms of our software "0" refers to the original variable without generalization. "1" would be the lowest level generalization, in the *age* example that are the 15 years intervals, level "2" would be 30 year intervals and level "3" refers to completely replacing the information by an asterisk "*" or with "Any". The user can make this choice for each attribute of the QID and therefore decide on the granularity and amount of information loss.

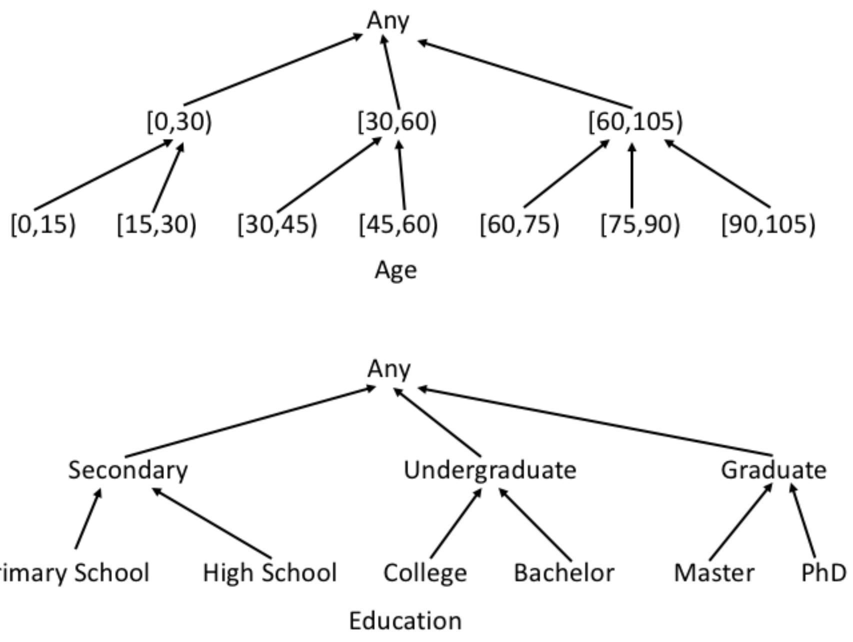

Figure 1: Example generalization hierarchies

Note that this generalization approach is rather static and not data driven. The focus of this study is to provide an approach that is parallelizable and therefore scalable to high speed data streams. Furthermore, this static approach, as we will describe in detail later provides a guarantee for constant information loss, that is controllable and robust to outliers. With this setup we are able to leverage the functionality of Apache Flink in the best way.

The novel approach that will be described in the following paragraphs is mainly based on Li et al. (2008) especially with respect to the generalization procedure, furthermore it makes use of some characteristics of the approaches of Zhou et al. (2009), but extends it by the l-diversity principle. Therefore, we will further introduce these algorithms and the differences to our approach at this point.

Let's start with the algorithm of Li et al. (2008): They introduce a data structure called the specialization tree. This tree is a combination of the generalization hierarchies (as described above) of the attributes in the quasi-identifier, where the root is the most general node. For a new arriving tuple, this specialization tree is scanned from the root until the most specific node is found, where the tuple fits in. Each node possesses a buffer, once a buffer satisfies k-anonymity, the tuples in the buffer are released and the node is marked as a work-node. The corresponding counterpart to the buffer in our approach is the window-functionality of Apache Flink, which will be described in detail later on. However, the main difference in the approach of Li et al. (2008) is that as soon as a node is marked as work-node, all arriving tuples at a work-node, will be released immediately. Note that this approach as a significant shortcoming. Assume an adversary registers that a certain equivalence class, that is, buffer of a node, was released and he knows the order or arrival time of tuples of that class arriving later, he would be able to infer the sensitive attribute of these tuples. In contrast, our approach will release tuples only in k-anonymous and l-diverse batches.

Zhou et al. (2009) adopt the same approach and only release k-anonymous batches of tuples. The main characteristic of the algorithms of Zhou et al. (2009) is the data structure they use in order to group tuples to equivalence classes and generalize them, namely a R-Tree. Once a tuple arrives they insert the tuple into the R-Tree, therefore grouping similar tuples together into equivalence classes. The R-Tree is a dynamic index structure for multi-dimensional information and since it is dynamic, it is constantly reorganizing its nodes, which means that nodes are split at the insertion of new tuples, therefore always grouping tuples with the aim to minimize information loss (Guttman, 1984). Zhou et al. (2009) start with a randomized approach, where the probability that a equivalence class C, that is, a leaf node in the R-tree, is $Pr(C) = \frac{\beta}{loss(C)}$ to be published. The important thing to note is that the larger the information loss is, the smaller the probability for release is. On the other hand, β indicates how long the leaf node stays in the R-Tree, that is the longer, the higher the probability of release. Subsequently, they introduce further measures and modifications to this algorithm in order to take into account the density distribution of attributes, or they hold tuples in dense areas to achieve higher anonymization quality. Basically the rationale behind this is that every split of leaf nodes in the R-tree leads to lower information loss, since it gets more granular. Therefore, if the distribution indicates that future tuples might lead to a split of leafs, it can be worth to wait until such tuples have arrived. Figure 2 illustrates the results for varying k. As can be seen, while certainly the anonymization security increases with k, so does the information loss in their approaches. This is the fact that we try to tackle in our approach by deploying a maybe simple generalization procedure, but at the same time being able to constrain information loss.

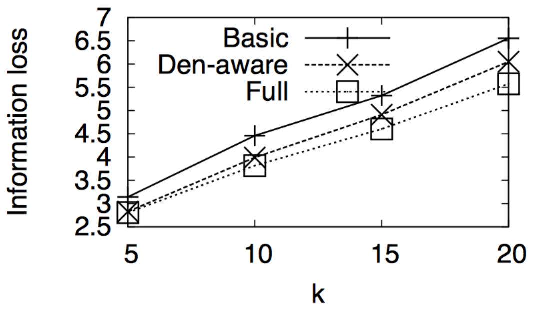

Figure 2: Information loss for different k in the algorithms of Zhou et al. (2009)

Before we introduce the algorithm, let us clarify some notation: For a set S of j sensitive attributes, let $|S_m|$ denote the number of unique combinations of values for the j attributes within some equivalence class h. For a tuple t_i of the stream, let g_i represent the generalized version of the tuple, which is retrieved by calling the function $generalize(t_i, QID)$, where QID is the quasi-identifier set of attributes previously defined and with the corresponding generalization hierarchies. Furthermore, let q_i be the values of the quasi-identifier for a tuple t_i **after the generalization**, that is, these values are part of g_i. Lastly, let each $m \in M$ be a collection of generalized tuples g_i, where each g_i will be mapped to a collection m, with q_i as mapping key. Each of the $m \in M$ runs in one of p distributed and parallel processes. Then the algorithm can be defined as below.
The most important characteristic of the algorithm that needs to be noted, is that the generalization of

the tuples happens right when they arrive. This is only possible due to the nature of the static generalization, since one needs to be able to generalize single tuples. While this is a very restrictive necessity, it offers a number of advantages: 1. Tuples are grouped by the generalization of the quasi-identifier, that ensures that only tuples are grouped together into an equivalence class which are very similar in their non-generalized version. This ensures robustness against outliers and therefore minimal information loss, given the quasi-identifier generalization. In the traditional approaches mentioned in the background section, the only criteria to group tuples together, is that they can't come from the same individual, this can lead to equivalence classes, that differ significantly in the values of the quasi identifier. However, these groups need to be generalized such that the tuples within such an equivalence class are indistinguishable. This can lead to a large information loss, to illustrate the problem consider for example two tuples one with Age 18 and another individual that has age 83, now if these two tuples are in the same equivalence class, it is impossible to find a generalization that preserves the information. The only way is to make very large interval, for example (15,85] or to replace the value by an asterisk "*". In comparison, since in our approach the tuples are grouped together only if they are very similar in their ungeneralized form and identical in their generalized form, this problem is prevented. Therefore, the approach is more robust to outliers and given a certain generalization, the information loss is minimal. 2. The static generalization guarantees constant information loss, since no matter in which order or how the tuples arrive, they are always generalized in the same way, that is generalizing two identical tuples will always be generalized in the same way and with the same information loss. 3. It enables us to distribute the process in a parallel way. The practical drawback of this approach, however, is that tuples with a less common quasi-identifier generalization will suffer from very long delays, even to the extent that the tuple can be considered as stuck. The number of tuples in the system at any given time has an upper bound induced by the number of unique possible combinations of the generalized quasi-identifier. In section 4 these potential advantages and drawbacks will be analyzed, and possible solutions discussed.

Algorithm 1 L-Diversity

Input: A stream T of tuples to be anonymized, each tuple $t_i \in T$, where i is the arrival time and with a set of attributes QID, being the quasi-identifier and a set of j sensitive attributes S. Parameters k, l and p, where p is the number of parallel processes.
Output: a stream T' of l-diverse generalizations of T

```
 1: procedure L-DIVERSIFY($T, k, l, p$)
    Initialization
 2:     Initialize $QID$ as a set of generalization hierarchies
 3:     Initialize set of collections $M = \emptyset$
    When a tuple $t_i \in T$ arrives at instant $i$
 4:     let $g_i$ := $generalize(t_i, QID)$                    ▷ Generalize the tuple
 5:     retrieve $m \in M$ where $q_i$ maps to $m$     ▷ Check if collection corresponding to $q_i$ exists
 6:     if there is no such $m$ then
 7:         create $m$
 8:     $m \leftarrow g_i$                                  ▷ Assign generalized tuple to collection
 9:     assign $m$ to one of the $p$ parallel processes
10:     $M \leftarrow m$                                    ▷ Put $m$ into the set $M$
11:     if $|m| > k$ and $|S_m| \geq l$ then      ▷ If $m$ satisfies $k$-anonymity and $l$-diversity
12:         release all $g \in m$ to output stream $T'$
13:         remove $m_h$ from $M$
```

C Solution architecture in Apache Flink

This section serves to give a short description, of how the algorithm aligns with the functionality of Apache Flink and how the main parts have been implemented with the framework.

C.1 Data types

Given that the *adult* data set from the UCI Machine Learning repository was used as a starting point of this study, the data is encapsulated in a class called *AdultData* (Dheeru & Karra Taniskidou, 2017). Figure 3 depicts class diagram of the data types that were implemented.
In later phases of the study the data set was replaced by a continuous data source that is creating tuples with the same value distributions as the original data set, in order to simulate longer streams and conduct the scalability and latency experiments in section 4.

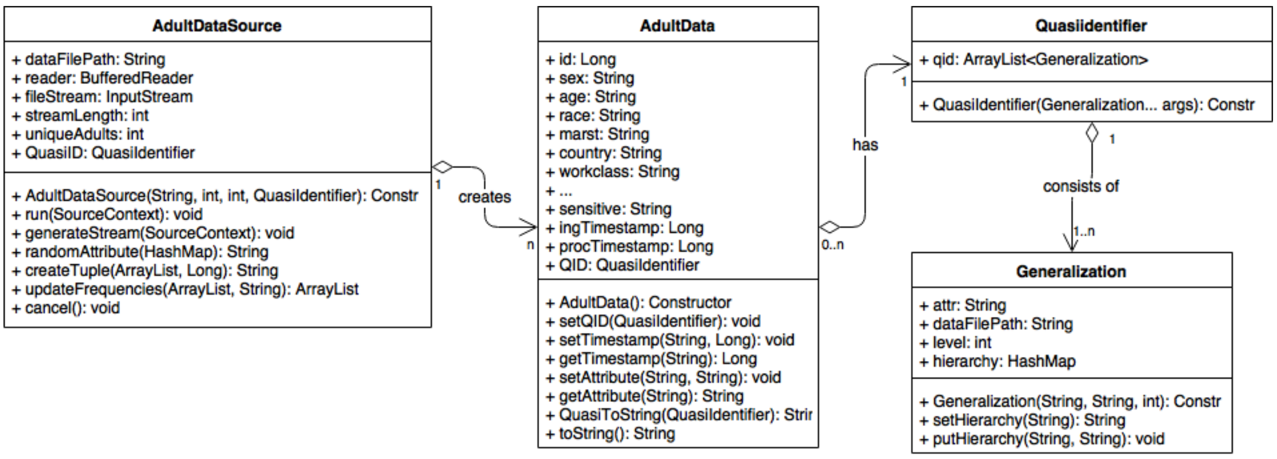
AdultDataSource
+ dataFilePath: String
+ reader: BufferedReader
+ fileStream: InputStream
+ streamLength: int
+ uniqueAdults: int
+ QuasiID: QuasiIdentifier
+ AdultDataSource(String, int, int, QuasiIdentifier): Constr
+ run(SourceContext): void
+ generateStream(SourceContext): void
+ randomAttribute(HashMap): String
+ createTuple(ArrayList, Long): String
+ updateFrequencies(ArrayList, String): ArrayList
+ cancel(): void
1
creates
n
AdultData
+ id: Long
+ sex: String
+ age: String
+ race: String
+ marst: String
+ country: String
+ workclass: String
+ ...
+ sensitive: String
+ ingTimestamp: Long
+ procTimestamp: Long
+ QID: QuasiIdentifier
+ AdultData(): Constructor
+ setQID(QuasiIdentifier): void
+ setTimestamp(String, Long): void
+ getTimestamp(String): Long
+ setAttribute(String, String): void
+ getAttribute(String): String
+ QuasiToString(QuasiIdentifier): Strir
+ toString(): String
0..n
has
1
QuasiIdentifier
+ qid: ArrayList<Generalization>
+ QuasiIdentifier(Generalization... args): Constr
1
consists of
1..n
Generalization
+ attr: String
+ dataFilePath: String
+ level: int
+ hierarchy: HashMap
+ Generalization(String, String, int): Constr
+ setHierarchy(String): String
+ putHierarchy(String, String): void

Figure 3: Class diagram of the implemented data types

C.2 Logic and Class implementations

The core part of the implementation that ensures the parallelization of the algorithm is the piece of code in fig. 4. Figure 5 illustrates how the anonymization is implemented in Apache Flink. In this section we will comment on the code and usage of custom and Flink classes.

Class	Extends/Implements	Functionality
KeyedJob	-	Main Class
AdultData	-	A data type to contain the adult data tuples.
AdultDataSource	SourceFunction	Reads the adult data file, creates an extended stream of data.
lDiversityTrigger	Trigger	Asserts if a given GlobalWindow is K-anonymous and L-diverse, triggers Release process if so.
Release	ProcessFunction	When triggered, releases all AdultData objects in the triggered window.
ProcessTimestamp	ProcessFunction	Adds a time stamp to each element in a window. Is used to add ingestion time stamps to AdultData objects.
Generalization	Serializable	A data type to contain a generalization given a hierarchy and a generalization level.
QuasiIdentifier	Serializable	A data type containing a Quasi Identifier consisting by several Generalizations
QidKey	KeySelector	A function to return a key that will be used in the keyBy(), in this case the Quasi Identifier concatenated as a string.
Generalize	MapFunction	Maps the ungeneralized value of the Quasi Identifier to the respective value in the generalization hierarchy.

Table 5: Explanation of implemented classes

```
DataStream<AdultData> output = tsGenData
        .keyBy(new QidKey())
        .window(GlobalWindows.create())
        .trigger(lDiversityTrigger.of(k, l))
        .process(new Release());
```

Figure 4: Extract from KeyedJob.java

In the code executed in fig. 4, a datastream is defined. First of all, the datastream is keyed by Quasi Identifier generalization through Flink's *keyBy* function with the QidKey class as argument. Secondly, Flink's *GlobalWindows* are used to keep track of the number of tuples in a key and the number of different sensitive values present in a equivalence class. Quoting Flink's documentation, a global window has the following characteristics: "A global windows assigner assigns all elements with the same key to the same single global window. This windowing scheme is only useful if you also specify a custom trigger. Otherwise, no computation will be performed, as the global window does not have a natural end at which we could process the aggregated elements." (*Apache Flink v1.3 Documentation*, 2018) Thirdly, the data stream is assigned a custom implemented trigger, *lDiversityTrigger*, which asserts if a given *GlobalWindow* is K-anonymous and L-diverse. Lastly, the data stream is assigned the *Release* function which upon being triggered by the *lDiversityTrigger* collects the tuples of a *GlobalWindow* to the output stream.

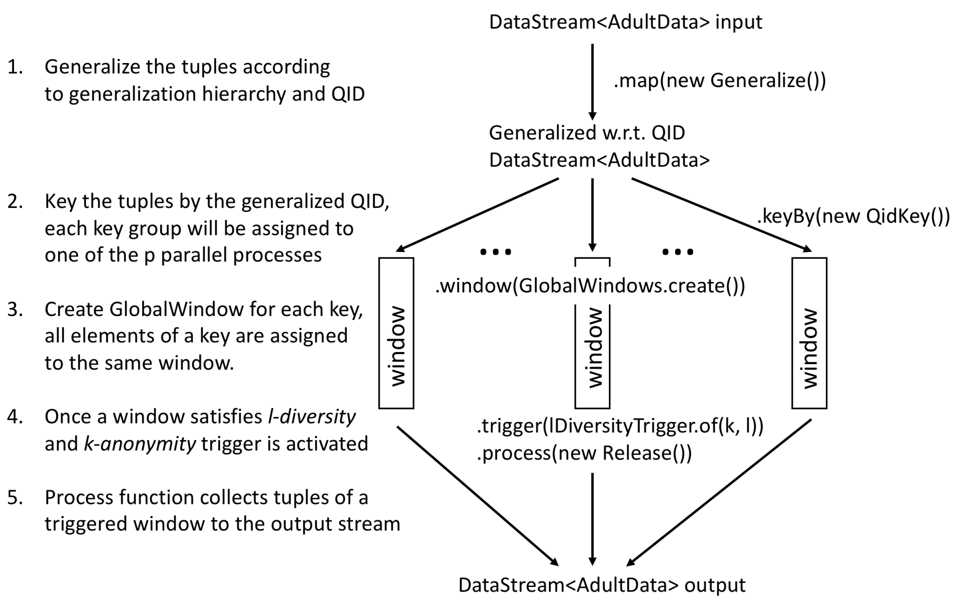
DataStream<AdultData> input
.map(new Generalize())
Generalized w.r.t. QID
DataStream<AdultData>
.keyBy(new QidKey())
.window(GlobalWindows.create())
window
window
window
.trigger(lDiversityTrigger.of(k, l))
.process(new Release())
DataStream<AdultData> output
1. Generalize the tuples according to generalization hierarchy and QID
2. Key the tuples by the generalized QID, each key group will be assigned to one of the p parallel processes
3. Create GlobalWindow for each key, all elements of a key are assigned to the same window.
4. Once a window satisfies l-diversity and k-anonymity trigger is activated
5. Process function collects tuples of a triggered window to the output stream

Figure 5: Implementation in Apache Flink

4 Evaluation

This section contains the experimental study surrounding our proposed algorithm. The aim of the experiments is to evaluate: 1. How the latency differs by different values of k, l and p, and 2. How the setting of the parameters affects the amount of tuples getting stuck in the system, as well as discussing a possible solution to this shortcoming. As described earlier, due to the static generalization, there is no point in evaluating the information loss, since every tuple would be generalized in the same way, independent of the parameters, the information loss is constant. The only thing that can affect information loss is therefore the composition of the quasi-identifier, but since this is not a parameter that in reality can be chosen by the data publisher, we are not concerned with this scenario.

A Data and experimental design

A.1 Data and experiment setup

The *adult* data set from the UCI Machine Learning repository was used as a starting point of these experiments (Dheeru & Karra Taniskidou, 2017). Extraction of this data set was originally done by Barry Becker from the 1994 Census database. The distributions and unique values of this data set were used to create a data stream with the same characteristics of n tuples.

Data set characteristics	Attribute characteristics	Number of instances	Number of attributes
Multivariate	Categorical, Integer	48842	14

Table 6: Characteristics of the *adult* data set

For comparability, all evaluations were done with the quasi-identifier *(age, educ, marital status)* with the generalization hierarchy levels described in table 7.

Identifier	Generalization Level	Number of classes
Age	2	10
Education	2	3
Marital Status	1	2
Sensitive attribute (not part of QID)	-	10

Table 7: Quasi-Identifier generalization

The experiments were conducted on Apache Flink (version 1.3) in "mini cluster"-mode on a mid 2015 Apple Macbook Pro with 2,2 GHz Intel Core i7 processor and 16GB 1600 MHz DDR3 RAM.

A.2 Parameters

The parameters subject to variation in the experiments were the following:

k - The k-anonymity constraint.

l - The l-diversity constraint.

n - The stream length, i.e. the number of tuples in the stream.

p - The parallelism, i.e. the number of parallel processes in Apache Flink.

B Experiments and results

B.1 Overall speed

This experiment serves to show the overall speed of the anonymization process for different parameters. This will later serve as an indicator of algorithm efficiency, since a data stream would possibly never ending in practice.

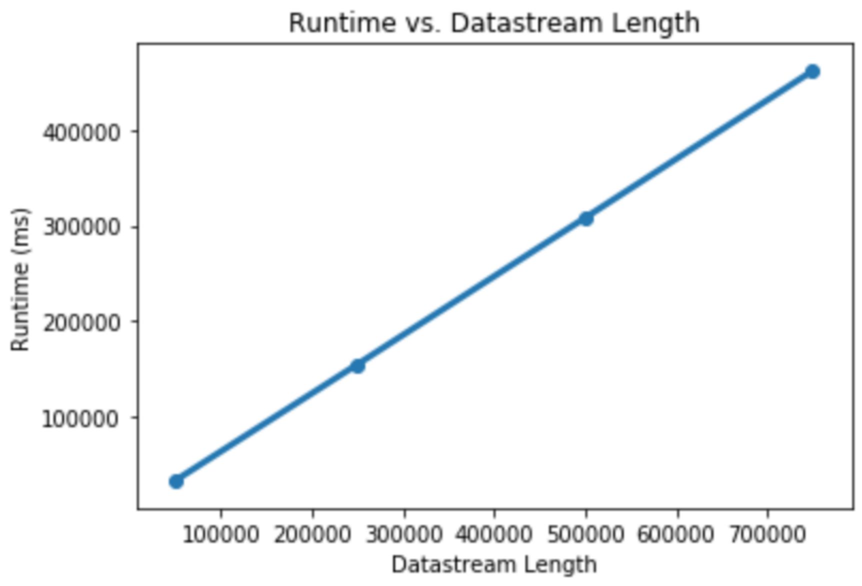

Figure 6: Run time (ms) for data streams of length n = *50 000, 250 000, 500 000 and 750 000* $(k = 40, l = 5, p = 10)$.

As can be seen from figure 6, the overall running time increases linearly with n.

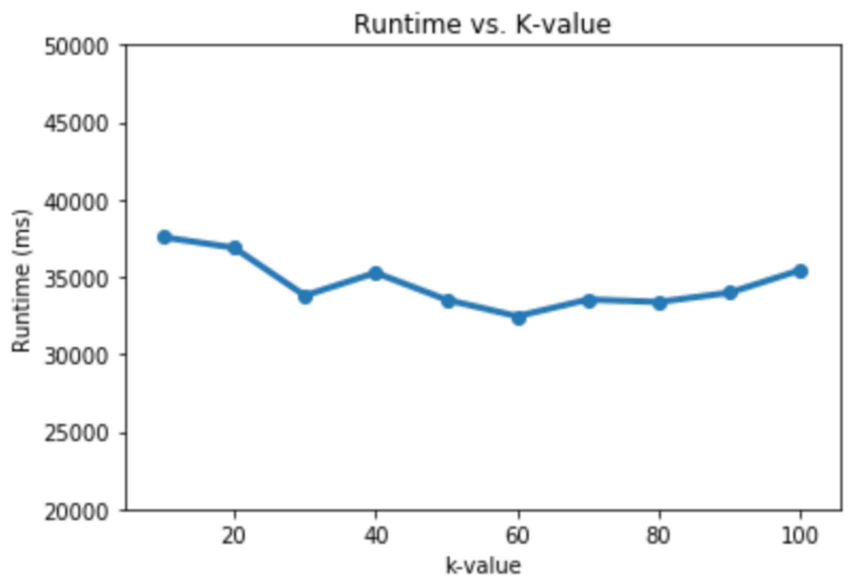

Figure 7: Run time (ms) for values of *10* $\leq k \leq 100$ and $n = 50000, l = 5, p = 10$.

As can be seen from figure 7, the overall running time is more or less constant for different values of k.

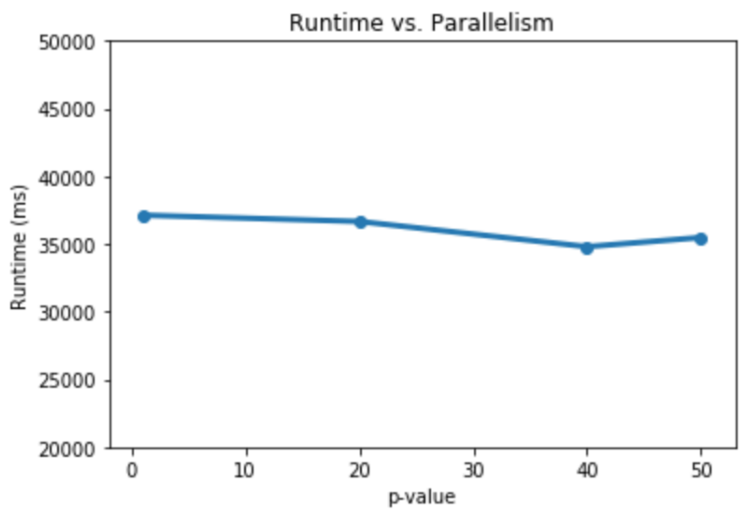

Figure 8: Run time (ms) for values of *p = 1, 10, 20, 30, 40, 50* and $n = 50000, l = 5, k = 40$.

As can be seen from figure 8, the overall running time is more or less constant for different values of p.

B.2 Tuple Delay

For each tuple processed in accordance to the algorithm in section 3.B, a delay is induced. While some delay is inescapable, too much delay might render the data to be inaccurate or irrelevant by the time it is omitted. In the first part of the experiments, we aim to establish a better understanding of how the delay varies depending on the different variables *k, l* and *p*. The tuple delay is measured through adding an injection timestamp to each AdultData-object upon tuple arrival, and adding a processing timestamp just before the object is emitted. In these experiments the stream length *n* was fixed to *n = 50000.*

K-anonynymity

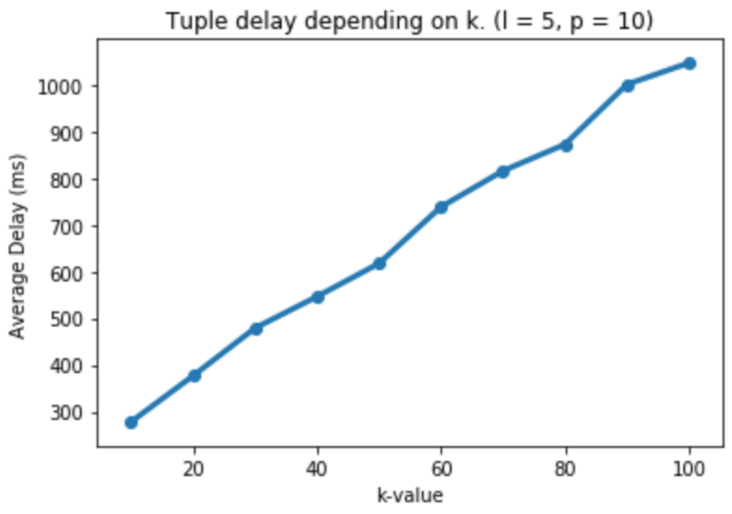

Figure 9: Average tuple delay depending on k, *(l = 5, p = 10)*

As can be seen from figure 9, the average delay of the tuples increases linearly with k.

L-diversity

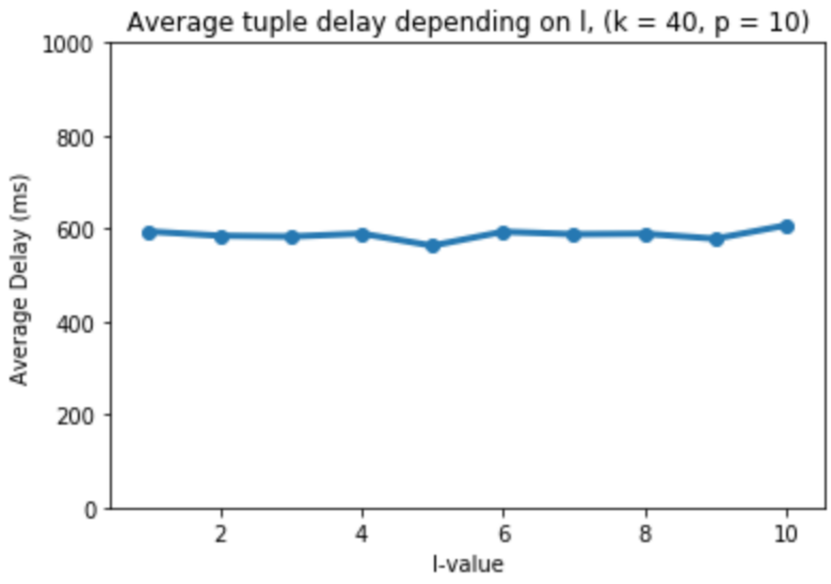

Figure 10: Average tuple delay depending on l, *(k = 40, p = 10)*

As can be seen from figure 10, the average delay was more or less constant for different values l.

Parallelism

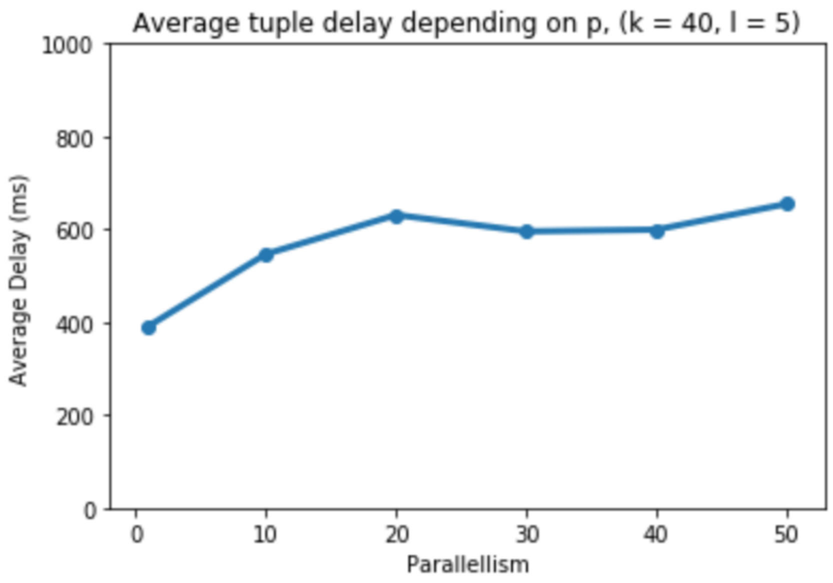

Figure 11: Average tuple delay depending on p, *(k = 40, l = 5)*

As can be seen from figure 10, the experience could show no trivial correlation between the average tuple delay and the number of parallel processes. l.

B.3 Stuck Tuples

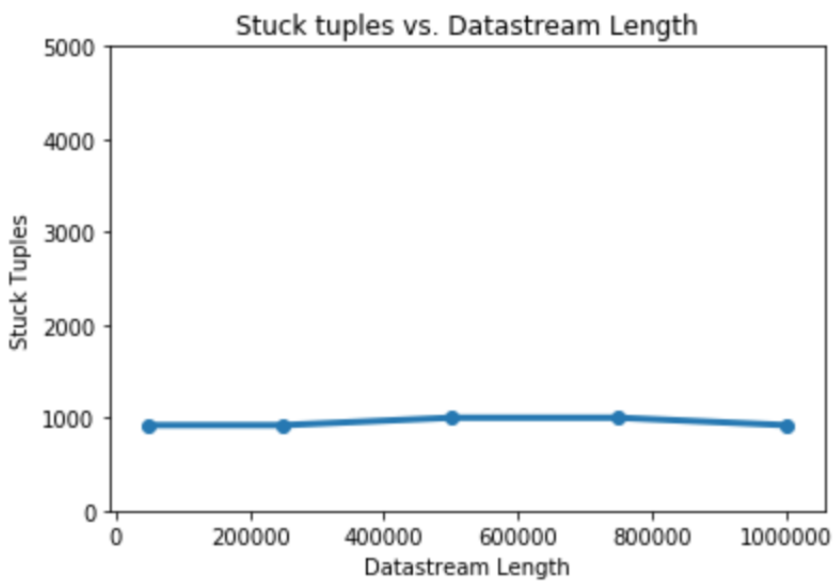

Figure 12: Number of stuck tuples for data streams of length *n = 50 000, 250 000, 500 000, 750 000 and 1 000 000*.

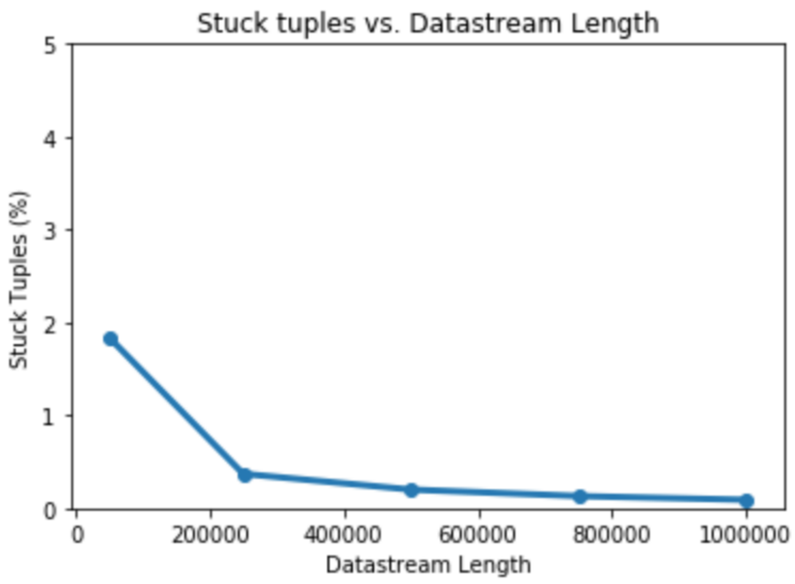

Figure 13: Stuck tuples as a percentage of total data stream length, for data streams of length *n = 50, 250, 500, 750 and 1000* $* 10^3$.

As can be seen from figure 12 and 13, it could be shown that the number of stuck tuples was constant for streams of lengths up to *n = 1 000 000*.

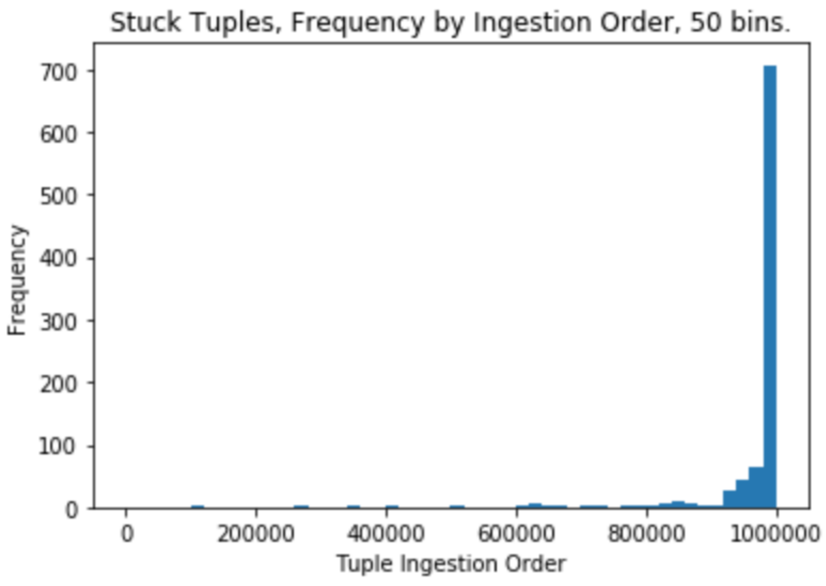

Figure 14: Stuck tuples, frequency by ingestion order for a data stream of length *n = 100000*.

As can be seen from figure 14, it could be shown that the stuck tuples were in majority a part of the last tuples according to ingestion order. (reformulate??)

C Discussion

C.1 Runtime and Tuple Delay

First of all, the runtime was expectedly linear to the data stream length, which helps us draw the conclusion that the algorithm probably not will be neither more or less efficient for infinitely long data streams. As it was to be expected, the average delay of the tuples increases with an increase of k, as shown in fig 9. As the window for each QID needs more tuples to reach K-anonymity and thereby released, the result is longer relay for the tuples already in that window. In comparison, the runtime evaluation of different values of k in fig. 7 showed that despite these delays, the algorithm in itself still finishes with constant efficiency independent of the size of k.
However, it is hard to draw conclusions regarding the tuple delay variation depending on l. In our experiments, the runtime appears to be constant for any $1 \leq l \leq 10$, where 10 is the number of different sensitive classes in our data and hence the largest possible diversity among any group of tuples. Since this is a relatively small value, for example in comparison to k, it can not be excluded that the runtime may vary for larger values of l.
As seen in fig. 8 and fig. 11, it could not be found that the method benefited from parallelization in terms of either total efficiency or tuple delay. A possible explanation could be that in the current setup, the overhead of running multiple processes is higher than the possible benefits of the parallelization, thus preventing real advantage. It is possible that parallelization will be more beneficial as the number of possible QID generalizations increase.

C.2 Stuck Tuples

The experiments could show that the number of stuck tuples remained constant for any tested data stream length. This is an expected outcome. As mentioned in section B the amount of stuck tuples has an upper bound induced by the number of unique possible combinations of the generalized quasi-identifier. Furthermore, the distribution of fig. 14 show that the tuples still in the pipeline was predominantly tuples which were inserted during the last 10% of the data stream lifetime. While expected, these results verify that the logic behind the approach of keying tuples by QID is actually working in practice. However, it must be pointed out that the upper bound of stuck tuples will become significantly bigger with a more complex QID, and that depending on data distribution a significant amount of tuples may still become stuck indefinitely in some implementations.

C.3 The keying by QID-approach

As mentioned in B, many current approaches of applying *k*-anonymity on streamed data induce an information loss that increases linearly with *k*. Through these evaluations, we could show that the approach of keying by QID with a weak, static generalization introduces a comparably small information loss, at the price of an introduced tuple delay. While the mentioned approach may reduce information loss, there still remains tuples in the pipeline considered as stuck or very delayed in the current form of the approach. As a possible solution, we would like to suggest the possibility to continuously collecting, applying *k*-anonymizing and releasing tuples which are considered as stuck according to a certain constraint. Such a constraint could be either expressed as time delay or order delay. As the percentage of stuck tuples showed to be very low, the potential information loss in total remains low.

5 Conclusion

In this research project we provided a short survey about previously conducted research in the area of privacy preserving publishing of data streams. Subsequently, we proposed a very practical algorithm, that aims at tackling the information loss with increasing k in k-Anonymity in previously conducted research, while at the same time providing a way to implement the algorithm in a parallelized way.
A wide variety of Flink's functionality was examined in order to determine their use for the anonymization task. It turned out that the windowing functionality in combination with triggers is particularly useful in order to keep track of equivalence classes. The only issue that turned out to be quite cumbersome, is to access the information of the tuples after they have been added to a window, for example to check if there already is a tuple of an individual present in the window. A successful implementation of this would probably require the use of more advanced Flink window functions than the rather general GlobalWindow.

Our empirical evaluation verifies the potential of the approach of keying by QID. While this introduces a delay trade off, our approach is robust to outliers meaning a lot can be won in terms of reduced information loss. On first sight, the analysis with respect to the parallelism does not look very promising, however, this would need further investigation in order to quantify the overhead introduced by Apache Flink through the parallelism, in order to make a reliable conclusion about whether there is a scenario when the parallelism can be fully exploited.

Hence, the paper leads to a few interesting directions for future work:
1. An imaginable scenario where the parallelism of the algorithm plays a larger role would be a case where the quasi-identifier has a very large number generalizations.
2. Another interesting aspect would be the quantification of the information loss for different QID in direct comparison to the algorithms of Zhou et al. (2009). This would also require further evaluation of the suggested methods of k-anonymizing stuck tuples after a certain delay constraint mentioned in the discussion. However this needs to be done on exactly the same data streams for comparability, which at this moment was out of the scope of the project.
3. Additionally, we did not scale the project to the largest extent possible with Apache Flink due to constraints by running the simulations on local machines. Further scalability tests on big Flink clusters with larger network buffer are still necessary, in order to completely evaluate the effects of parallelization.

References

Apache flink v1.3 documentation. (2018). `https://ci.apache.org/projects/flink/flink-docs-release-1.3/`. (Accessed: 2018-03-14)

Bayardo, R. J., & Agrawal, R. (2005). Data privacy through optimal k-anonymization. In *Proceedings of the 21st international conference on data engineering* (pp. 217–228). Washington, DC, USA: IEEE Computer Society. Retrieved from `https://doi.org/10.1109/ICDE.2005.42` doi: 10.1109/ICDE.2005.42

Byun, J.-W., Sohn, Y., Bertino, E., & Li, N. (2006). Secure anonymization for incremental datasets. In W. Jonker & M. Petković (Eds.), *Secure data management* (pp. 48–63). Berlin, Heidelberg: Springer Berlin Heidelberg.

Dheeru, D., & Karra Taniskidou, E. (2017). *UCI machine learning repository.* Retrieved from `http://archive.ics.uci.edu/ml`

Guttman, A. (1984, June). R-trees: A dynamic index structure for spatial searching. *SIGMOD Rec.*, *14*(2), 47–57. Retrieved from `http://doi.acm.org/10.1145/971697.602266` doi: 10.1145/971697.602266

Li, J., Chin Ooi, B., & Wang, W. (2008, 05). Anonymizing streaming data for privacy protection. In *Proceedings - international conference on data engineering* (p. 1367 - 1369).

Machanavajjhala, A., Gehrke, J., Kifer, D., & Venkitasubramaniam, M. (2006, April). L-diversity: privacy beyond k-anonymity. In *22nd international conference on data engineering (icde'06)* (p. 24-24). doi: 10.1109/ICDE.2006.1

Sweeney, L. (2000, January). Simple demographics often identify people uniquely. *Data Privacy, Carnegie Mellon University*.

Sweeney, L. (2002, October). K-anonymity: A model for protecting privacy. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, *10*(5), 557–570. Retrieved from `http://dx.doi.org/10.1142/S0218488502001648` doi: 10.1142/S0218488502001648

Xiao, X., & Tao, Y. (2007). M-invariance: Towards privacy preserving re-publication of dynamic datasets. In *Proceedings of the 2007 acm sigmod international conference on management of data* (pp. 689–700). New York, NY, USA: ACM. Retrieved from `http://doi.acm.org/10.1145/1247480.1247556` doi: 10.1145/1247480.1247556

Xiao, X., Yi, K., & Tao, Y. (2010). The hardness and approximation algorithms for l-diversity. In *Proceedings of the 13th international conference on extending database technology* (pp. 135–146). New York, NY, USA: ACM. Retrieved from `http://doi.acm.org/10.1145/1739041.1739060` doi: 10.1145/1739041.1739060

Zhou, B., Han, Y., Pei, J., Jiang, B., Tao, Y., & Jia, Y. (2009, March). Continuous privacy preserving publishing of data streams. In *Proceedings of the 12th international conference on extending database technology* (pp. 648–659). ACM.