Politecnico di Torino
Master's Degree in Computer Engineering

POLITECNICO
DI TORINO

Privacy enforcing framework in high dimensional data streams for real-time data markets

Supervisors: Prof. Marco MELLIA
Dott. Ing. Martino TREVISAN
Dott. Ing. Nikhil JHA

Tesi di Laurea di:

Tommaso Bacconi

Academic Year 2020-2021

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

In the era of big data, the technological ability to store and process large amounts of information has arisen previously unseen possibilities. Data has quickly become a valuable resource and consequently a new market where billions of data are bought, sold and exchanged at any given moment. The economic importance of data has issued many privacy concerns. Especially the rise of web applications has led citizens, during their online life, to be the object of incessant and ever more advanced tracking and profiling activities about their behaviors.
This loss of privacy should not be underestimated: our behavioural patterns, ideas and relationships are already being used for targeted advertising to better sell us products we don't need or persuade us which politician is worthy our vote. This type of advertisement is deteriorating the quality of online information by giving priority to sensationalist headlines and news based on fear, hate and anger that generate more traffic.
But even if the data collected is not for profiling purposes, the privacy issues do not run out. Once a user's data goes into circulation, there is no turning back. When a collection of data is shared publicly or sold between companies, user's informations can still be extracted and used with malicious intent. For example the data extracted from citizens can likely be used to decide who deserves access to a certain service, insurance, loan etc., creating new forms of discrimination.
Within this context a discussion has been raised on the need to impose restrictions on data collection and its usage. This has led to the passing of laws attempting to curb the problem, first of all the General Data Protection Regulation (GDPR) in Europe.
This is why it is important to take steps to prevent the data collected from being traced back to individuals. Unfortunately removing just the user's identifiers (name, phone ecc.) is not enough to make them anonymous as shown in some famous studies[1][2]. Indeed, an attacker looking for private data can link some user's attributes called quasi-identifiers (QIs) with external information such as other datasets, public lists or direct knowledge of the user to identify the person

and gain access to further sensible data.
Moreover, more and more applications incessantly collect large amounts of data, making it necessary to anonymise them in real time and not retrospectively.
In parallel with the need for greater privacy, many studies have been done on data anonymization. Following the path laid out by them, this thesis will focus on how to anonymize large amounts of data with zero delays.

1.1 Goals

The core of the thesis is a new method called z-anonymity that aims to anonymize a continuous streaming of large amounts of data making decisions without delays based only on the last window of time. The whole work is an effort to achieve the best perfomance in terms of privacy level, computational time and usefulness of the anonymized data. The proposed algorithm combines several techniques used in the field, particularly the generalization of numeric and non-numeric attributes as a substitute for suppression alone and also data perturbation. Moreover, the algorithm is able to adjust by itself the main parameter z as data arrives to be more flexible if volume and type of data significantly change over time. The result are tested and evaluated on different use cases to provide a context as comprehensive as possible of the potential merits and limitations.

1.2 Related Works

The focus on privacy has led mathematical models, algorithms, or other approaches to anonymizing data to become increasingly common. Among them several focus on data streams. [3] and [4] use microaggregation to achieve k-anonymity, l-diversity and differential privacy. [5][6] involve clustering data. The generalization applied to k-anonymity is presented in several studies[7][8]. Other works apply k-anonymity to a stream of data[9], also with concepts of generalization[10][11] where each quasi-identifier has a pre-defined domain generalization hierarchy. Often the streaming data are temporarly stored and published with a certain delay using sliding windows ([12][13]). However they are all based on the tuple accumulation strategy. That is, streaming tuples are postponed until they satisfy the given publication condition, i.e., reach a certain privacy level or satisfactory data utility. [14] faces this problem with a delay-free anonymization framework to preserve the privacy of electronic health data streams in real time using l-diverse counterfeit values.
My thesis is based upon the paper "z-anonymity: Zero-Delay Anonymization for Data Streams"[15] where the authors expose the functioning of the algorithm and a probabilistic model to derive k-anonymity properties from z-anonymized streams. In turn, the paper is based on another algorithm called α-mon [16] .
Regarding instead the concept of enthropy, it has been first introduced by C. E.

Shannon in [17] and has been applied to database in a similar way to what is proposed in the thesis in [18].

Chapter 2

Anonymization: a summary

There are several methods and algorithms to preserve privacy in published data, all of them require fuzzing and changing the data. The purpose is to prevent the data from being linked back to the person from whom it was extracted, causing a leakage of privacy. Unfortunately, obscuring only personal data is not enough, as it is possible to find a pattern among other data that is uniquely tied to a person. This has been proven by several studies, a famous one[1] has identified the subscribers from a dataset published by Netflix to support the Netflix Prize data mining contest. In [2] it was found that 87% of the population in the United States had reported characteristics that likely made them unique based only on three fields of information: 5-digit ZIP, gender and date of birth. These attributes that can trace back to a person identity but are not unique are called quasi-identifiers (QIs) and they are manipulated usually in three ways to lower the risk of re-identification:

- Suppression: the QI is simply not released.
- Generalization: the QI is replaced with a more general attribute, i.e. the age 23 could be released as interval [20,25].
- Perturbation: noise is added to original data to hide what is true information and what is not.

For example the perturbation is used among other methods in *differential privacy*. Differential privacy first appeared in [19] and can be considered as a set of mathematical definitions that assures a privacy standard by describing the patterns of groups within the dataset while withholding information about individuals in the dataset. The idea is that, considering a database on which are willing to extract overall statistics, the applied function is differential privacy compliant if the presence or absence of a singular row in the database doesn't change the amount of gathered information. The biggest difference here with respect to the methods we're going to see is to shift the focus from dataset anonymization to the function that performs statistics.

In contrast, generalization and suppression are widely used by *k-anonymity* which is the classical model to anonymize a static dataset and the one which we will focus on.

2.1 k-anonymity

k-anonymity (or k-anon) imposes that the quasi-identifiers of each person contained in the release cannot be distinguished from at least $k-1$ individuals whose QIs also appears in the release. In other words, if each individual is part of a larger group of k individuals with the same quasi-identifiers then no assumption can be made about who the single person is as shown in table 2.1 and table 2.2. The

Name	Postcode	Age	Gender	Has Cancer
Sara	10125	48	Female	No
Marco	10115	56	Male	Yes
Susanna	10134	42	Female	No
Sofia	10121	40	Female	Yes

Table 2.1: Orginal data from cancer tests in a certain day

Name	Postcode	Age	Gender	Has Cancer
User x	101**	[40,50]	Female	No
***	***	***	***	***
User y	101**	[40,50]	Female	Yes
User z	101**	[40,50]	Female	No

Table 2.2: Published data

published data in table 2.2 respect k-anonymity with $k = 3$: even if we knew that Sofia had a cancer screening that day, we cannot deduce what the result of the examination is. On the contrary, Tommaso, being the only male, would have been easily identifiable, so his result has been suppressed. Of course, this comes with a loss of information. In general, anonymization works on balancing the amount of privacy and the usefulness of the data, and this thesis is no exception. However, k-anonymity has some flaws, in particular it can lead to attribute disclosures when a group is homogeneous with respect to some fields. For example, holding the case in Table 2.2, an attacker cannot make assumptions about whether Sofia is user x,user y or user z but if all three users had cancer, he could still infer with certainty that Sofia has cancer. This is why k-anonymity usually comes with complementary

methods such as l-diversity or t-closeness to overcome its limits. Also, the k-anon is conceived for tabular and static data, so the dataset must be completely available at anonymization time. This does not fit with increasingly frequent applications that collect and use large amounts of data in real-time, i.e. e-commerce transaction, positions of rental bikes, website traffic, IoT stuff ecc. This is why z-anon, starting from k-anon concepts, tries to reach anonymization without any delay.

2.2 z-anonymity

The concept of z-anonymity is treated in the paper "z-anonymity: Zero-Delay Anonymization for Data Streams"[15]. The goal of the authors is to find a way to anonymize data that are:

- High dimensional
- Available as a stream

The data stream is intended as a continuos observation of users attributes. An observation is defined as (t, u, a), which means that at time t a user u exposed an attribute a. Similarly to k-anon, when a new attribute arrives it is released only if at least $z - 1$ individuals have presented the same attribute, with the difference that it does not consider the entire data history but only the past window of time Δt. Figure 2.1 shows the data stream along with the z-anon mechanism.

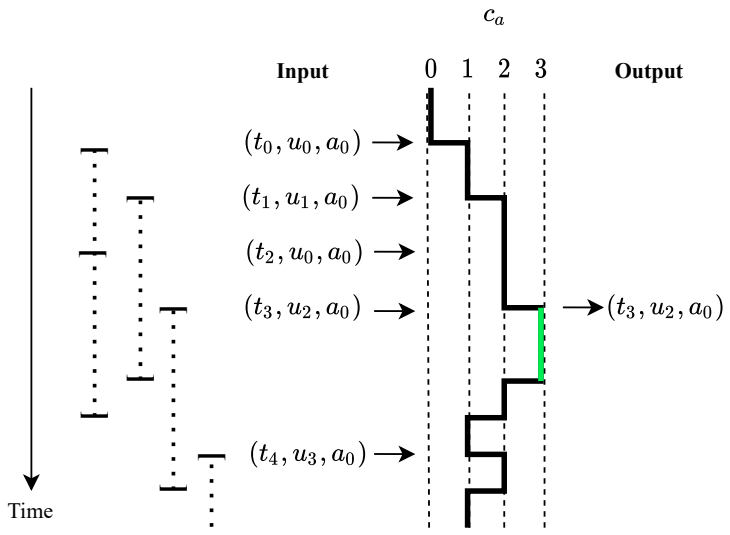


Figure 2.1: A graphical example of z-anon extracted from [15].

As explained in [15]:

> Assume $z = 3$. At time t_0 user u_0 is the first to expose the attribute-value a_0. The attribute a_0 is z-private at time t_0, hence it shall be obfuscated. Still, the information that u_0 exposed the attribute a_0 is kept in memory for a time equal to Δt. At time t_1, user u_1 also exposes a_1. Since the number of observations in Δt is still smaller than 3, the observation is not released. At time t_2 user u_0 re-expose again a_0 – extending the lifetime of the observation, but not changing the number of unique users having exposed a_0. At time t_3, user u_2 exposes a_0 – making the total users in the past Δt equal to 3. Thus the attribute-value pair a_0 is not z-private at time t_3 and the observation (t_3 , u_2 ,a_0) can be be released. At time $t_1 + \Delta t$ the attribute a_0 related to user u_1 expires, hence the total user count decreases back to 2. The same happens when u_0 observation expires (at $t_2 + \Delta t$), so that when u_3 exposes a_0 at t_4 the observation cannot be released.

2.2.1 Implementation

The data structures of z-anon are made of:

- An hash table $\mathcal{H}(a)$: with the set of user for a.
- A Least Recently Used list LRU(a) of tuples (t, u)
- A counter c_a

The hash table $\mathcal{H}$ contains all the attributes and users that appeared in the last time window Δt; once an observation (t, u, a) arrives, the attribute a is stored in $\mathcal{H}$ (if not present) and the user u is added to the set $\mathcal{H}(a)$ as shown in Algorithm 1 (lines 2-3). The counter c_a keeps track of how many user are present in the $\mathcal{H}(a)$ so it's incremented every time a new user is added. If u is already present in the set then $\mathcal{H}$ does not change, in this case only LRU(a) is updated. LRU(a) is a list ordered by timestamp of users that have exposed a; when a new user u arrives, the tuple (t, u) is added at the end of the list, if it's already present then it is moved at the end of the list and its t updated with the new timestamp (lines 12-13).
LRU(a) is needed to speed up the the evict process: infact the list of tuples (t, u) is scanned at any moment from the beginning to remove each tuple (t', u') where $t' < t - \Delta t$. Consequently, if a user is removed, c_a is decreased and $\mathcal{H}(a)$ updated (lines 15-23).
These data structures ultimately serve to make the decision to suppress or output a new observation (t, u, a) when it arrives, simply if $c_a >= z$ then (t, u, a) is released (lines 25-27).

Algorithm 1 Pseudo code of z-anon

Input: (t, u, a)
if $a \notin \mathcal{H}$ **then**
    $\mathcal{H}(a) = \{u\}$
    $c_a = 1$
    $\text{LRU}(a) = [(t, u)]$
else
    if $u \notin \mathcal{H}(a)$ **then**
        $\mathcal{H}(a) \leftarrow u$
        $c_a = c_a + 1$
        $\text{LRU}(a) \leftarrow (t, u)$
    else
        $(t', u) \leftarrow (t, u)$
        Move (t, u) at the end of $\text{LRU}(a)$
    end if
    for $((t', u') = \text{first}(\text{LRU}(a)); t' < t - \Delta t; (t', u') = next)$ **do**
        remove (t', u') from $\text{LRU}(a)$
        remove u from $\mathcal{H}(a)$
        $c_a = c_a - 1$
        if $c_a = 0$ **then**
            remove a from $\mathcal{H}$
            remove a from LRU
        end if
    end for
end if
if $c_a >= z$ **then**
    OUTPUT (t, u, a)
end if

2.2.2 z-anon limitations and goals of the thesis

z and Δt are parameters that can be tuned, so a large z and a small Δt results in higher anonymization, viceversa with a small z and a large Δt also rarer values are released.
This is because z always remains the same regardless of data traffic. This is not very efficient: the same z can anonymize everything at a time of low data traffic, as happens at night, and anonymize very little at peak times with large amounts of data. Moreover, attributes below z are simply blurred, while it's possible to preserve the same level of privacy also releasing some attributes.
Solutions to these problems proposed in the thesis are:

- Achieve greater utility of the data avoiding suppression and instead replacing the attributes, whenever it's possible, with a more generic value that is not z-private
- Allow the algorithm to vary the parameter z by calculating it as data arrives.
- Adding data instead of subtracting it: fake attributes are output to disrupt the data and not making the true private information recognizable.

Chapter 3

Datasets

3.1 Overview

To evaluate the algorithm we tested the results on three use cases. We simulated a stream of data from three different datasets with very diverses characteristics in order to have a better understanding of its impact in more situation. To get started, the three datasets are different in terms of the data field:

- Transactions from an ecommerce store[1]
  - Each row in the file represents an event of a user and it's related to a product.

	unix_time	user_id	category_code	product_id
0	1569880800	554748717	appliances.environment.water_heater	3900821
1	1569880801	519107250	furniture.living_room.sofa	17200506
2	1569880801	550050854	computers.notebook	1307067
3	1569880804	535871217	electronics.smartphone	1004237
4	1569880805	512742880	computers.desktop	1480613
5	1569880810	520571932	apparel.shoes.keds	28719074

Table 3.1: Ecommerce store head

[1] https://www.kaggle.com/mkechinov/ecommerce-behavior-data-from-multi-category-store

- Network traffic
  - Collected monitoring traffic of a real network. For each connection is extrapolated the top and second level domain of the requested server as attribute.

	unix_time	ip_address	domain
0	1494809696036	246.111.10.51	uol.com.br
1	1494809696250	180.105.185.9	mega.co.nz
2	1494809696519	180.105.185.9	mega.co.nz
3	1494809697295	246.111.10.51	uol.com.br
4	1494809699743	246.25.221.89	wasabii.com.tw
5	1494809701606	180.102.208.78	mega.co.nz

Table 3.2: Traffic domains head

- Positions of private vehicles
  - Every record has an address, a latitude and a longitude.

	unix_time	device_id	latitude	longitude	address
0	1545100000	2828722	44.9107	8.294838	ITALIA*PIEMONTE*AT*ASTI*A21
1	1545100003	4180363	45.083263	7.697360	ITALIA*PIEMONTE*TO*TORINO*CORSO...
2	1545100006	3338380	44.914364	7.846230	ITALIA*PIEMONTE*TO*POIRINO*SP029
3	1545100006	4209757	43.910472	10.943764	ITALIA*TOSCANA*PT*PISTOIA*A11
4	1545100009	4304898	45.007003	7.620442	ITALIA*PIEMONTE*TO*NICHELINO*A55
5	1545100009	4234898	45.018253	7.482941	ITALIA*PIEMONTE*TO*BRUINO*VIA...

Table 3.3: Vehicle positions head

3.2 Categories

Since we'll be dealing with generalization, we made sure to derive datasets where it was possible to fit attributes into categories. Most datasets that are collected these days are already well categorized or matched to tags so it's not hard to get a generalization hierarchy for each attribute.
For example, the ecommerce transaction dataset in Table has a *category_code* field with a hierarchy of categories divided by dots, with the rightmost category being the most specific and the leftmost the most universal.
Website domains being all second level domains have no categories (except *.com*,

	Ecommerce	Domains	Positions
Number of tuples	3,000,000	11,000,000	1,798,399
Distinct attributes	121,903	27,482	77,028
1-level categories	453	83	22,667
2-level categories	31	NaN	1,308
3-level categories	12	NaN	77
4-level categories	2	NaN	23
5-level categories	NaN	NaN	6

Table 3.4: All datasets - Number of categories

.org which we have excluded to use). We used the cisco umbrella api to get them[2]. In this way a site like *facebook.com* is associated to the *social_network* category.
The position dataset has a natural category structure consisting of *country-region-city-street* embedded in the address. Also, having geographic coordinates available, we used a more elastic form of generalization based on distances: we have subdivided the considered territory in cells of a certain largeness in meters obtaining a grid.
Each geographical point falls into a certain cell, which thus becomes its own category. We created five grids, each with cells of different sizes in order to have multiple levels of generalization. This makes it easy to vary cell precision and generalization levels with a single list passed as a parameter.

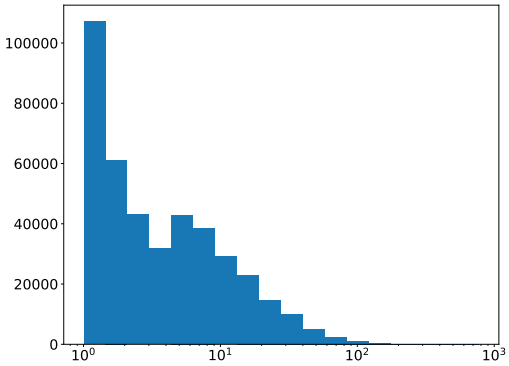
3.3 Users distribution

One of the features that most impacts the effectiveness of the algorithm on a dataset is the user distribution. As we will see in fact if a user exposes several attributes it is easier that he is not anonymized because it is possible to find a pattern unique to him. Vice versa, a dataset composed of many inactive users will be highly anonymized with little effort. For example, as summarized in Table 3.5 and represented in Figure 3.1, the ecommerce dataset has a lot of occasional users that tend to appear and not come back. Instead, the dataset on network traffic is completely the opposite: although considering many more tuples, the monitored users are always the same and each of them appears in average more than 1,400 times. The locations dataset represents a middle ground by having users occurencies an order of magnitude larger than the ecommerce.

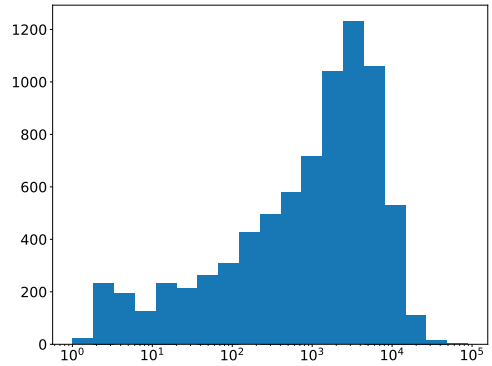
[2] `https://docs.umbrella.com/investigate-api/reference#get_domains-categorization-domain`

	Ecommerce	Domains	Positions
Number of tuples	3,000,000	11,000,000	1,798,399
Distinct users	410,609	7,809	23,500
Avg. samples per user	7.3	1,408.6	76.5

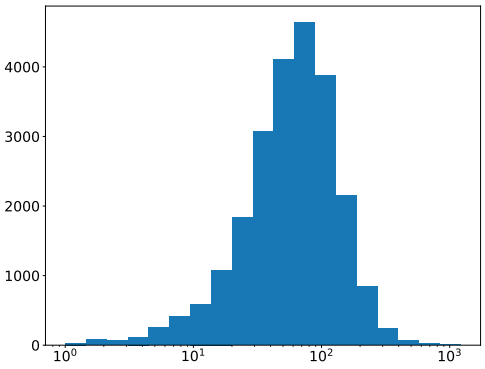
Table 3.5: All datasets - Users



(a) Ecommerce



(b) Domains



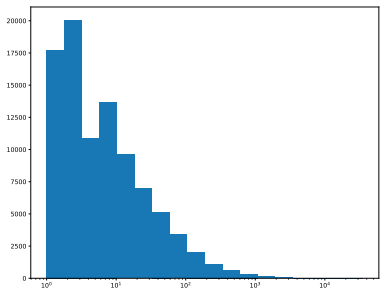
(c) Positions

Figure 3.1: Samples per user

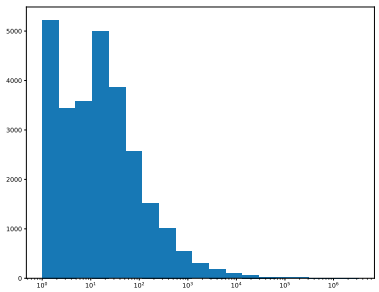
3.4 Attributes distribution

Since rarer attributes are anonymized while more frequent attributes are allowed to pass, it is also important to consider whether the datasets consist of a few recurring

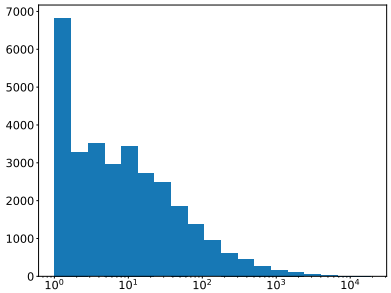
attributes or many occasional ones. Obviously attributes, being very specific, are also often rare. As we can see in Figure 3.2 the number of occurrences for attributes that appear few times is much higher. From the point of view of the usefulness of the data this is not good because only the very recurring attributes, which are those in the tails of the graphs, are likely to be released. The generalization thus allows not to anonymize all this data but to release a more generic attribute. Obviously, the more generic an attribute is, more likely it is to have been exposed previously. For example, Figure 3.3 shows the distribution considering the attributes as they are proposed in the data (Fig. 3.3a), then removing the attribute and keeping the most specific category (Fig. 3.3b), and finally also removing the most specific category and keeping a second (Fig. 3.3c) and a third (Fig. 3.5d) level of generalisation . It is possible to see that by generalising more and more rare attributes disappear in favour of frequent but less attributes. With a 3-level generalization the ecommerce store encloses the whole dataset with only 11 attributes.



(a) Ecommerce



(b) Domains

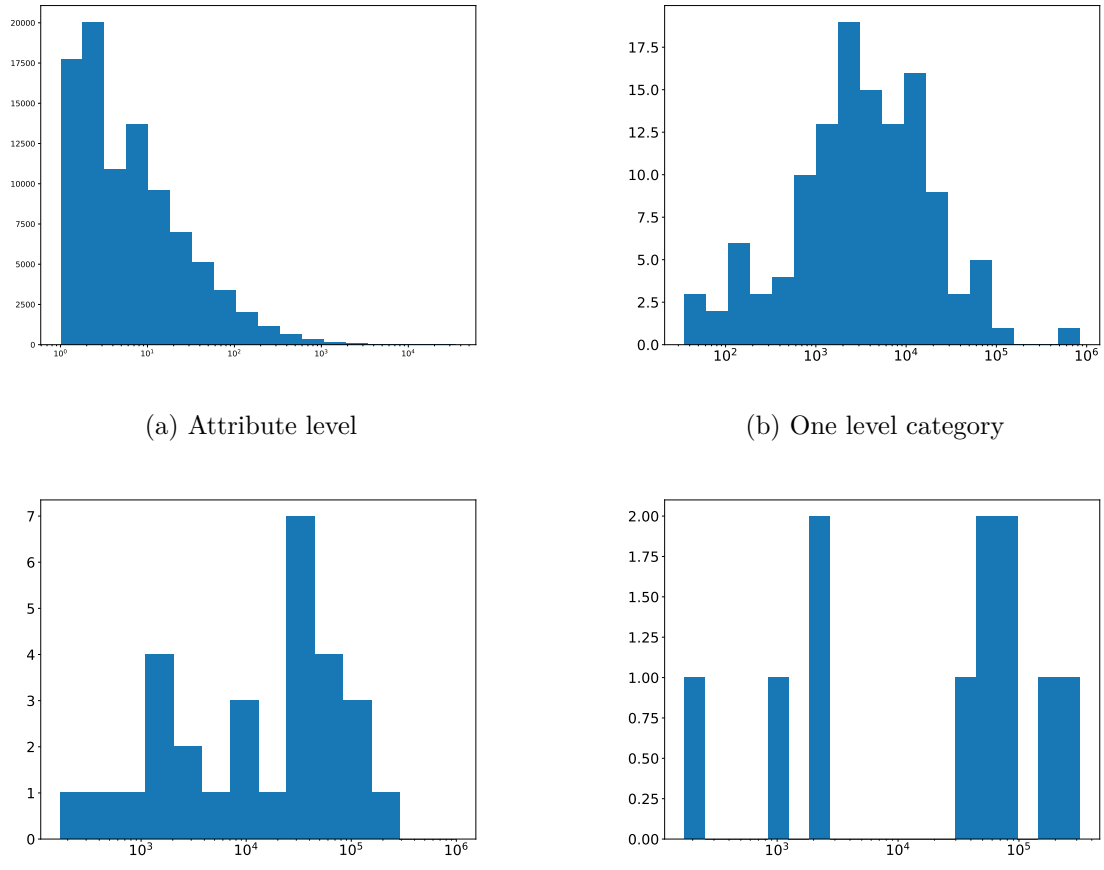


(c) Positions

Figure 3.2: Samples per attribute

This is different for domains, they have only one level of generalisation with a distribution similar to that of non-generalised attributes (Fig. 3.4).

The positions dataset represents again a middle ground, in fact it has the first level of generalisation like that of the domains (Figure 3.5b) but at the same time the more general categories are few and always the same(Fig. 3.5c and Fig. 3.5d). This is explained by the fact that the surveys are almost all in Piedmont and in a few adjacent regions/states, so the streets are very specific but the regions are the same recurring ones.
Looking at the characteristics of the ecommerce and positions datasets, we expect to always find a category that allows us not to suppress attributes altogether, whereas we expect to have to anonymize more in the case of domains.



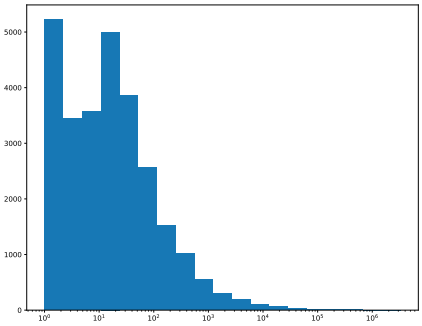
(a) Attribute level

(b) One level category

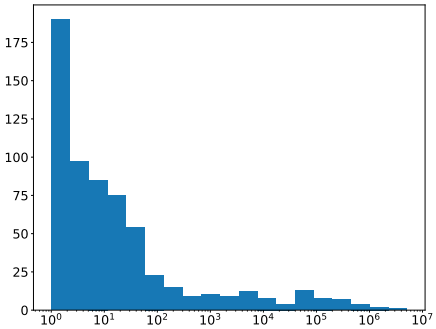
(c) Two level category

(d) Three level category

Figure 3.3: Samples per attribute, Ecommerce dataset

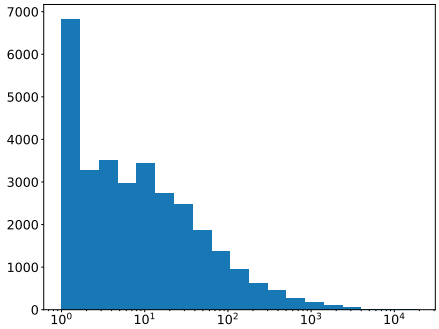


(a) Attribute level

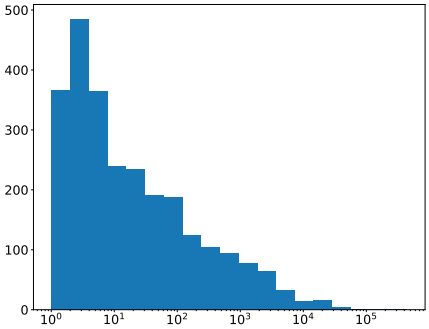


(b) One level category

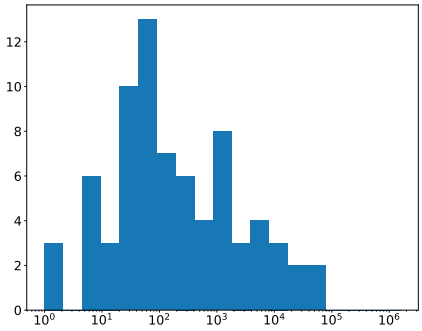
Figure 3.4: Samples per attribute, Domains dataset



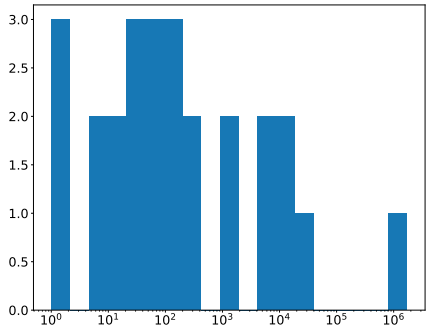
(a) Attribute level



(b) One level category



(c) Two level category



(d) Three level category

Figure 3.5: Samples per attribute, Positions dataset

Chapter 4

z-anonymity with generalization

4.1 The algorithm

So let's consider a stream of observations arriving in the form of (t, u, s) where s is a string formed by the concatenation of the various levels of generalisation.

Definition 4.1.1 *s = large_category*(...)*specific_category*attribute*

The idea is to save in the data structure also the categories with their respective counters. This way, when a new string arrives, it is possible to know the counter of each generalisation level and thus what can be output.
To do this, however, we had to rework the entire structure to make the algorithm efficient even with four or five times as many attributes in memory.
In particular, it is very important to be able to access the data in $O(1)$ time. With such a long list of attributes and the amount of traffic for which the algorithm is designed, any different time to access data would kill its usability.
To do this we incorporated LRU and hash table into a single data structure, an hash table of sorted dictionaries $\mathcal{D}_n$ that guarantee access in $O(1)$ time but that is able to keep memory of the insertion order and to remove and reinsert any key-value pairs in $O(1)$ time. How does an ordered dictionary work? unlike a hash table it stores keys in a doubly-linked list that makes deleting a key constant $O(1)$ time and then uses a second, hidden, dict to find the doubly-linked list node belonging to the key in $O(1)$ time.

Definition 4.1.2 $\mathcal{D}(u, t)$ *= ordered dictionary: an hash table that also keeps track of the order in which keys are added, u is the key and t is the value.*

In this way each attribute of the hash table is composed of a dictionary sorted by user arrival. This makes the process of expelling old users very fast. In addition,

not all users beyond the time window are ejected but only those who have exposed the attribute in question and thus affect the counter. All these measures take up slightly more memory but really speed up the process.
The result is that the algorithm is able to handle the larger amount of data more efficiently. So when a choice has to be made about what to output, the counter of the most specific attribute is checked to see if it's above the z threshold, if it is not, the next most generic attribute is checked to see if it is above the threshold, etc. As long as one attribute do meet the condition and is output or no attribute is found and then nothing is released (line 23 - 27 of Algorithm 2).

Algorithm 2 Pseudo code of z-anon with generalization

1: **Input**: (t, u, s)
2: **for** a in split(s) **do** //Add every category or attribute of s
3: **if** $a \notin \mathcal{H}$ **then**
4: $\mathcal{H}(a) =$ new $\mathcal{D}(u, t)$ //Create a dictionary for every new attribute
5: $c_a = 1$
6: **else**
7: **if** $u \notin \mathcal{H}(a)$ **then**
8: $\mathcal{H}(a) \leftarrow$ new $\mathcal{D}(u, t)$
9: $c_a = c_a + 1$
10: **else**
11: $\mathcal{D}(u, t')$ update t
12: Move $\mathcal{D}$ at the end of H(a)
13: **end if**
14: **end if**
15: **for** ($\mathcal{D}' =$ first($\mathcal{H}(a)$); $\mathcal{D}'(u) < t - \Delta t$; $\mathcal{D}'$=next) **do**
16: remove $\mathcal{D}'$ from $\mathcal{H}(a)$
17: $c_a = c_a - 1$
18: **if** $c_a = 0$ **then**
19: remove a from $\mathcal{H}$
20: **end if**
21: **end for**
22: **end for**
23: **for** a in reverse(split(s)) **do** // Find the most specific attribute not z-private.
24: **if** $c_a >= z$ **then**
25: OUTPUT (t, u, a)
26: break
27: **end if**
28: **end for**

4.2 First observations

First remarks to be made are about the amount of information that generalisation allows to be preserved. In Table 4.1 we can see the amount of tuples that are released based on their level of generalization, taking the ecommerce dataset as an example. With standard z-anonymity only tuples with 0-generalization would have been released, so with $z = 20$, 73% of the data would have been anonymized. Instead, in this way we manage to suppress only 6.4% releasing in most cases an attribute just a one level of generalization higher. Even if we have to consider different z values to see the impact on the other datasets (Tables 4.2 and 4.3) the basic point stands: the anonymized tuples are always few and in their place is output a more or less specific category .

	$z = 1$	$z = 3$	$z = 5$	$z = 10$	$z = 20$
0-generalization	100%	64.2%	51.5%	37.5%	27%
1-generalization	0%	35.2%	47.1%	58.6%	64%
2-generalization	0%	0.1%	0.3%	0.8%	2.3%
3-generalization	0%	0%	0%	0.1%	0.3%
Anonymized	**0%**	**0.5%**	**1.1%**	**3%**	**6.4%**

Table 4.1: Ecommerce - % of output tuple per level of generalization and z. Δt=1 hour.

	$z = 1$	$z = 5$	$z = 20$	$z = 100$	$z = 900$
Attribute	100%	88.9%	82%	72.1%	46.7%
Category	0%	11%	17.2%	23.2%	28.6%
Anonymized	**0%**	**0.1%**	**0.8%**	**4.7%**	**24.7%**

Table 4.2: Domains - % of output tuple per level of generalization and z. Δt=1 hour.

	$z = 1$	$z = 2$	$z = 5$	$z = 50$	$z = 150$
Street	100%	88.3%	70.6%	19.8%	5.4%
City	0%	10.1%	24.8%	51.7%	48.7%
Province	0%	1.3%	3.7%	22.5%	34%
Region	0%	0.2%	0.6%	3.2%	8%
Country	0%	0.1%	0.2%	2.5%	3.5%
Anonymized	**0%**	**0%**	**0.1%**	**0.3%**	**0.4%**

Table 4.3: Positions - % of output tuple per level of generalization and z. Δt=1 hour.

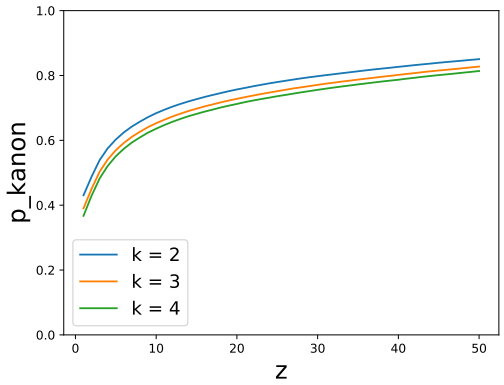
4.3 Evaluate with k-anon

One way to evaluate the impact of the algorithm is to evaluate the relationship with k-anonymity. Obviously z-anon cannot guarantee k-anonymity (and this is not the goal) since it does not consider combinations of attributes and users from the entire dataset but just windows of time. However, makes it possible to calculate the probability of the release data to respect k-anon (Definition 4.3.1) in case an attacker observes the traffic for multiple time windows and therefore adjusts the parameters accordingly.

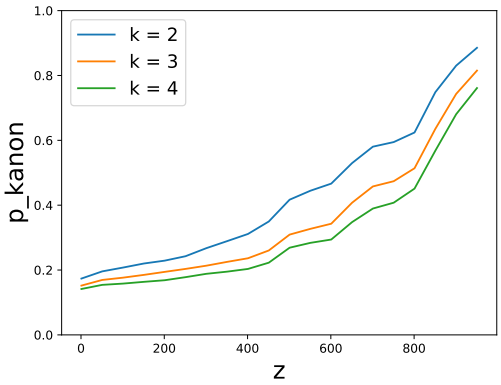
Definition 4.3.1 p_{k-anon} = *Probability of satisfying k-anonymity.*

All of these evaluations are done retrospectively just to assess how well the algorithm worked and with what parameters. In the next chapter we will discuss more extensively the tuning of z.
Impact of z on p_{k-anon} is shown in Figure 4.1. Obviously increasing z will result in more suppressed data and higher probability of k-anonymity, but the choice of z depends very much on the type of traffic: data extracted from the ecommerce site, having many users but less recurring, maintains a very high p_{k-anon} with low values of z (Fig. 4.1a), on the contrary with the domains data, for the characteristics of the dataset explained in the previous chapter, is difficult to guarantee a decent p_{k-anon} at least until about $z = 800$ (Fig. 4.1b).
Ultimately, the positions dataset linearly improves p_{k-anon} as z increases with discreet results above 150 (Fig. 4.1c).

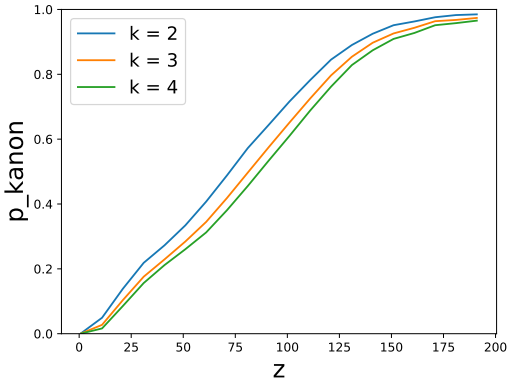
While the effect of generalization on data utility is prominent, its impact on the probability of achieving k-anonymity is less obvious since it releases many more attributes and users. Also in this case the results vary a lot depending on the case taken into consideration. The data curve where generalization is considered satisfies p_{k-anon} more slowly than the other one with a very pronounced difference in the domains case (Fig. 4.2b) and almost no difference in the ecommerce case (Fig. 4.2a).
This means that, using generalization, we will need higher z to keep the same level of privacy. This is an acceptable compromise considering, as shown before, that z-anon with generalization can output a lot of useful data even with higher z.



(a) Ecommerce

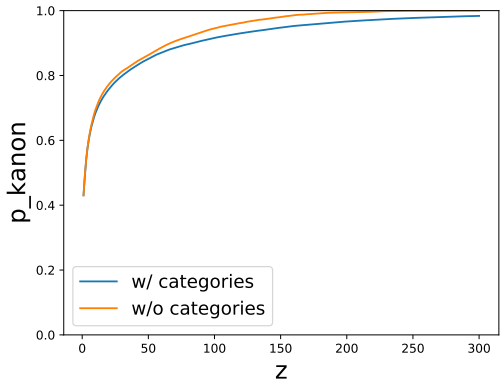


(b) Domains

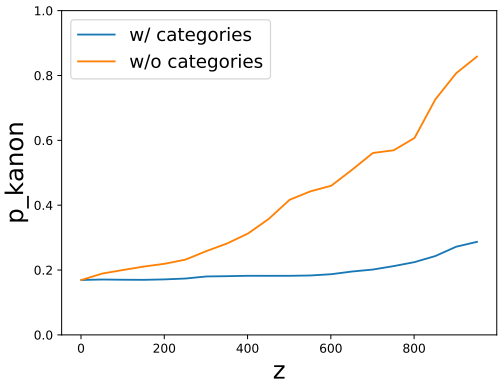


(c) Positions

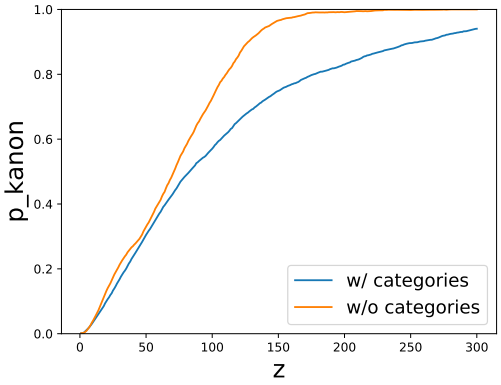
Figure 4.1: Impact of z on k-anonymity - no generalization



(a) Ecommerce



(b) Domains



(c) Positions

Figure 4.2: Impact of z on k-anonymity - with and without generalization

4.4 Evaluate over time

Being applied on a stream, z-anonymity inevitably is conditioned by the amount of data, attributes or users that passed in the last window of time. That amount, regardless of dataset, varies with a certain periodicity within day, week etc.. So it's useful to represent how data is released over time and also how it fulfills p_{k-anon} during the simulation.
In order to calculate that we need to keep track of the tuples that have been output. To do this we used a deque which is a double-ended queue, i.e., it allows items to be added and removed from both ends (Fig. 4.3). This way when a tuple is output, it is added to the end of the deque and with each new tuple the deque is cleaned by removing the too old tuples from the beginning.
This deque contains the elements that have been output in the last window of time, calculating p_{k-anon} on this data gives us a measure of how the algorithm is working.

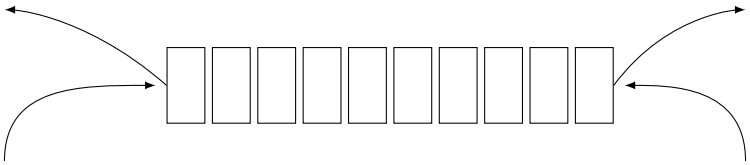
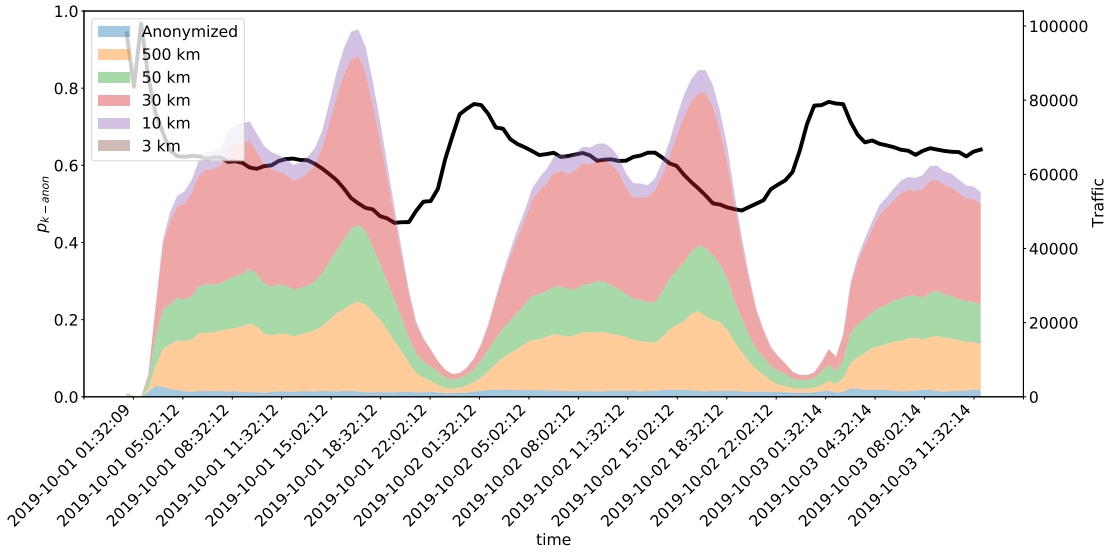
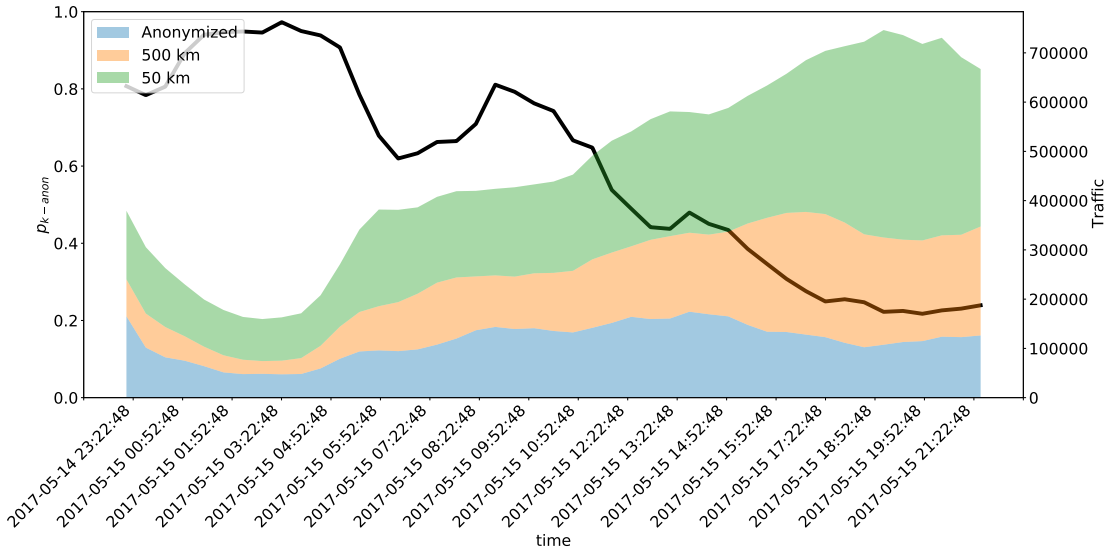


Figure 4.3: Deque - both items at the ends can be managed

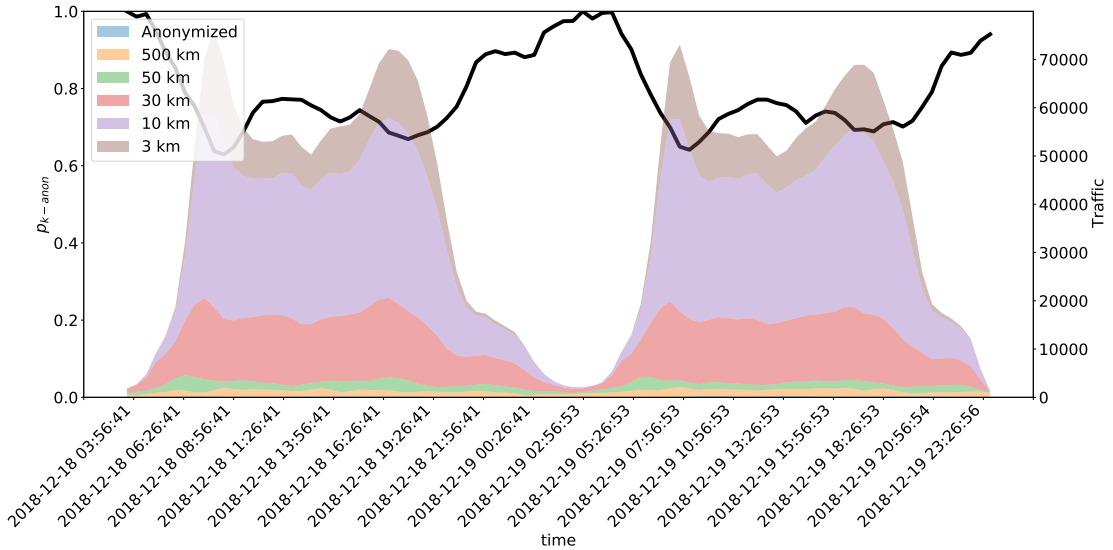
Let's start with ecommerce data. First, we have to choose proper parameters, from Figure 4.1a we can deduce that $z = 10$ should be enough and, since the dataset covers a period of time of two days and a half, a window of time of one hour sounds good. The result is plotted in Figure 4.4a that shows how the traffic volume varies and how it is released: in the blue band are the tuples that were suppressed because they had no attribute or category that exceeded z, instead the other colored bands represent the order of detail whereby the data is output. The level of detail is how many attributes the string s is composed of. So 4-detail represents a very specific attribute, while 1-detail is the most generic possible.
In this case p_{k-anon} oscillates around 0.6 depending on the traffic volume, as it does in a similar way the positions dataset even if it requires a little higher z (Fig. 4.4c). Instead, as we have seen, the domain dataset needs a much higher z, so using for example $z = 900$ we see that, not being able to generalize further, so much is anonymized and in particular we see how as traffic increases p_{k-anon} collapses. (4.4b).



(a) Ecommerce - $z = 10, \Delta t = 1hour$



(b) Domains - $z = 900, \Delta t = 1hour$



(c) Positions - $z = 60, \Delta t = 1hour$

Figure 4.4: p_{k-anon}

Chapter 5

Self-tuning z

As mentioned, so far we have considered a constant z throughout the simulated time. So the amount or type of data is not considered. A more useful approach for a streaming algorithm, that does not know retrospectively the traffic pattern, would be to vary z as needed.
Obviously, the calculation can only be done on the data that are available, i.e., those from the last time window.
How then to decide which value of z is most suitable at any moment? we have considered two algorithms:

- Incremental approach
- Binary search

In both cases the assumption is to obtain a z that satisfies a certain p_{k-anon}. So z adjusts itself according to what goal is given to it in the input parameters.
These new parameters are:

- k_goal: whether the data should be 2-anonymized, 3-anonymized etc.
- p_{k-anon}_goal: the probability of having a k-anonymized data.
- Update rate: how often must z be calculated.
- (Only for incremental) z_0: starting z.

5.1 Incremental approach

The incremental approach is the simpler of the two. Algorithm 3 shows how it works. After setting the goals, at each update z is slightly adjusted according to whether it is below or above the target (line 8 - 12).
This type of approach is a step up from a fixed z as depicted in figure 5.1 (let's consider $\Delta t = 1$ hour, $k\_goal = 2$ and $p_{k-anon}\_goal = 0.8$ for all use cases). Here z, represented by the red line, anonymizes according to the target in a dynamic way. From the figure we can see some limitations of this method: z follows the trend of p_{k-anon} lowering and raising itself depending on the data, the p_{k-anon} curve increases when traffic drops because z is naturally lagging by calculating the probability on past data and not predicting future ones. The problem can in part be contained by increasing the frequency with which z like in figure 5.2 where it's update every 5 minutes. Same speech for positions: z does a good job of keeping p_{k-anon} almost constant above 0.8 (Figure 5.4). The problem arises with a dataset such as domains, in fact in this case even with a high z_0 and a 5 minute $update\_rate$ z remains too low to satisfy the probability of reaching k-anon.
To summarize, by changing z incrementally we already get results. Moreover, the algorithm - very simple - can be improved by changing the increment according to the proximity or distance from the target. In any case we need a suitable z_0 and this means to know the data retrospectively. The next method uses binary search to overcome this limitation and get a z dependent only on the goal.

Algorithm 3 Algorithm with incremental tuning

```
1: z = z_0
2: Initialize k_goal
3: Initialize p_{k-anon}_goal
4: Set update_rate
5: Input: (t, u, s)
6: if t − last_update == update_rate then
7:     Compute p_{k-anon} on H
8:     if p_{k-anon} < p_{k-anon}_goal then
9:         z = z + 1
10:    else
11:        if p_{k-anon} > p_{k-anon}_goal then
12:            z = z − 1
13:        end if
14:    end if
15: end if
16: ... Algorithm 2
```

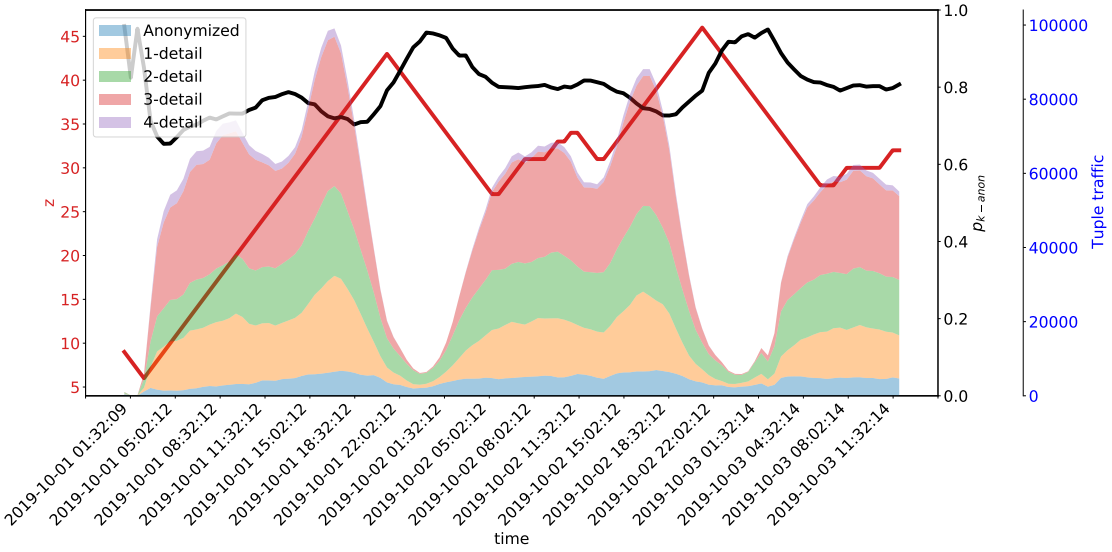


Figure 5.1: Ecommerce - $z_0 = 10$, $update\_rate = 30$ min

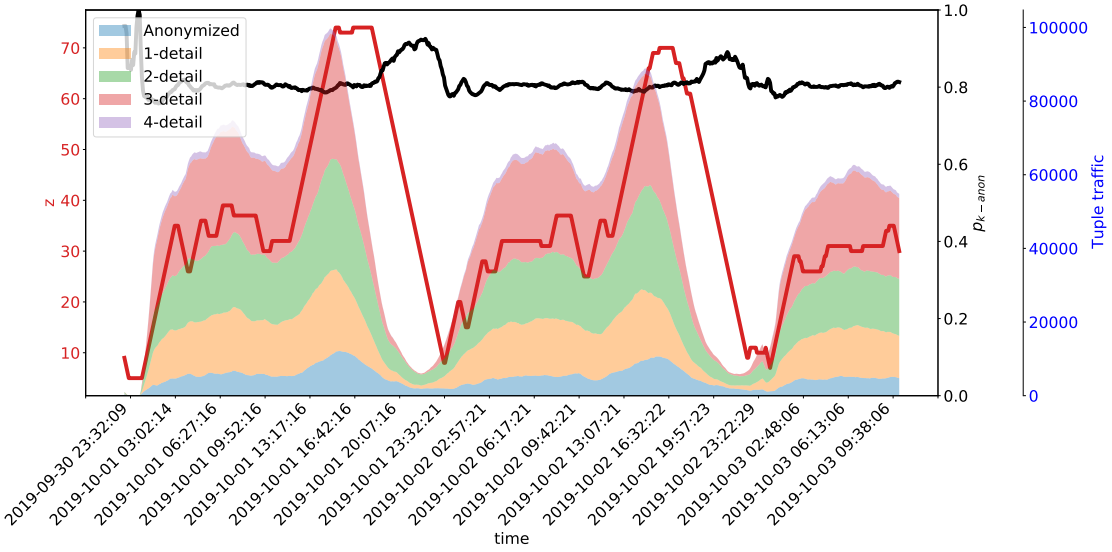


Figure 5.2: Ecommerce - $z_0 = 10$, $update\_rate = 5$

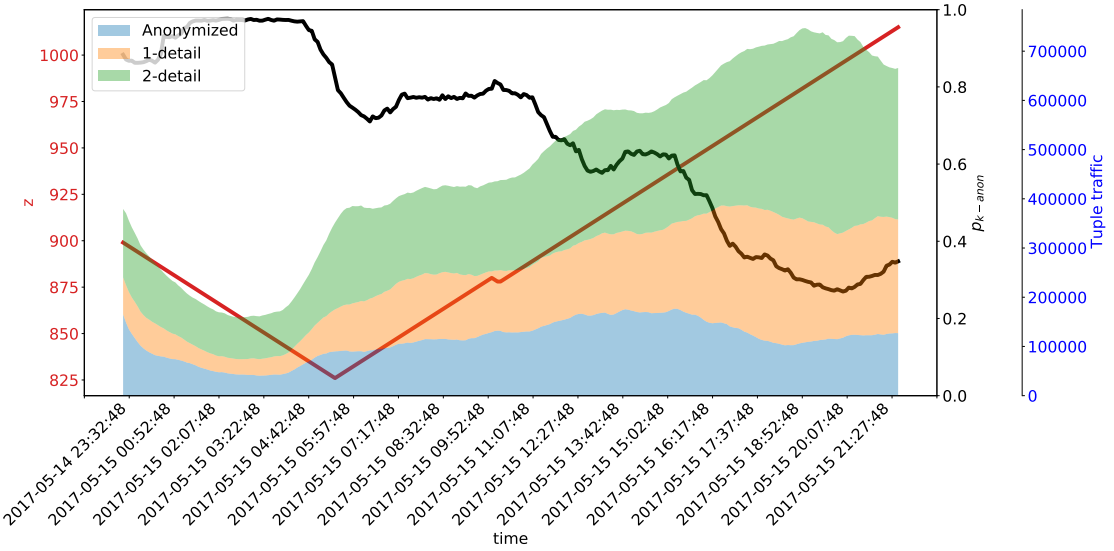


Figure 5.3: Domains - $z_0 = 900$, $update\_rate = 5$ min

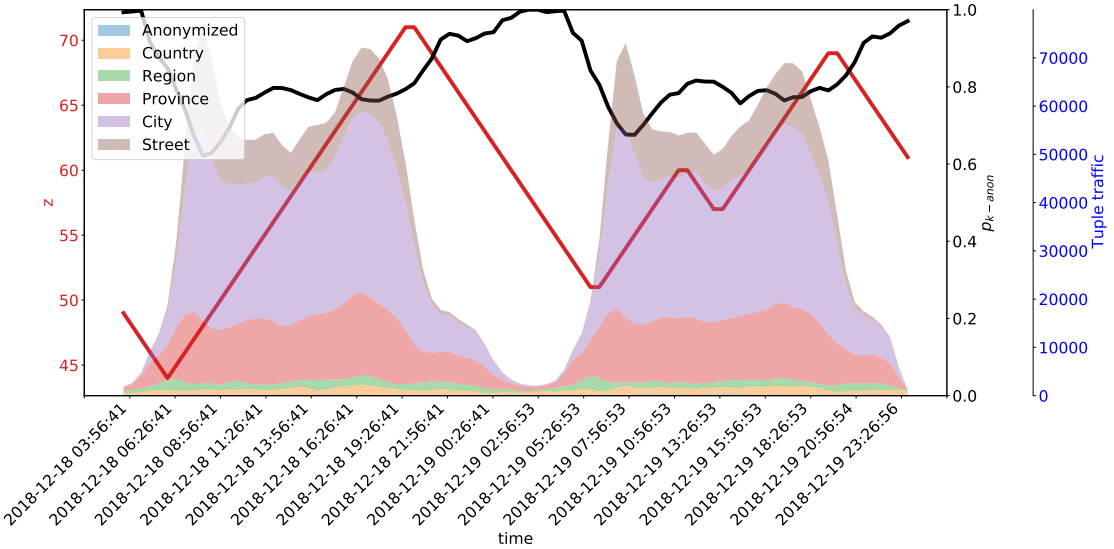


Figure 5.4: Positions -$\Delta t = 1$ hour, $z_0 = 50$, $update\_rate = 30$

5.2 Binary search

Instead of calibrating an initial z and going by trial and error, a more suitable approach is to calculate at each update exactly which z satisfies p_{k-anon} in the data structure. So we took a range of z wide enough such that we can assume that there exists a z within the range for which our data structure will have the desired p_{k-anon}. Formally:

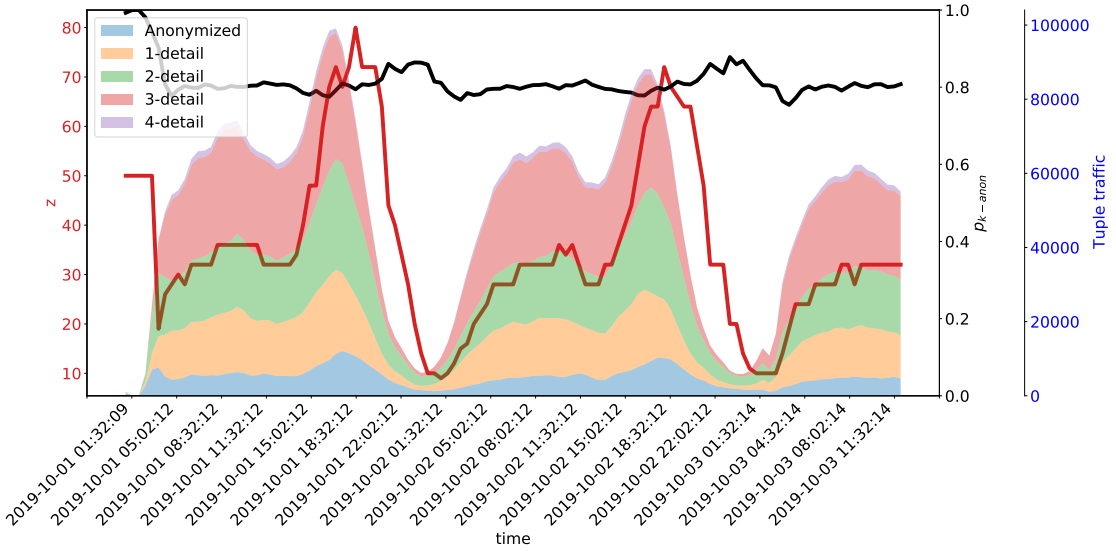
$$\exists z \; in \; [1, N] \ni p_{k-anon} \; of \; \mathcal{H} == p_{k-anon\_goal}$$

It thus becomes a search algorithm and the fastest way to find the position of a value in an ordered array is binary search. The algorithm 4 shows the search loop that basically splits the range of possible z at each iteration (lines 8 - 19).

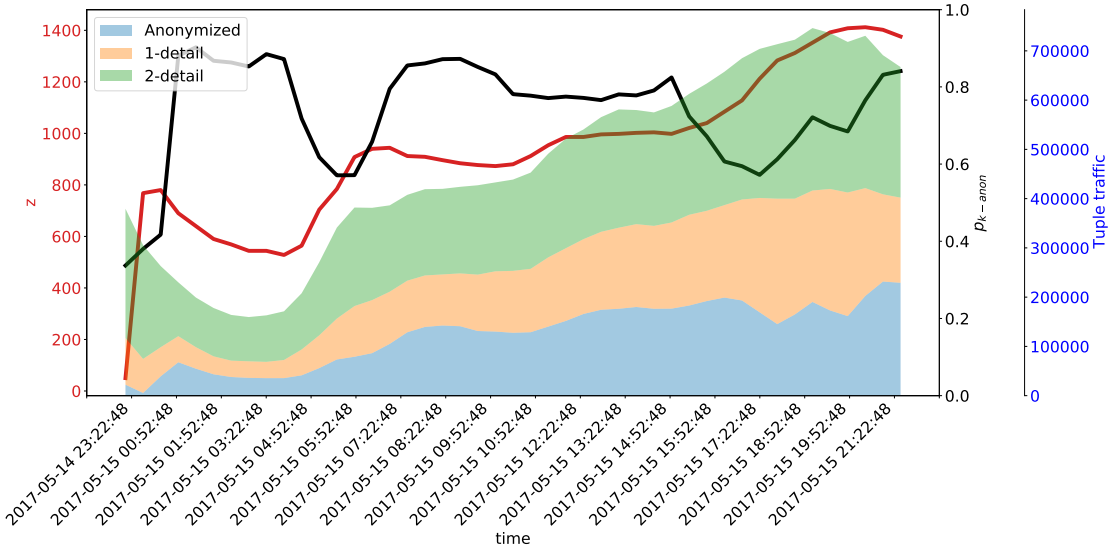
Algorithm 4 Algorithm with binary search

```
1: Initialize k_goal
2: Initialize p_{k-anon}_goal
3: Set update_rate
4: lowerBound = 1
5: upperBound = N
6: Input: (t, u, s)
7: if t − last_update == update_rate then
8:     while z not found do
9:         z_mid = (lowerBound + upperBound) ÷ 2
10:        Compute p_{k-anon} on H using z_mid
11:        if p_{k-anon} == p_{k-anon}_goal then
12:            Exit with z = z_mid
13:        end if
14:        if p_{k-anon} > p_{k-anon}_goal then
15:            upperBound = z_mid − 1
16:        else
17:            lowerBound = z_mid + 1
18:        end if
19:    end while
20: end if
21: ... Algorithm 2
```

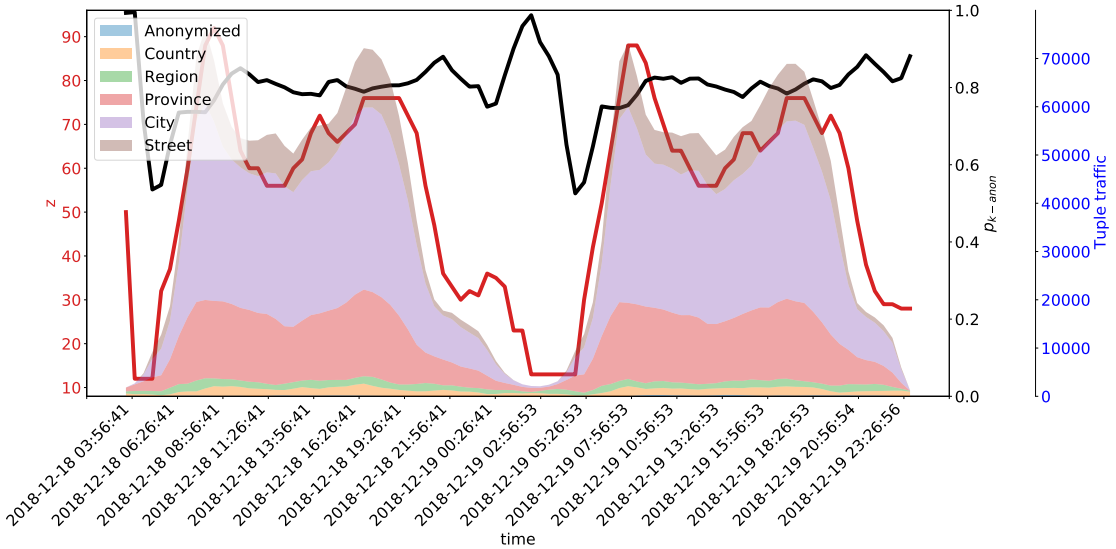
So the search time is best-case scenario $O(1)$ if the right z is exactly in the range middle and worst-case scenario $O(logN)$ when the search reaches the deepest level of the tree.
Always considering a time window of one hour, from the results in Figure 5.5 we can see how in this way the algorithm calculates a z which allows to have results that are around the set goal. It does so very effectively with easy to anonymize datasets like the ecommerce one (Fig. 5.5a) and also the positions dataset which has the only weakness when traffic suddenly increases and the still low z exposes too many attributes (Fig. 5.5c). The dataset of domains is much more oscillating, but it never goes below 0.6 and manages not to anonymize too much using very high z (Fig. 5.5b).
Figure 5.6 illustrates how they perform instead with a more ambitious target such as $p_{k-anon} = 0.9$ and $k = 3$.
We see how the algorithm is perfectly capable of scaling z which follows the traffic volume almost perfectly. The domain dataset is also more stable around the target (Fig. 5.6a).
Obviously a higher target implies more anonymization - so blue bands more visible in the dataset of ecommerce and domains - and more generalization as evident in the case of locations where streets almost stop being published at the expense of cities and provinces (Fig. 5.6b).
In addition to these three examples, as mentioned earlier, we derived a fourth use case from the position dataset. Using in fact latitude and longitude we can map the territory into cells and frame each tuple in a certain cell. Using larger or smaller diameters, we can thus construct a generalization hierarchy at will and not rigidly tied to dataset categories. Figure 5.7 shows two examples with different diameters - specified in the legend - just to give an idea of how in certain areas it is possible to generalize elastically depending on dataset and objectives.



(a) Ecommerce

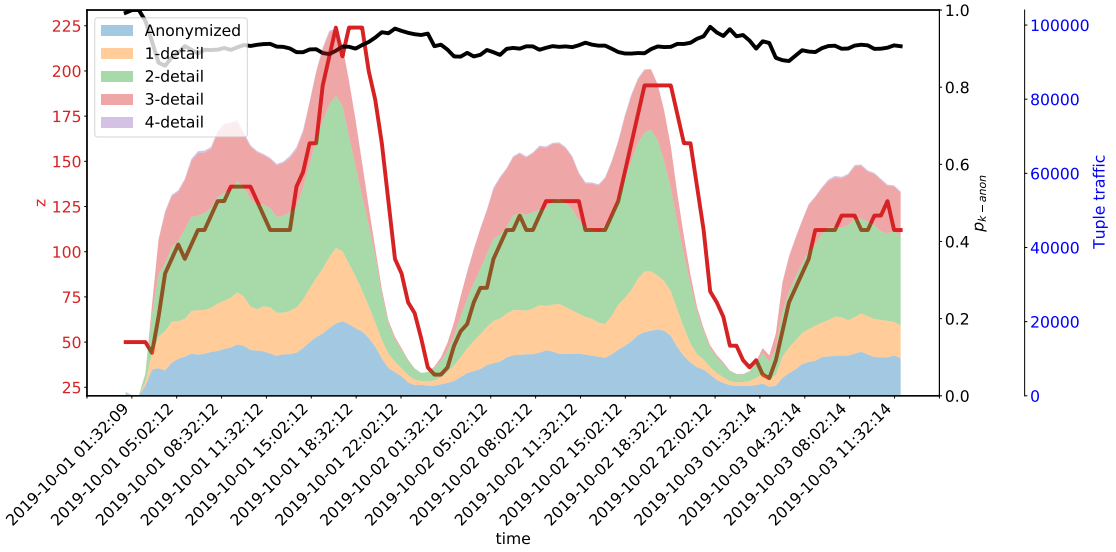


(b) Domains

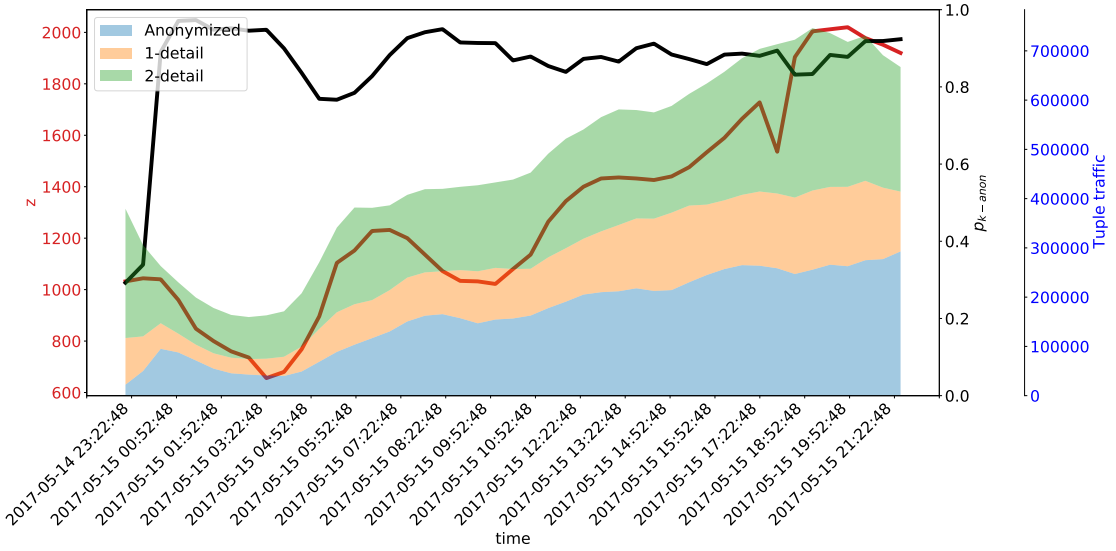


(c) Positionss

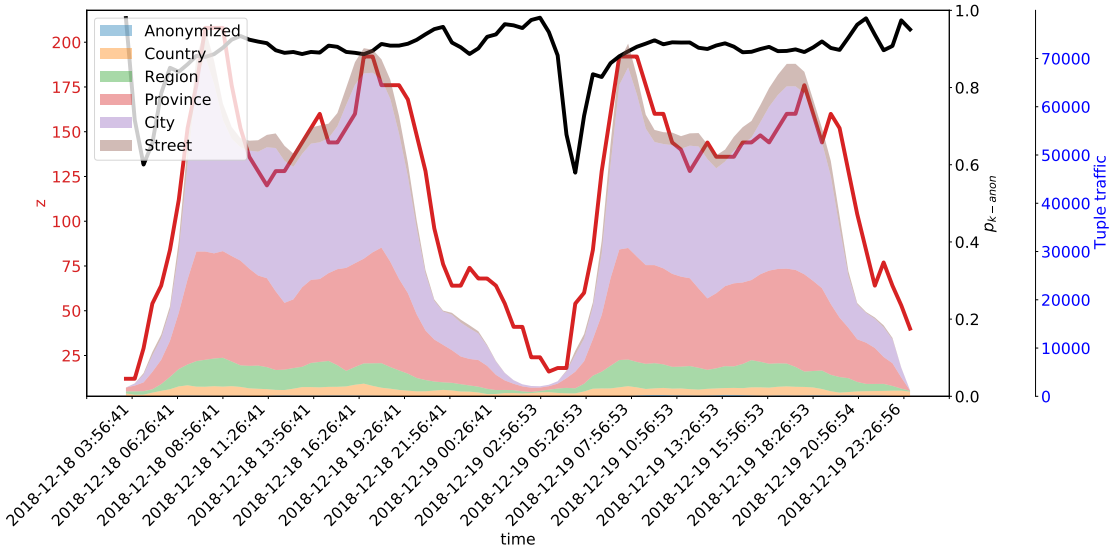
Figure 5.5: Binay search - $p_{kanon}\_goal = 0.8$, $k = 2$, $update\_rate = 30min$



(a) Ecommerce

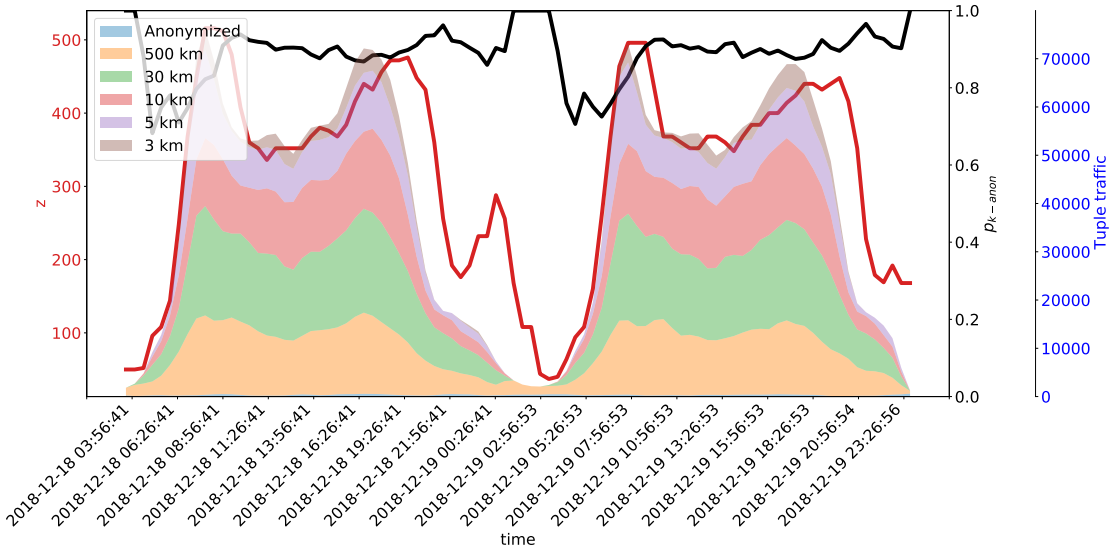


(b) Domains

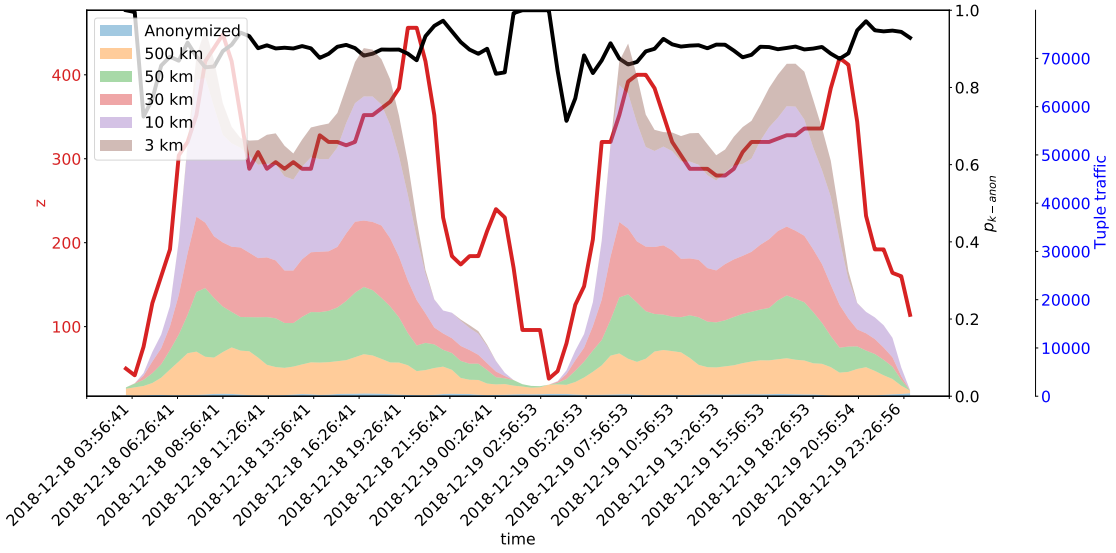


(c) Positionss

Figure 5.6: Binay search - $p_{kanon}\_goal = 0.9$, $k = 3$, $update\_rate = 30min$



(a) Positions with cells - first



(b) Positions with cells - second

Figure 5.7: Binay search - $p_{kanon}\_goal = 0.9$, $k = 3$, $update\_rate = 30min$

5.3 Entropy

Of course, higher values of z involve more anonymization and loss of information. How to measure this impact? One way to evaluate the amount of information is entropy, which is usually expressed as the number of bits required to store or transmit the information.
Formally a entropy H of a discrete random variable X with outcomes $x_1, ..., x_n$ which occur with probability $P(x_1), ..., P(x_n)$ is:

$$H(X) = -\sum_{i=1}^{n} P(x_i) log_2 P(x_i)$$

The concept of entropy can be applied to datasets if one considers each record as an outcome and its probability as the number of times it recurs in the dataset. If, for example, all records were unique their probability would be $\frac{1}{tot\_record}$.
Let's now consider our datasets. We can consider as records the users and as columns the attributes they have exposed. In this case we have groups of users who have exposed the same attributes and who will have the same outcome probability. The definition becomes:

$$H = -\sum_{G} \frac{|G|}{U} log_2 \frac{|G|}{U}$$

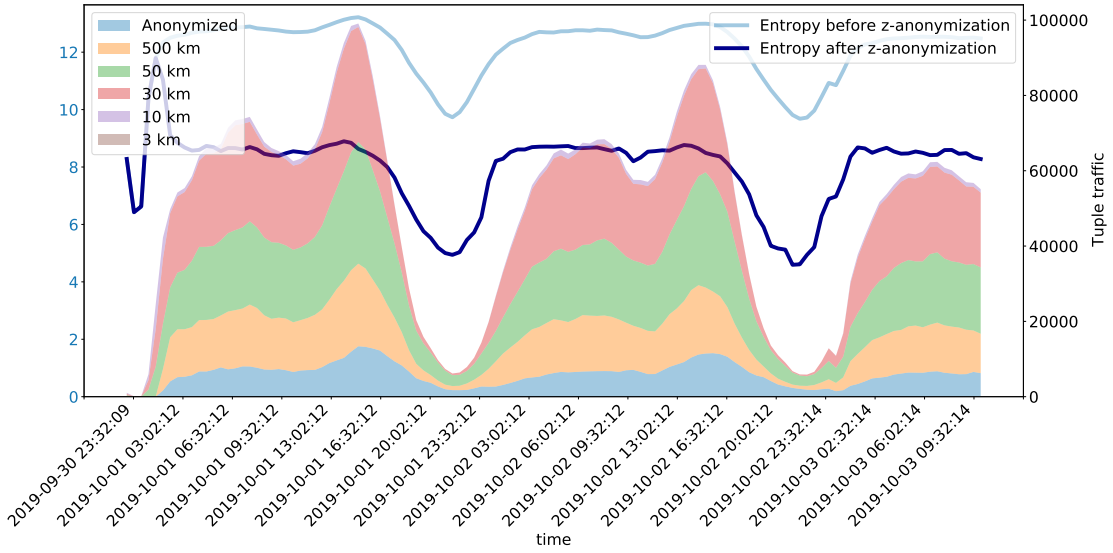
Where:

- G is a group of users with same attributes
- U is the group of all users.

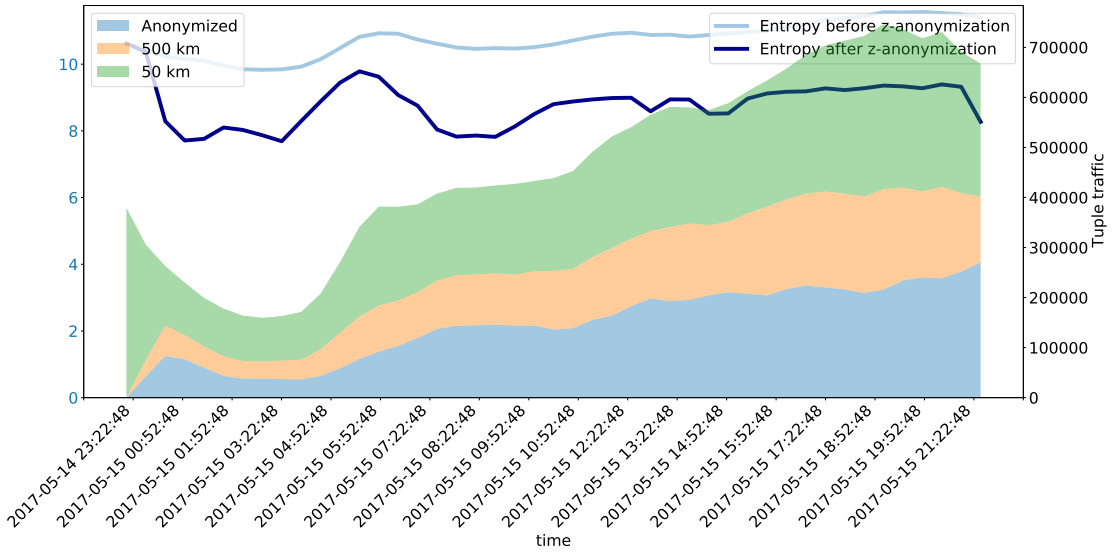
Note that any user belongs only to one group:

- $\Sigma_G |G| = U$
- $G_i \bigcap G_j = \emptyset \; \forall i, j, \; i \neq j$

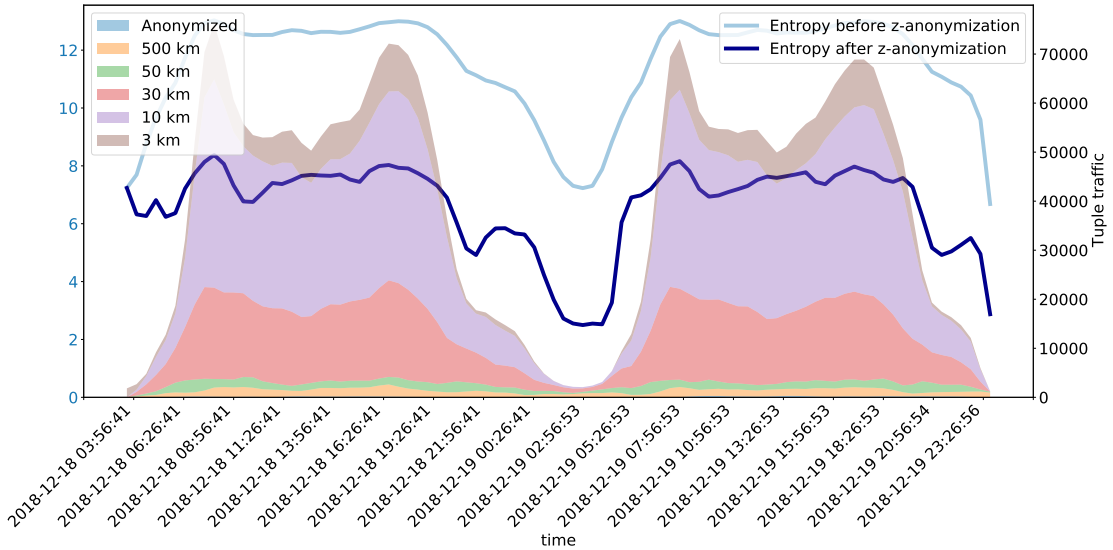
In the same way we can calculate the entropy on the deque introduced in the previous chapter and evaluate its path in time. In Figure 5.8 are plotted both the entropy that would have the data without going through z-anonymity (or using z =1) and using the self-adjusting z with binary search.
We see that ecommerce and positions datasets on average have higher entropy due to greater data complexity, many different products and therefore lot of information. They are also the datasets where, however, z-anonymity results in greater loss of information because the final data is generalized to a few groups that can be represented with a low amount of information as we saw in the characterization of the datasets. On the contrary, the domains dataset, even having more anonymization, releases more attributes than categories and this results in less information loss.



(a) Ecommerce



(b) Domains



(c) Positions

Figure 5.8: Entropy before and after z-anonymization - $p_{k-anon\_goal} = 0.8, k\_goal = 3$

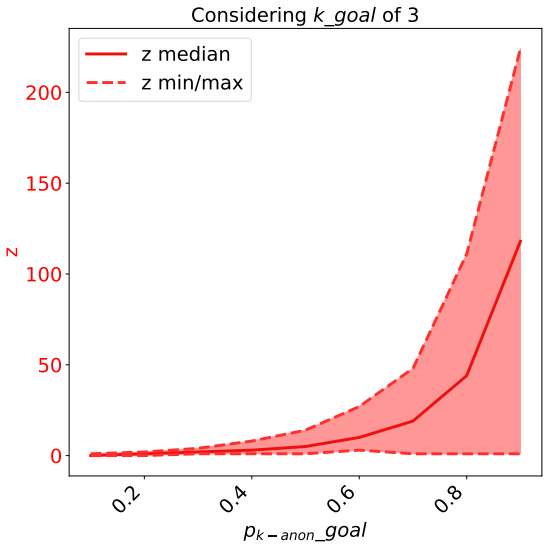
5.4 Impact of parameters on results

We have seen that by changing the parameters we have different results on the order of magnitude of z and therefore also the amount of information that is lost. Although it is intuitive that the value of z increases with higher targets, Figures 5.9 and 5.10 quantify exactly the range in which z moves when changing p_{k-anon}_goal and k_goal respectively. Tending to raise p_{k-anon}_goal corresponds to an exponential increase in the median of z for ecommerce and locations datasets. This is in line with the characteristics of the two datasets that are easier to anonymize and thus get a p_{k-anon}_goal of 0.7 with low z. The ecommerce dataset achieves spontaneous 3-anonymization without using z up to about p_{k-anon}_goal of 0.5. The domains dataset instead needs high z since p_{k-anon}_goal of 0.3 and it exceeds a median of 1000 for p_{k-anon}_goal of 0.9.
Increasing k_goal instead results in a linear increase in the median of z for all datasets (Fig. 5.10).

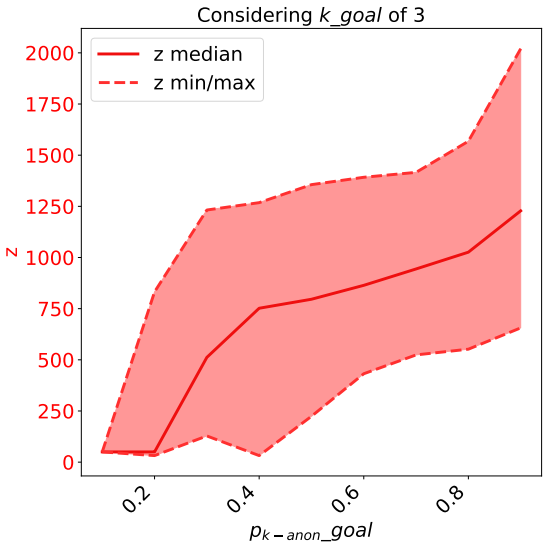
Using the formulas of previous paragraph, we saw that we can compute the original entropy and the entropy of the anonymized dataset. I will call residual information the ratio between the entropy after anonymization and before anonymization.

Definition 5.4.1 *Residual information = final H ÷ starting H*

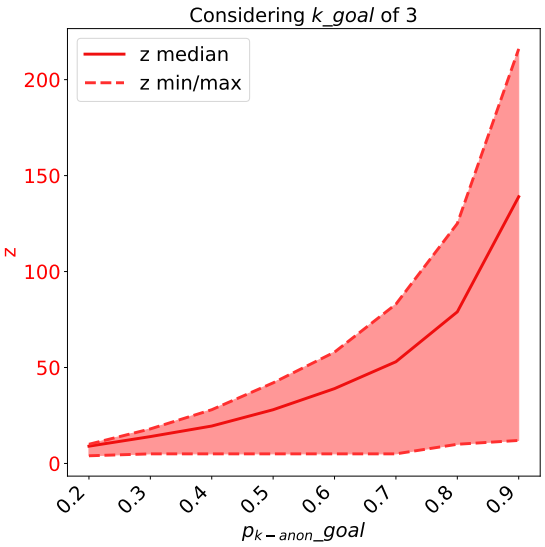
In Figure 5.11 we see how this residual information changes as p_{k-anon}_goal varies. The dataset for which the most information is lost is the ecommerce dataset while - as shown before - on the opposite side stands the domain dataset that seems to lose little information even with high p_{k-anon}_goal.
Varying k_goal even on entropy has no particular impact, at least not for the order of magnitude considered (Figure 5.12).



(a) Ecommerce

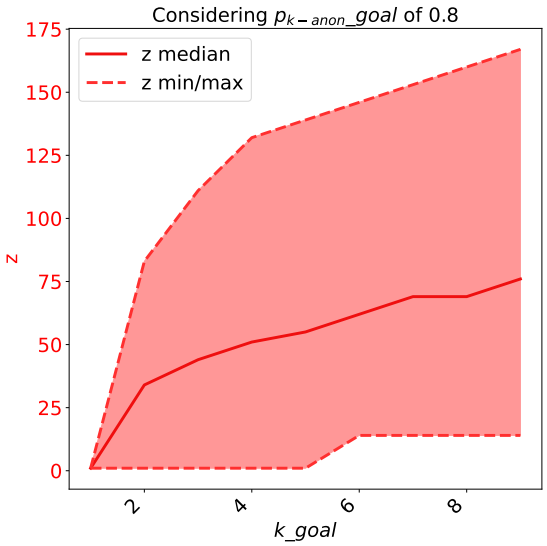


(b) Domains

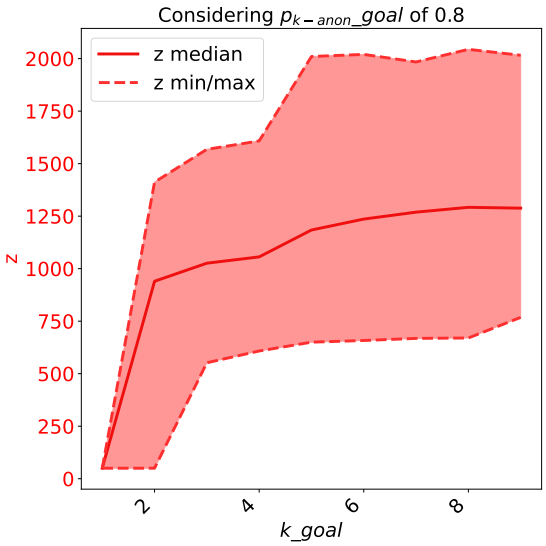


(c) Positions

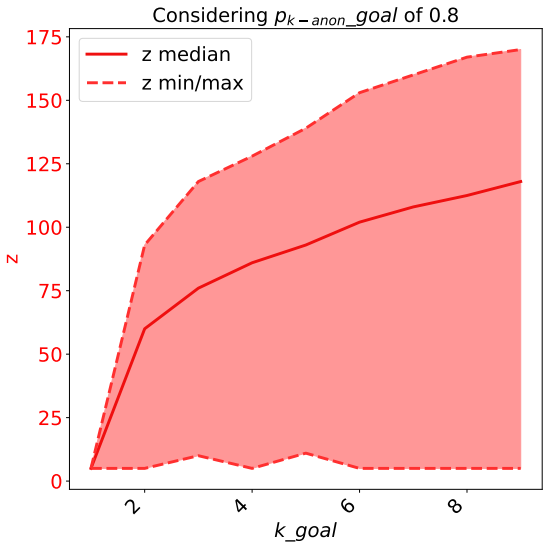
Figure 5.9: Impact of p_{k-anon_goal} on z



(a) Domains

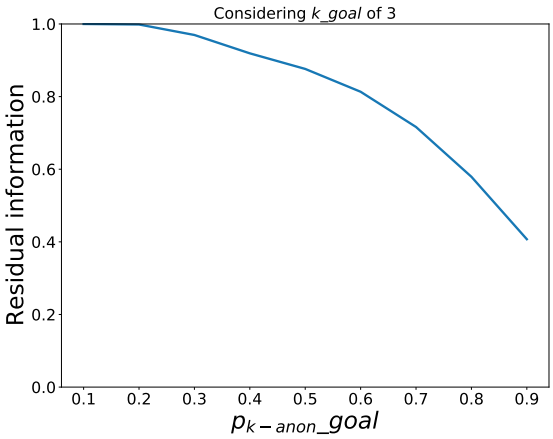


(b) Domains

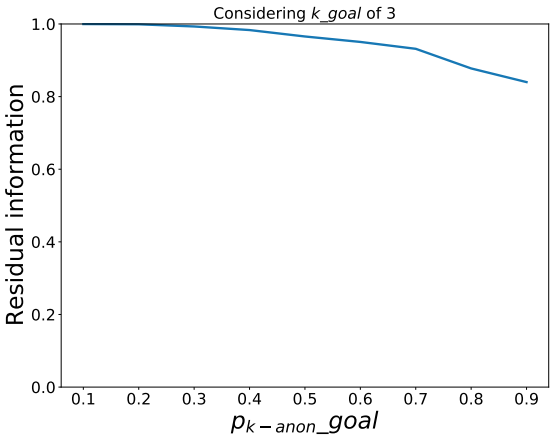


(c) Positions

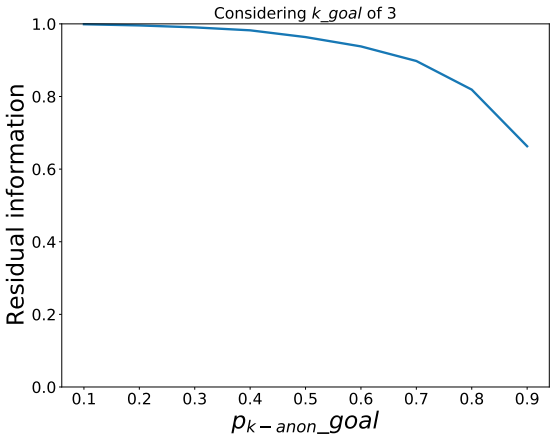
Figure 5.10: Impact of *k_goal* on *z*



(a) Ecommerce

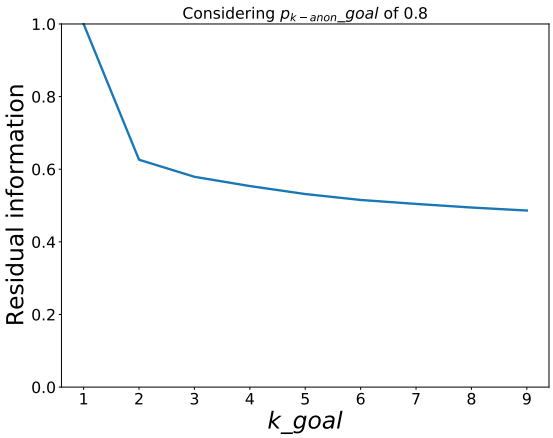


(b) Domains

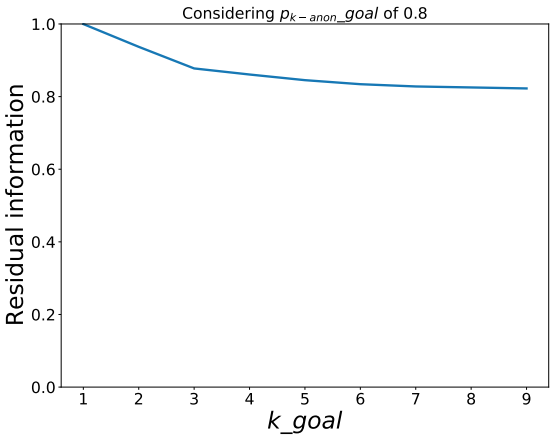


(c) Positions

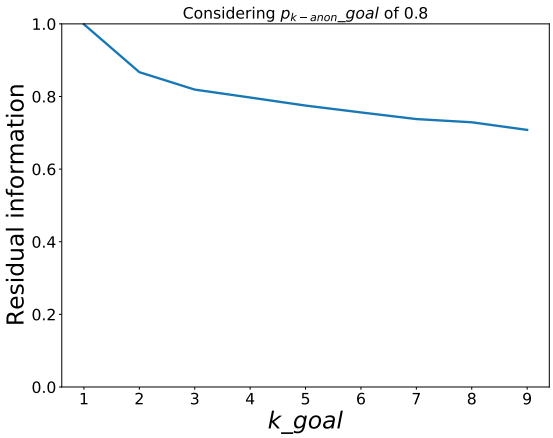
Figure 5.11: Impact of p_{k-anon}_goal on residual information



(a) Ecommerce



(b) Domains



(c) Positions

Figure 5.12: Impact of *k_goal* on residual information

Chapter 6

Adding noise

We have so far proceeded to protect the data by filtering the input tuples, a large selection at the entrance that only works by subtracting from the original data, a user-attribute pair can only be output as long as it is confused with other pairs with the same attribute. However, this condition can be met not only by suppressing but also by adding fictional user-attribute pairs that make an attribute no longer z-private.
By introducing a few fake pairs you can thus output attributes that were under threshold by a small margin and at the same time, as long as they are a minority, they do not compromise the usefulness of the final data.
Obviously, fake tuples must be consistent with previously issued data to prevent an attacker from easily recognizing them. The chosen user must therefore have exposed a similar attribute and in as recent time as possible.
If we keep generalization hierarchies we can derive a possible method to do that: every time we have to search for a user suitable for a certain attribute, we have to search among the users who have exposed the category that contains that attribute. For example, if we need users to output *Via Po*, we will have to look for them among those who have exhibited *Torino*. Categories already work as a clustering of attributes to search users from.

6.1 The algorithm

Algorithm 5 shows how it works. You start searching from the most recent users and form a list of users who meet this criteria. If there are enough users to fill the gap with z then they are put into output.
Obviously the appropriate user must not have exposed the attribute under consideration because it would not impact z-anonymity (line 11).
This mechanism can be triggered at any time, but for reasons of speed, usefulness

and data reliability it is best to use it only when the attribute is not too far from the threshold. we'll call this range $diff$ which is the distance from the counter to z.

Definition 6.1.1 $diff = z - c_a$

$diff$ can be left as a parameter either as an absolute value or relative to z.

Algorithm 5 Pseudo code of z-anon with noise

```
1: Input: (t, u, s)
2: ..Algorithm 2
3: if c_a >= z then
4:     OUTPUT (t, u, s)
5:     break
6: else
7:     fake_users ← initialize empty list
8:     diff = z − c_a
9:     cat = take first category of s
10:    for {u', t'} = last(H(cat)); first(H(cat)); next {u', t'} do
11:        if (u' is not u) and (u' not in H(s)) and (u' not in fake_users) then
12:            fake_users ← u'
13:            if length(fake_users) == diff then
14:                Break For cycle
15:            end if
16:        end if
17:    end for
18:    if length(fake_users) == diff then
19:        fake_users ← u
20:        for all u' in fake_users do
21:            OUTPUT (t, u', a)
22:        end for
23:    end if
24: end if
```

Let's try to see the effectiveness of the agorithm by isolating it from generalization. We applied z-anonymity on the datasets using binary search to get the optimal z. The parameters are: $p_{k-anon}\_goal = 0.8$, $k\_goal = 3$, $\Delta t = 1\ hour$, $update\_rate = 1\ hour$. Attributes and categories are then saved in the data structure but not given in output. Therefore, with diff = 0 data passes only if the attribute is above or below threshold as in the original z-anonymity.
Table 6.1 shows the results in terms of anonymization and Table 6.2 in terms of amount of data output. The results are computed for different $diff\ max$ which is the maximum distance between the counter and z for which noise algorithm is

applied. The results are very different from one dataset to another, for this reason the diff value must be calibrated and parameterized according to the use case. In general terms, the wider diff is the more tuples have the possibility to be output. For every tuple that is not anonymized, however, n fake tuples are emitted, creating imbalances in the output data. For the dataset of domains, that has a high degree of attributes, in the worst-case scenario means to increase the data in output also of 1313% against an improvement in anonymization of ten percentage points.

$diff\ max$	0	$z/4$	$z/3$	$z/2$
Ecommerce	84%	81%	79.8%	76.6%
Domains	46.3%	39.9%	38.8%	36.7%
Positions	85%	80.8%	78.9%	73.9%

Table 6.1: Noise - % of anonymization

$diff\ max$	0	$z/4$	$z/3$	$z/2$
Ecommerce	-84%	-57%	-35.3%	+50%
Domains	-46.3%	+494%	+704%	+1313%
Positions	-85%	-34.6%	+12.8%	+192%

Table 6.2: Noise - % of volume increase

6.2 Noise and generalization

The algorithm considered is hardly usable on its own. However, it could be used in combination with generalization. Some attributes close to the threshold would thus be output with a few fake tuples while the others are still generalized. The algorithm thus becomes a concatenation of Algorithm 4 and Algorithm 5. First we check if it is possible to emit fake tuples and if it is not we proceed to output a more general attribute.
To assess the impact of this hybridization, we again used residual information. Indeed, the addition of noise should allow tuples to be output that would otherwise be generalized, thus allowing more information to be preserved. The entropy of the output data is, in this case, calculated without considering fake tuples that would have made a comparison with the initial data impossible.
The results in Tables 6.3, 6.4 and 6.5 show that the anonymization percentages are not altered, what changes is the level of detail with which the tuples are emitted. In fact, a little perturbation allows in fact to preserve more residual information by decreasing the impact of generalization.

As already seen before when you consider a $diff\ max$ too wide with the output data you exceed the initial data. Especially in the dataset of domains, where you can more often use the perturbation, the volume increase is rather remarkable even with a $diff\ max$ of $z/6$ and $z/5$.

$diffmax$	0	$z/6$	$z/5$	$z/4$
Anonymization	11.9%	11.9%	11.9%	11.9%
Volume increase	-11.9%	-3.6%	+0.1%	+9.2%
Residual information	61.3%	63.2%	63.6%	64.4%

Table 6.3: Ecommerce - noise and generalization

$diffmax$	0	$z/6$	$z/5$	$z/4$
Anonymization	31%	31%	31%	31%
Volume increase	-31%	+177%	+296%	+558%
Residual information	89.2%	92.5%	93.3%	94.1%

Table 6.4: Domains - noise and generalization

$diffmax$	0	$z/6$	$z/5$	$z/4$
Anonymization	0.3%	0.3%	0.3%	0.3%
Volume increase	-0.3%	+18.6%	+28.3%	+48%
Residual information	81.9%	84.4%	84.9%	85.8%

Table 6.5: Positions - noise and generalization

Chapter 7

Conclusions and future works

Starting with z-anonymity, an algorithm that aims to anonymize large amounts of data without delays, we introduced several methods to optimize results and performances.
The results are clear: generalization allows to anonymize much less while maintaining high level of privacy and usefulness of the data. At the same time using a binary search algorithm to find z that reach a certain probability of satisfying k-anonymity at any given time allows the algorithm to be more resilient to privacy leaks in the considered time window. Finally, an algorithm to perturb the data with fake tuples was also proposed as an alternative or complement to generalization.
The algorithm has been tested on three use cases and shows that it works even with very different distributions of users and attributes.
The final parameters that the algorithm accepts are described in Table 7.1.

	Parameter	Type
	Δt	int
	p_{k-anon}_goal	float
	k_goal	int
	$generalization$	boolean
if not($generalization$)	z_0	int
	$tuning$	boolean
if($tuning$)	$update\_rate$	int
	$noise$	boolean
if($noise$)	$diffmax$	int

Table 7.1: Algorithm parameters

The algorithm is now available as a python module [1] and can be installed with the bash command:

```
pip install zanon
```

7.1 Limitations and future Works

z-anonymity only hides z-private attributes to avoid user reidentification. Other attacks based on timing or order of appearance are not considered. Moreover, the algorithm reasons only on the data of the last time window without saving any information about the periodicity of attributes or users over time. Future works may improve the algorithm by considering data about lower night or weekend traffic. This can be done in two ways: introducing machine learning concepts to allow the algorithm to recognize patterns in the data or inserting prior information.
This way it's possibile to take precautions to avoid that a sudden increase in data can lead to low p_{k-anon}, for example varying the time window, accommodating more hours at times of low traffic, or putting a minimum value at certain times to z.

[1] https://pypi.org/project/zanon/

Bibliography

[1] Arvind Narayanan and Vitaly Shmatikov. "Robust de-anonymization of large sparse datasets". In: *2008 IEEE Symposium on Security and Privacy (sp 2008)*. IEEE. 2008, pp. 111–125.

[2] Latanya Sweeney. "Simple demographics often identify people uniquely". In: *Health (San Francisco)* 671.2000 (2000), pp. 1–34.

[3] M. Khavkin and M. Last. "Preserving Differential Privacy and Utility of Non-stationary Data Streams". In: *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*. 2018, pp. 29–34. DOI: 10.1109/ICDMW.2018.00012.

[4] J. Domingo-Ferrer, J. Soria-Comas, and R. Mulero-Vellido. "Steered Microaggregation as a Unified Primitive to Anonymize Data Sets and Data Streams". In: *IEEE Transactions on Information Forensics and Security* 14.12 (2019), pp. 3298–3311. DOI: 10.1109/TIFS.2019.2914832.

[5] J. Cao et al. "CASTLE: Continuously Anonymizing Data Streams". In: *IEEE Transactions on Dependable and Secure Computing* 8.3 (2011), pp. 337–352. DOI: 10.1109/TDSC.2009.47.

[6] Jimmy Tekli et al. "(k, l)-Clustering for Transactional Data Streams Anonymization". In: *Information Security Practice and Experience*. Ed. by Chunhua Su and Hiroaki Kikuchi. Cham: Springer International Publishing, 2018, pp. 544–556. ISBN: 978-3-319-99807-7.

[7] Latanya Sweeney. "Achieving k-anonymity privacy protection using generalization and suppression". In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 10.05 (2002), pp. 571–588.

[8] Pierangela Samarati and Latanya Sweeney. "Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression". In: (1998).

[9] Pu Wang et al. "B-castle: An efficient publishing algorithm for k-anonymizing data streams". In: *2010 Second WRI Global Congress on Intelligent Systems*. Vol. 2. IEEE. 2010, pp. 132–136.

[10] Jianzhong Li, Beng Chin Ooi, and Weiping Wang. "Anonymizing streaming data for privacy protection". In: *2008 IEEE 24th International Conference on Data Engineering*. IEEE. 2008, pp. 1367–1369.

[11] Ankhbayar Otgonbayar et al. "K-VARP: K-anonymity for varied data streams via partitioning". In: *Information Sciences* 467 (2018), pp. 238–255. ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2018.07.057. URL: https://www.sciencedirect.com/science/article/pii/S0020025518305772.

[12] J. Zhang et al. "KIDS:K-anonymization data stream base on sliding window". In: *2010 2nd International Conference on Future Computer and Communication*. Vol. 2. 2010, pp. V2–311–V2–316. DOI: 10.1109/ICFCC.2010.5497420.

[13] J. Wang, C. Deng, and X. Li. "Two Privacy-Preserving Approaches for Publishing Transactional Data Streams". In: *IEEE Access* 6 (2018), pp. 23648–23658. DOI: 10.1109/ACCESS.2018.2814622.

[14] Soohyung Kim, Min Kyoung Sung, and Yon Dohn Chung. "A framework to preserve the privacy of electronic health data streams". In: *Journal of biomedical informatics* 50 (2014), pp. 95–106.

[15] Nikhil Jha , Thomas Favale , Luca Vassio , Martino Trevisan and Marco Mellia. "z-anonymity: Zero-Delay Anonymization for Data Streams". In: (2020).

[16] T. Favale et al. "α-MON: Traffic Anonymizer for Passive Monitoring". In: *IEEE Transactions on Network and Service Management* (2021), pp. 1–1. DOI: 10.1109/TNSM.2021.3057927.

[17] C. E. Shannon. "A mathematical theory of communication". In: *The Bell System Technical Journal* 27.3 (1948), pp. 379–423. DOI: 10.1002/j.1538-7305.1948.tb01338.x.

[18] E. A. Unger, L. Harn, and V. Kumar. "Entropy as a measure of database information". In: *[1990] Proceedings of the Sixth Annual Computer Security Applications Conference*. 1990, pp. 80–87. DOI: 10.1109/CSAC.1990.143755.

[19] Cynthia Dwork. "Differential Privacy: A Survey of Results". In: *Theory and Applications of Models of Computation*. Ed. by Manindra Agrawal et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 1–19. ISBN: 978-3-540-79228-4.