Using Grammatical Inference to Build Privacy Preserving Data-sets of User Logs

Victor Connes victor.connes@univ-nantes.fr
Colin De La Higuera cdlh@univ-nantes.fr
Hoël Le Capitaine hoel.lecapitaine@univ-nantes.fr
LS2N Université de Nantes - faculté des Sciences et Techniques (FST)
Bâtiment 34, 2 Chemin de la Houssinière, 44322 Nantes.

Editors: Jane Chandlee, Rémi Eyraud, Jeffrey Heinz, Adam Jardine, and Menno van Zaanen

Abstract

In many web applications, user logs are extracted to build a user model which can be part of further development, recommendation systems or personalization. This is the case for education platforms like X5gon. In order to obtain community collaboration, these logs should be shared, but logical privacy issues arise. In this work, we propose to build a user model from a data-set of logs: this will be a timed and probabilistic k-testable automaton, which can then be used to generate a new data-set having statistically close characteristics, yet have in which the original sequences have been sufficiently chunked the original data to not be able to identify the original logs. Following ideas from Differencial Privacy, we provide a second algorithm allowing to eliminate any strings whose influence would be too great. Experiments validate the approach.

Keywords: Open education, privacy data publishing, probabilistic finite automata, sequential data

1. Introduction

With websites more and more interactive, applications using user navigation traces or logs are abundant. This is the case in many settings and specifically in that of education. The recent political commitments in favor of open education, the COVID-19 pandemic context and the growing success of online learning platform constitute a major turning point for online education. In this context, large amounts of Open Educational Resources (OER) are available today on the web but these resources are disseminated on multiple repository websites. This dissemination of the resources does not facilitate the access and the navigation through the OER for the learner and constitutes an important obstacle to the diversity of resources in terms of providers, modality, language and cultures. A major challenge for open education is today to connect these repositories to make this diversity accessible and enjoyable for everyone.

It is in this spirit that project X5gon[1] was launched. X5gon aims to index these resources in a Global OER Network (GON) and to provide educational-driven tools to users for navigation through this network. Elaborating these tools is an important challenge, and allowing researchers to design their owns solutions and to compare them is arguably the best

[1] For more information about project X5gon, its search engine, its recommendation engine, its APIs, refer to www.x5gon.org.

way to achieve this goal. Logically, gathering and sharing user data is therefore a crucial issue to promote research and initiatives on this issue. This is all the more true since there is a lack of free data-sets on data related to open education in the state-of-the-art (Drachsler et al., 2015). This lack of data-sets can be explained by several factors at the forefront of which is the need to preserve the users privacy.

Indeed, the publication of user data sets cannot be done without guaranteeing the protection of users' personal data. Classical approaches anonymising a subset of the actual user data and publishing it have been shown to be very ineffective in terms of privacy (Mayer et al., 2016; Riederer et al., 2016; Cecaj et al., 2016). Unfortunately, the anonymisation process remains a challenging open question: It is difficult to ensure with certainty that an effective anonymisation process does not degrade the data too much and, at the same time, ensures total anonymisation of the users. Taking into consideration this statement, we choose to focus on another kind of approach to obtain a shareable data-set. We train a probabilistic model on real user activities and to use this model to generate artificial user activities which mimic as closely as possible the characteristics of the real data-set.

The model we choose is a new type of finite state machine, both timed and probabilistic (Section 2). Then, we show how this automaton is able to generate an arbitrarily large data-set with statistical characteristics close to the initial data-set. In order to deal with privacy issue, we use ideas from a very popular privacy framework to analyze the obtained automaton and remove data-points which would have a too big importance and could therefore be identified from the automaton or the newly generated data (Section 3). We also give some properties of this algorithm. Finally, we apply our method on a set of user interactions over OERs coming from the X5gon project (Section 4). These methods have been applied with success on data from the X5gon data-set which was built in 2020 for an international hackathon[2].

2. From sequential data-sets to Probabilistic Timed k-Testable Automata

Log data-set and data-set of strings The data for user logs can come in many forms. We will use an abstraction where a user session is a sequence of events, each event occurring during some specific time. We will suppose that the number of possible events is finite, corresponding to specific web-pages visited or resources consulted. A *timed sequence* X is $X = (d_1, s_1), \ldots, (d_i, s_i)$, where (d_i, s_i) are duration and event pairs of the log. Consequently a data-set of timed sequences is a multi-set of timed sequences (e.g: $DL = \{X_1, X_2, \ldots, X_{|DL|}\}$). The usual definition of timed sequence uses time instead of duration but since in a practical setup it is easy to transform time into duration, we choose duration, more convenient for our purpose.

With each Log data-set we can also associate a multi-set of strings over the alphabet of events by forgetting the times. And we define a *data-set of sequences* D of size $|D|$ composed of a multi-set of events sequences, that are simply strings, $D = \{w_1, w_2, \ldots, w_{|D|}\}$.

Frequency Deterministic Finite State Automata (FDFA) Fdfa have been introduced by de la Higuera (2010) as a model counting how many times a state or a transition is

[2] https://www.x5gon.org/event/hackathon/

used when parsing a particular data-set. They can easily be transformed into Probabilistic Deterministic Finite State Automata (Pdfa) through normalization.

Definition 1 *A* frequency deterministic finite state automaton *(FDFA) is a 6-tuple $\langle Q, q_0, \Sigma, \delta, C, I\rangle$ where $Q = \{q_0 \dots q_n\}$ is the set of states with q_0 the initial state, transition function is $\delta : Q \times \Sigma \to Q$. Counts are given by $C : (Q \times \Sigma) \cup Q \to \mathbb{N}$ where $C(q, a)$ indicates the number of times strings use transition, and $C(q) \in \mathbb{N}$ indicates the number of times state q is used for ending. $I \in \mathbb{N}$ indicates how many strings have started. A specific consistency condition ensures that the counts entering a state equal those exiting it. The following constraint holds:*

$$\forall q \in Q,\ C(q) + \sum_{a\in\Sigma} C(q, a) = \sum_{q'\in Q, b\in\Sigma \text{ s.t. } \delta(q',b)=q} C(q', b) + I \text{ if } q = q_0$$

Counts are frequencies, and through normalization we can obtain relative frequencies which can be interpreted as probabilities. Hence both of the local transition and of each individual string: We write $\mathbb{P}^Q_{\mathcal{A}}(q, a) = \frac{C(q,a)}{C(q)+\sum_{b\in\Sigma} C(q,b)}$ and $\mathbb{P}^Q_{\mathcal{A}}(q) = \frac{C(q)}{C(q)+\sum_{b\in\Sigma} C(q,b)}$ for the probabilities to generate symbol a and (resp.) to halt when in state q.

The probability $\mathbb{P}_{\mathcal{A}}(s)$ of (generating) by finite state automaton $\mathcal{A}$ a string $s \in \Sigma^*$ can be computed by first extending δ: $\delta(q, \lambda) = q$; $\delta(q,(a_1 \dots a_n)) = \delta(\delta(q, a_1), a_2 \dots a_n))$.

And with the frequencies we can also associate probabilities recursively:

$$\begin{aligned} \mathbb{P}(q, \lambda) &= \mathbb{P}^Q(q) \\ \mathbb{P}(q, a_1 \dots a_n) &= \mathbb{P}^Q(q, a_1) \cdot \mathbb{P}(\delta(q, a_1), a_2 \dots a_n) \end{aligned} \tag{1}$$

A Pdfa $\mathcal{A}$ models a distribution $\mathcal{D}_{\mathcal{A}}$. And by extension, so does a Fdfa. We will drop the subscript $\mathcal{A}$ when there is no ambiguity. Fdfa can be easily derived from Pdfa in linear time.

K-testable machines K-testable languages in the strict sense (k-TSS) were introduced by McNaughton and Papert (1971). Intuitively, a k-TSS language is determined by a finite set of sub-strings of length at most k that are allowed to appear as sub-strings in the strings of the language. It has been proved that, unlike for regular languages, algorithms can learn k-TSS languages in the limit from text (Garcia et al., 1990) which has made these models attractive and used in many applications. A k-TSS language is determined by sets of strings of length $k - 1$ that are allowed as prefixes and suffixes and by sets of strength of length k sub-strings, respectively, together with all the short strings (with length at most $k - 1$) contained in the language.

Readers interested in explanation about how to build a Deterministic Finite-state Automaton (Dfa) recognizing a k-TSS language may refer to (de la Higuera, 2010).

Probabilistic Timed k-Testable Automata The goal of this work is to obtain a generative model for timed sequences. For this, we introduce Probabilistic Timed k-Testable Automata (PTk-TA) which is a timed extension of Probabilistic Finite State Automata, with a k-testable structure, allowing to model the duration of the transitions. In a PTk-TA the Pdfa properties are directly issued from the k-test vector. In classical timed automata (Lasota and Walukiewicz, 2008), time is measured upon transitions and guards are used mainly to determine the acceptance or the rejection of a timed word. Our model is generative, so guards are not used. Instead, a (different) time distribution is associated with each transition, given by the mean and the variance of a Normal distribution.

Definition 2 *A Probabilistic Timed k-Testable Automata (PTk-TA) is a six-tuple* $\langle Q, q_0, \Sigma, \delta, \mathbb{P}, \Delta_t \rangle$ *where* $\langle Q, q_0, \Sigma, \delta, \mathbb{P} \rangle$ *is a* Dfa *and* Δ_t *is a probabilistic model for duration of transitions given by* $\Delta_t : Q \times \Sigma \rightarrow \mathbb{R}^+ \times \mathbb{R}^+$.

Each transition (q, a), $\Delta_t(q, a)$ is a pair (μ, σ), used as parameters of a positive Gaussian $\mathcal{N}^+(\mu_{(q,a)}, \sigma_{(q,a)})$, where $\mu_{(q,a)}$ is the average duration for this transition and $\sigma_{(q,a)}$ is its variance.

Algorithm 1 can be used to generate a random timed sequence with a PTk-TA.

Algorithm 1: Sequence generation algorithm

```
Input: $A$: timed Pdfa
Result: a timed sequence X
$q = q_0$
while $not\ end$ do // Iterate until $F$ is drawn
    $a \sim next(q)$
    $\mu, \sigma \leftarrow \Delta_t(q, a)$
    $d \sim \mathcal{N}^+(\mu, \sigma)$// d is drawn from the positive Gaussian distribution $\mathcal{N}^+(\mu, \sigma)$
    $X \leftarrow X \cup (a, d)$
    $q \leftarrow \delta(q, a)$
end
Return X
```

Statistics can also be extracted for each PTk-TA, and we can compute the expected duration of a sequence, for example.

The usual algorithm allowing to build a k-testable automaton can be adapted to compute, from a data-set of timed sequences, probabilities for transitions and final states and to estimate all the parameters for the different Δ_t functions. The result will thus be a Probabilistic Timed k-Testable Automata.

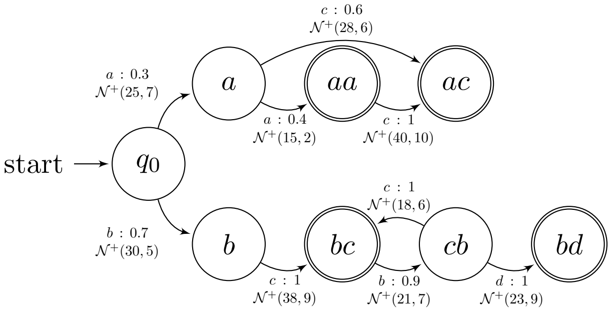

Figure 1: Toy example of Probabilistic Timed k-Testable Automata

In Figure 1 we represent a simplified Probabilistic Timed 2-Testable Automaton. The alphabet is $\{a, b, c, d\}$. Event c, when taking place in state (a), leads us to state (ac). Its probability is 0.4 and the duration is a value drawn from a Normal distribution $\mathcal{N}^+(28, 6)$.

From this automaton a typical sequence generated by this PTk-TA could be $(b, 22), (c, 27), (b, 15), (d, 21)$. Such a sequence would correspond to user browsing page b for 22 seconds, then page c for 27 seconds, then page b again for 15, and finally the page d for 20seconds and halting.

3. Dealing with privacy issues

Given the Probabilistic Timed k-Testable Automata obtained from the proposed construction can be used to generate new strings: the distribution of these strings is close to the original distribution (from the data-set). The k-testable method chunks data and therefore will not preserve the long term dependencies: this is sometimes seen as an obstacle in using this method but can be seen as an advantage here.

Yet several privacy issues still remain unsolved. These are studied in the field of *differential privacy* and our aim is to follow their guidelines. We should remember that the original data-set contains user navigation logs so these are clearly concerned by these questions. Considering the users are themselves learners, this is of course a serious issue.

Before giving relevant mathematical definitions, let us try to present the issues in a more informal way. The question raised is the following: “Can some information in the learned model depend too heavily on a unique user/log?”. A positive answer to this question is problematic for two reasons: (1) it shows a low robustness of the model, and (2) too much information about this single user can be seen. We therefore want to detect this situation and clean the data-set accordingly. In the context of Probabilistic Timed k-Testable Automata, this means scrutinizing the different transitions of the corresponding Fdfa and checking if the counts for one single string don’t exceed a given threshold.

Some notions from differential privacy Differential privacy (DP) is a mathematical setup introduced by Dwork et al. (2006) to ensure provable privacy guarantee. It was originally developed to deal with linkage attacks and background knowledge based attacks. These attacker models try to re-identify individuals in an anonymized data-set by combining that data with background information; typically in our case this could be a specific url accessed. These attacks are often fruitful even on sequential data has demonstrated in Mayer et al. (2016); Riederer et al. (2016); Cecaj et al. (2016). Fortunately, DP is completely independent of the adversary’s background or prior knowledge.

The main technique used in DP consists in adding noise to a data-set so that an adversary cannot decide whether a particular record (e.g., a sequence in our case) is included in the data-set or not. It is currently the most widely accepted formal mathematical model that ensures privacy protection (Yang et al., 2017). The main advantages of DP over others frameworks are: 1) The robustness against post-processing: any post-processing computation will also be differentially private. 2) The composition-property: composability of differentially private computations with a controlled degradation of the privacy loss. More precisely, the sequential execution of k algorithm $f_1, \ldots, f_k$ each satisfying ϵ_i-*differentially private* results in an algorithm that is ϵ-*differentially private* for $\epsilon = \sum_i \epsilon_i$ (McSherry and

Talwar, 2007). 3) The group privacy: possibility to ensure the privacy of a group rather than of single individuals.

Definition 3 *(Differential Privacy) Dwork et al. (2006). A randomized function f is (ϵ, γ)-differentially private if for any two data-sets D and D' such that D differ of D' on only one record and for any possible output $Y \subseteq Img_f$ it holds that:*

$$P[f(D) \in Y] \leq exp(\epsilon) P[f(D') \in Y] + \gamma$$

As a particular case, if f is $(\epsilon, 0)$-*differentially private* (i.e $\gamma = 0$), then we say it is ϵ-*differentially private*. In this definition, γ controls the probability of uncontrolled breach due to the possible stochastic nature of the algorithm and ϵ controls the multiplicative worst-case privacy loss if no breach. Obviously, the mechanism does not only need to be private. It should also keep as well as possible under private constraint the original distribution of the data-set. Broadly speaking, the *utility* of a mechanism is its capability to minimize the error expressed as the distance between the original db/statistics and the generated output db/statistics. Unfortunately, DP often lead to very poor utility of the data-set released in case a sequential data, final data-set distribution far from original one (Yang et al., 2017; Shaked and Rokach, 2020).

Pruning the automaton and the data-set The ϵ-sensitivity we define here is a relaxation of differential privacy. We assume that any removed input string does not impact more than by a multiplicative factor ϵ, the probability $\mathbb{P}^Q(q, a)$ of any transition.

Let us note that if $D \subseteq D'$ and $\mathcal{A}_D$ is the Fdfa obtained through the k-testable algorithm from D ($\mathcal{A}_{D'}$ for D'), it follows that $\mathcal{A}_D$ is a sub-automaton of $\mathcal{A}_{D'}$, and all its states and transitions appear also in $\mathcal{A}_{D'}$. This is a consequence of the k-testable construction method. We will use this result in the sequel.

Definition 4 *(ϵ-sensitivity) Let D be a data-set. Let $\mathcal{A}$ be a k-testable* Fdfa *and w be a string from the building corpus. We say that w is ϵ-sensitive for the transition (q, a) in A (here a can be instanced by λ) iff:*

$$\mathbb{P}^Q_{\mathcal{B}}(q, a) < \epsilon \mathbb{P}^Q_{\mathcal{A}}(q, a)$$

where $\mathcal{B}$ is the k- testable Fdfa *built from the same corpus of $\mathcal{A}$ but without the string w.*

This may be rewritten using frequencies as:

$$\frac{C_{\mathcal{A}}(q, a) - C(w, q, a)}{C(q) + \sum_{c \in \Sigma} C(q, c) - C(w, q, a)} < \epsilon \frac{C_(q, a)}{C(q) + \sum_{c \in \Sigma} C(q, c)}$$

An Fdfa $\mathcal{A}$ without any ϵ-sensitive transitions for any string in D is said to be ϵ-private on D.

Algorithm 2 prunes the input Fdfa in order to guarantee that no remaining transitions in the output Fdfa are ϵ-sensitive for any string. The algorithm iterates over D and parses each string in D while increasing $C(w, q, a)$ each time the transition (q, a) is reached by the string w. We assume that since D is a multi-set, it is computed independently for each w

Algorithm 2: Pruning algorithm

```
Input: $\langle Q, q_0, \Sigma, \delta, C, I \rangle$: Fdfa, $D$: Sequential database, $\epsilon$: [0, 1]
Result: $\epsilon$-private pruned Fdfa $\langle Q, q_0, \Sigma, \delta, C, I \rangle$
$\forall w \in D\ \forall (q, a), C(w, q, a) \leftarrow 0$
foreach $w \in D$ do // Iterate over all strings
    $q \leftarrow q_0$
    foreach $a \in w$ do // Iterate over symbols in $w$
        $C(w, q, a) + +$
        $q \leftarrow \delta(q, a)$
    end
    $C(w, q, \lambda) + +$
end
$convergence \leftarrow False$
$\forall q \in Q, C(q, *) \leftarrow \sum_{a \in \Sigma} C(q, a)$
$D' \leftarrow D$
while not convergence do
    $convergence \leftarrow True$
    foreach $\forall w \in D', \forall (q, a) \in (Q \times \Sigma) \cup Q, C(w, q, a) \neq 0$ do
        if $\frac{C(q,a)-C(w,q,a)}{C(q,*)-C(w,q,a)} < \epsilon \frac{C(q,a)}{C(q,*)}$ then // Check if $w$ has an $\epsilon$-sensitive transition
            $convergence \leftarrow False$
            $D' \leftarrow D' \setminus \{w\}$
            $\forall (q, b) \in (Q \times \Sigma) \cup Q, C(w, q, a) \neq 0, C(q, b) \leftarrow C(q, b) - C(w, q, a)$
            $\forall q \in Q, C(w, q, a) \neq 0, C(q, *) \leftarrow C(q, *) - C(w, q, a)$
        end
    end
end
return counting(D', k), D'
```

in D. Additionally, for each state reached, we compute the total number of times $C(q, *)$ it is reached by any string. Finally we duplicate the input data-set D into D'. The second step makes the pruning job, until convergence we iterate over each triplet (w, q, a) with non zero values of count, rephrasing all transitions q, a reached by each string w. Whenever the given string is ϵ-sensitive on this given transition, we delete it from the data-set and update all the global counts for each transition reached by the strings as well as for each state. Convergence is reached when no ϵ-sensitive string remains for any transition. Once done, we return the obtaining Fdfa with the new pruned data-set.

Convergence Consider a Fdfa, and two strings w_1, w_2 from the corpus used to construct the Pdfa. We assume w_1 and w_2 are ϵ-sensitive for the transition (q, a). Such that after the pruning of w_1, w_2 they remain ϵ-sensitive for the transition (q, a) because:

$$\frac{C(q,a) - C(w_2,q,a)}{C(q) + \sum_{b \in \Sigma} C(q,b) - C(w_2,q,a)} < \epsilon \frac{C(q,a)}{C(q) + \sum_{b \in \Sigma} C(q,b)}$$

$$\implies \frac{C(q,a) - C(w_2,q,a) - C(w_1,q,a)}{C(q) + \sum_{b \in \Sigma} C(q,b) - C(w_2,q,a) - C(w_1,q,a)} < \epsilon \frac{C(q,a) - C(w_1,q,a)}{C(q) + \sum_{b \in \Sigma} C(q,b) - C(w_1,q,a)}$$

We can easily deduce of that a proof of convergence for our pruning algorithm.

Some properties The above algorithm does two things: on one hand, it ensures ϵ-privacy by removing logs from the data which would be ϵ-sensitive. In doing so the resulting new data-set and its corresponding Probabilistic Timed k-Testable Automata have the property which is of help to ensure or weak form of ϵ-differentially privacy: the impact of each individual sequence is locally small.

The following definition is relevant:

Definition 5 *Let $\mathcal{A}$ and $\mathcal{B}$ be 2* Fdfa *sharing the same structure. $\mathcal{A}$ and $\mathcal{B}$ have ϵ-*close *counts if for every state q and every symbol a, we have, using the definitions of probabilities from Equation (1),*

$$\begin{aligned}&\text{if } \mathbb{P}_{\mathcal{A}}(q, a\Sigma^*) \leq \mathbb{P}_{\mathcal{B}}(q, a\Sigma^*),\ \mathbb{P}_{\mathcal{A}}(q, a\Sigma^*) < (1-\epsilon)\mathbb{P}_{\mathcal{B}}(q, a\Sigma^*)\\ &\text{if } \mathbb{P}_{\mathcal{B}}(q, a\Sigma^*) \leq \mathbb{P}_{\mathcal{A}}(q, a\Sigma^*),\ \mathbb{P}_{\mathcal{B}}(q, a\Sigma^*) < (1-\epsilon)\mathbb{P}_{\mathcal{A}}(q, a\Sigma^*)\end{aligned} \tag{2}$$

We now have as an immediate corollary:

Proposition 6 *If $\mathcal{A}$ is ϵ-private for data-set D, then it is ϵ-differentially private.*

There are other issues here: one important question is to understand in what way Algorithm 2 is going to modify the initial distribution. This poses interesting (and unanswered) questions: given two Pdfa sharing a same structure, do small local changes to the probabilities have only a bounded influence on the difference between the two distributions?

4. Case study: X5gon user data

We evaluate our method on a data-set of users from project X5gon. For our experiments we try different values of k (from $k = 1$ to $k = 5$ but for the sake of staying concise and without loss of generalisation we choose to present result only for $k = 2$ and $k = 4$). And for each k, different values of ϵ ($\epsilon = 0$, $\epsilon = 0.25$, $\epsilon = 0.5$, $\epsilon = 0.75$). For each couple of values a Probabilistic Timed k-Testable Automata is inferred and we generate a new artificial data-set of 10000 sequences.

Statistics and specifics of the data-set The X5gon data-set is composed of 100217 sessions totaling more than 2 million log-items over 13787 resources. Resources are accessed 154 times in average in our data-set, nevertheless the distribution follows a power-law so 25% of the resources have less than 2 accesses and 50% less than 71 accesses. Table 1 details the statistics in terms of log-items by sessions in its first row, with a session length varying between 4 and 2605, but more than 90% of the sessions contain less than 50 log-items. Similarly, Table 2 details statistics in terms of the duration of timed sequences. In the first row, the average timed sequence duration is about $\approx$ 1h20min, but we observe a large variance such that 50% of timed sequences last less than 31m and around 90% more than 4 hours.

These statistics show that it is going to be hard to anonymize the data-set while conserving a good utility. Firstly, the large universe size (number of resources) is infrequent on data-sets in the literature. As an example MSNBC and STM which are classical benchmarks have respectively a universe size of 17 and 342. Their universe size is very low compared to the size of our data-set (13787) while the number of sequences is roughly the same ($\approx$

1 *million*) in the three data-sets (MSNBC, STM, and X5gon). In terms of automata this means a huge alphabet size, resulting in a very large number of states, as seen in Table 3

The same applies for the string length which is significantly larger on average in our data-set ≈ 21 against ≈ 5 or 6 in the other data sets. Secondly, many resources have a very small number of accesses which makes them sensitive in terms of privacy. Indeed, it is the very purpose of linkage attacks to find events that are rare enough to be used as quasi-identifiers for users. For this reason following the Definition 4, the strings containing these resources are so-called ϵ-sensitive and are removed of the data-set during the pruning algorithm.

			count	min	max	mean	std	25%	50%	75%	90%
raw data			100253	4	2605	21.31	31.80	6	12	23	46
(RD)	k=2	ϵ= 0.25	12241	4	148	9.23	9.21	5	6	10	17
	k=4	ϵ= 0.25	5489	4	125	6.91	6.36	4	5	7	11
(GD)	k=2	ϵ= 0	10000	2	191	21.1	20.04	7	15	28	46
		ϵ= 0.25	10000	2	110	9.2	7.68	4	7	12	19
	k=4	ϵ= 0	10000	4	315	21.21	19.68	7	15	28	46
		ϵ= 0.25	10000	4	76	6.96	4.77	4	5	8	12

Table 1: Distribution of session lengths for raw, remaining and generated data. (RD) stands for remaining data and (GD) for generated data

			count	min	max	mean	std	25%	50%	75%	90%
raw data			100253	0	18h57m	1h21m	1h57m	6m	30m	1h43m	4h03m
(RD)	k=2	ϵ=0.25	12241	0	13h53m	32m	1h7m	1m	8m	30m	1h25m
	k=4	ϵ= 0.25	5489	0	14h27m	23m	59m	0	4m	18m	54m
(GD)	k=2	ϵ= 0	10000	0	1d3h30m	2h18m	2h40	28m	1h24m	3h11m	5h45m
		ϵ= 0.25	10000	0	1d19h46m	1h5m	1h25m	15m	39m	1h25m	2h33m
	k=4	ϵ= 0	10000	0	18h57m	1h46	2h05m	21m	1h	2h25m	4h31m
		ϵ= 0.25	10000	0	22h27m	39m	58m	10m	23m	48m	1h26m

Table 2: Distribution of session durations for raw, remaining and generated data. (RD) stands for remaining data and (GD) for generated data

Length and duration statistics The tables Table 1 and Table 2 report distributions of respectively length and duration of timed sequences for real data, data remaining after the pruning step and final generated data. By comparing remaining data and their corresponding generated data, we observe that the distributions in terms of length and in terms of duration are well conserved by the generation algorithm.

As explained before, obtaining privacy guaranteed on this data-set implies the removing of many infrequent strings. Therefore, we observe a gap in terms of distributions between real data and remaining ones which is propagated on generated data. We observe that the longest strings are more likely removed than the short one, this is a logical consequence of our pruning algorithm. Since the longer is a string, the more likely it is to use a transition where it is $\epsilon-$sensitive. Even if the tables present the result only for ϵ=0.25, we observe a similar behaviour for other value of ϵ (0.5, 0.75).

Model parameters Table 3 reports the number of deleted strings, symbols as well as the number of states and transitions for different values of k and ϵ. We observe the effect of pruning: even for the smaller values of ϵ, only 29%(4068/13787) of the resources are kept. This is explained by the ratio of accesses over resources. Since many resources have few accesses, strings containing these will be deleted by Algorithm Pruning in order to preserve privacy. Additional results not reported here show that for ϵ lower than 0.5, the observed parameters ($|D|$, $||D||$, $|Q|$, n_{trans}) are still close to those observed for $\epsilon \in \{0.25, 0.5\}$. At the opposite, for ϵ higher than 0.5, these parameters become significantly lower. This suggests that some strings are observed so infrequently that they are a privacy hazard and are therefore removed even for very low ϵ values. The reason why these strings are hazards for privacy is that they are ideal targets for linkage attacks so they are ϵ-sensible according to the definition Definition 4 and removed. From $\epsilon = 0.5$ on-wards other strings with more moderate privacy hazards are gradually removed.

			$\|D\|$	$\|\|D\|\|$	$\|Q\|$	n_trans
model parameters	k=2	ϵ=0	100257	13787	184600	471356
		ϵ=0.25	12241	4068	8033	15573
		ϵ=0.5	11506	3946	7577	14428
		ϵ=0.75	1619	814	992	1452
	k=4	ϵ=0	100257	13787	697057	1002063
		ϵ=0.25	5489	3633	7670	10354
		ϵ=0.5	5437	3633	7616	10256
		ϵ=0.75	1156	802	992	1259

Table 3: Model parameters with $|D|$ the number of strings, $||D||$ the number of symbols (resources), $|Q|$ the number of states, and n_trans the number of transitions

Count queries Count query consist in counting the number of times a given sub-string (the query) appears in a data-set. Count queries are used as an evaluation method in literature (Chen et al., 2012b) for that the relative error the relative error between the number of occurrences in the real data-set D and the number of occurrences in the generated one $\tilde{D}$ is measured and use as utility metric. The error is computed as:

$$error(Q(\tilde{D})) = \frac{|Q(\tilde{D}) - Q(D)|}{max(Q(D), s)}$$

where s is sanity bound, allowing to mitigate the effect of queries with small selectivity. In practice, we set s to the standard value of 0.1% of $|D|$. Usual evaluations generate random queries with a variable maximum length of the sub-string. Each event in a query is uniformly selected at random from the event universe, finally the average error is reported.

We follow this method and generate 100000 random queries from various maximum length ($max_{length} \in \{2, 4, 8, 12, 16, 20\}$). In our case, due to the large number of events in our real data-set, the queries generated have usually no occurrence in the real data-set ($Q(D) = 0$). That $\forall Q, Q(D) = Q(\tilde{D}) = 0$ is therefore an average error of 0, for any value of k, ϵ and max_{length}. This demonstrates that in average a sub-string (i.e. query) not observed in D is also not observed in the generated data-set $\tilde{D}$. This is a positive result which shows that the model only generates sequences containing sub-strings present in the starting data. This is expected for sub-string with length lower than k by construction of k-testables but remains true experimentally for longer sub-strings.

Nevertheless, since queries are not observed in D, we cannot evaluate the ability of our method to generate sub-strings observed in D. We propose a different evaluation where queries are randomly drawn from existing sub-strings in D. The results are reported in Table 4. We observe that the value of ϵ does not negatively affect the ability of the model to handle query count. Furthermore, we obtain very good results for long sub-strings, arguing for the ability of the model to preserve their counts.

		max_{length}:	4	8	12	20
generated data	k=2	ϵ=0	0.38	0.23	0.16	0.11
		ϵ=0.25	0.39	0.23	0.17	0.11
		ϵ=0.5	0.39	0.23	0.17	0.11
		ϵ=0.75	0.40	0.24	0.17	0.11
	k=4	ϵ=0	0.37	0.22	0.16	0.11
		ϵ=0.25	0.39	0.23	0.17	0.11
		ϵ=0.5	0.39	0.23	0.17	0.11
		ϵ=0.75	0.40	0.24	0.17	0.11

Table 4: Average absolute error for sub-string observed in real data of length lower than max_{length}

Frequent Sequential Pattern Mining We also consider frequent sequential pattern mining for evaluation. We are interested in the top $N = \mathcal{N}$ most frequent sub-strings from D where $\mathcal{N}$ is an integer. These patterns are computed using the PrefixSpan algorithm (Pei et al., 2001). We report the true positive rate: the percentage of frequent patterns correctly identified. by denoting $\mathcal{F}_{\mathcal{N}}(D)$(respectively $\mathcal{F}_{\mathcal{N}}(\tilde{D})$) the top $\mathcal{N}$ most frequent pattern in D (respectively $\tilde{D}$), we compute the true positive rate as:

$$TPR = \frac{|\mathcal{F}_{\mathcal{N}}(D) - \mathcal{F}_{\mathcal{N}}(\tilde{D})|}{\mathcal{N}}$$

Results for different value of $\mathcal{N}$ are reported. We observe in Table 5 that, even without pruning, between 40% and 60% of the most frequent patterns appear in the data-set generated by the Probabilistic Timed k-Testable Automata. These values can be used as baseline to evaluate the effect of the pruning. As expected when ϵ increases the number of patterns found decreases, it is the trade-off between utility and privacy. We denote an exception to this rule for $\epsilon = 0.25$ and $\epsilon = 0.5$, this main be explain by the fact that the two PTA are close since there are very few samples removed between $\epsilon = 0.25$ and $\epsilon = 0.5$.

			$\mathcal{N}$=20	$\mathcal{N}$=40	$\mathcal{N}$=60	$\mathcal{N}$=80	$\mathcal{N}$=100
generated data	k=2	ϵ=0	0.4	0.37	0.48	0.43	0.49
		ϵ=0.25	0.15	0.22	0.23	0.26	0.26
		ϵ=0.5	**0.35**	**0.27**	**0.31**	**0.27**	**0.28**
		ϵ=0.75	0.2	0.2	0.25	0.25	0.25
	k=4	ϵ=0	0.45	0.45	0.48	0.52	0.58
		ϵ=0.25	0.25	**0.35**	**0.41**	**0.45**	**0.46**
		ϵ=0.5	**0.35**	0.27	0.30	0.26	0.27
		ϵ=0.75	0.1	0.15	0.23	0.2	0.19

Table 5: True positive rate on top$\mathcal{N}$ most frequent patterns

Discussion As we have seen, the small number of accesses for a large number of resources as well as the overall size of the universe make the data set difficult to make private. For this reason, we observe that the pruning method we propose modifies the initial distribution in favour of the shortest sequences. Nevertheless, the results on the two evaluation tasks show that essential features of the initial distribution such as the most frequent patterns are preserved in the generated version.

In the PTk-TA, the duration are encoded with normal distributions. One may ask the question, is this encoding justified in practice? Intuitively, the duration could be very different from a user to another for a given page, so does this variable follow a normal distribution? Practically, in X5gon little additional information are collected outside of the pages viewed, in particular the only demographic data collected is the country of origin of the user as reported in the web browser. For this reason, the most important factors found to determine the time spent on a resource are not demographic factors. Instead, other factors such as the date of the learning session (day of the week, time of day...), or indirect information about the user such as the average duration of these learning sessions, or the time of consultation of the resource in a session (beginning or end of session) are typically factors that can influence these duration. The argument that pushes us to use a normal distribution is the presence of numerous mechanisms in the literature allowing to make the mean and the variance differentially private. Of course the use of a finer method of modelling duration taking these different factors into account may better model the duration distribution but it is also much more dangerous in terms of privacy. A such modelling of duration, with privacy guarantee and better utility remains an open question in the literature.

5. Related works

Several authors also develop algorithms to make public sanitized sequential data under differential privacy. Chen et al. (2012b,a) propose two algorithms: the first is based on a noisy prefix tree that combines sequences with identical prefixes for the same branch (Chen et al., 2012b), the second uses a variable n-gram model (Chen et al., 2012a), which can extract the essential information of the data-sets to adjust the scales of injected noise. We can also mention the PrivTree technique (Zhang et al., 2016) which implements a representation of a variable length Markov chain model to overcome the need to set some predefined threshold for the recursion depth, as required by the previously mentioned techniques.

More recently, deep learning based approach has been developed, Abay et al. (2019) use an Auto-encoder neural architecture for learning latent structure of subgroups of data and uses expectation maximization algorithm to simulate the groups. Two other methods focus on Generative Adversarial Network architecture, the former SeqGAN Yu et al. (2017) is based on a reinforcement learning training where the reward signal is produced by the discriminator. The second Frigerio et al. (2019) uses dp-GAN model with Long Short Term Memories (LSTM) unit inside the generator to model sequential data.

The above-mentioned approaches are often only interested in sequential data without taking into account the duration of the events. Moreover, approaches respecting differential privacy often entail a low utility in the generated data for this reason: Shaked and Rokach (2020) presents differential privacy as a constraint that is too restrictive. And they propose in their paper a new framework of privacy for sequential data. Finally, the approaches proposed above do not formalize the problem in the field of grammatical inference. One other approach, Jacquemont et al. (2009) use a grammatical inference based approach to study the flow of cars, nevertheless this paper focuses more on a case study than on a generic method and does not use modern privacy guarantee.

6. Conclusion

In this paper, we introduce a new type of finite state machines, both timed and probabilistic. Learning these machines from set of user navigation logs is possible through k-testable automata learning techniques, adapted to take into account durations and probabilities. We show how this automaton is able to efficiently generate an arbitrarily large data-set with statistical characteristics close to the initial data-set. We use ideas from the differential privacy research area to analyze the obtained automaton and remove data-points which would typically have a too big importance and could therefore be identified from the automaton or the newly generated data. We also prove some properties of this algorithm. Finally, we conduct a case study on a large data-set of log interactions. The results show that even in the context of a very large universe of items, the aforementioned method is still able to identify the most frequent patterns in the data-set.

In perspective we are interested in applying our method to classical benchmarks from the literature, but also in applying classical methods from the literature to the X5gon data-set.

References

Nazmiye Ceren Abay, Yan Zhou, Murat Kantarcioglu, Bhavani Thuraisingham, and Latanya Sweeney. Privacy preserving synthetic data release using deep learning. In Michele Berlingerio, Francesco Bonchi, Thomas Gärtner, Neil Hurley, and Georgiana Ifrim, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 510–526. Springer International Publishing, 2019.

Alket Cecaj, Marco Mamei, and Franco Zambonelli. Re-identification and information fusion between anonymized CDR and social network data. *J. Ambient Intell. Humaniz. Comput.*, 7(1):83–96, 2016.

Rui Chen, Gergely Acs, and Claude Castelluccia. Differentially private sequential data publication via variable-length n-grams. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 638–649, 2012a.

Rui Chen, Benjamin CM Fung, Bipin C Desai, and Nériah M Sossou. Differentially private transit data publication: a case study on the montreal transportation system. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 213–221, 2012b.

Colin de la Higuera. *Grammatical inference: learning automata and grammars*. Cambridge University Press, 2010.

Hendrik Drachsler, Katrien Verbert, Olga C Santos, and Nikos Manouselis. Panorama of recommender systems to support learning. In *Recommender systems handbook*, pages 421–451. Springer, 2015.

Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography*, pages 265–284. Springer Berlin Heidelberg, 2006.

Lorenzo Frigerio, Anderson Santana de Oliveira, Laurent Gomez, and Patrick Duverger. Differentially private generative adversarial networks for time series, continuous, and discrete open data. In *ICT Systems Security and Privacy Protection*, pages 151–164. Springer International Publishing, 2019.

Pedro Garcia, Enrique Vidal, and José Oncina. Learning locally testable languages in the strict sense. In *ALT*, pages 325–338, 1990.

Stéphanie Jacquemont, François Jacquenet, and Marc Sebban. Discovering Patterns in Flows: a Privacy Preserving Approach with the ACSM Prototype. In *ECML PKDD*, volume 5782 of *Lecture Notes in Computer Science*, pages 734–737. Springer, 2009.

Slawomir Lasota and Igor Walukiewicz. Alternating timed automata. *ACM Transactions on Computational Logic (TOCL)*, 9(2):1–27, 2008.

Jonathan Mayer, Patrick Mutchler, and John C. Mitchell. Evaluating the privacy properties of telephone metadata. *Proceedings of the National Academy of Sciences*, 113(20):5536–5541, 2016.

Robert McNaughton and Seymour A. Papert. *Counter-Free Automata (M.I.T. Research Monograph No. 65).* The MIT Press, 1971.

Frank McSherry and Kunal Talwar. Mechanism design via differential privacy. In *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '07, page 94–103. IEEE Computer Society, 2007.

J. Pei, Jiawei Han, B. Mortazavi-Asl, Helen Pinto, Q. Chen, U. Dayal, and M. Hsu. Prefixspan,: mining sequential patterns efficiently by prefix-projected pattern growth. *Proceedings 17th International Conference on Data Engineering*, pages 215–224, 2001.

Christopher Riederer, Yunsung Kim, Augustin Chaintreau, Nitish Korula, and Silvio Lattanzi. Linking users across domains with location data: Theory and validation. WWW '16. International World Wide Web Conferences Steering Committee, 2016.

Sigal Shaked and Lior Rokach. Privgen: Preserving privacy of sequences through data generation. *arXiv preprint arXiv:2002.09834*, 2020.

Xinyu Yang, Teng Wang, Xuebin Ren, and Wei Yu. Survey on improving data utility in differentially private sequential data publishing. *IEEE Transactions on Big Data*, PP: 1–1, 06 2017.

Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, page 2852–2858. AAAI Press, 2017.

Jun Zhang, Xiaokui Xiao, and Xing Xie. Privtree: A differentially private algorithm for hierarchical decompositions. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, page 155–170, New York, NY, USA, 2016. Association for Computing Machinery.