Differential Privacy - II

Prof. Dan A. Simovici

UMB

Rui Chen and colleagues studied the transportation traffic of *Société de transport de Montrèal* or **STM**, the public transit system of the city of Montreal in Quebec, Canada.
The goals of their analysis were:

1. to protect individual privacy, and
2. to preserve utility in sanitized data for data analysis.

The STM System Characteristics

- Since 2008 STM uses a smart card automated fare collection (SCAFC) system to ensure user validation and fare collection.
- System collects daily passengers' transit data which must be shared (after being anonymized):
  - administrative regulations;
  - profit sharing;
  - data analysis.
- SCAFC is integrated with other transit networks in adjacent cities.

- transit information (smart card number and station ID) is collected and stored in a central database system;
- the transit info of a passenger is organized as a sequence of stations in time order;
- the nature of transit data is raising major privacy concerns on the part of card users.

Challenges of Exploring Transit Data

- the are close to 1,000 bus and subway stations in the STM system;
- if the maximum length of a sequence of stations is 20, a database $\mathcal{D}$ would need to contain about $\sum_{i=1}^{20} 1000^i$ records which is about 10^{60} entries, which is impossible.

System is data driven: only sequences and counts generated by real sequences are stored.

Data mining tasks:

- answering count queries, and
- frequent sequential pattern mining.

With accurate answers to count queries, data recipients can answer questions, such as how many passengers have visited both stations Park Station and Government Center".
Frequent sequential pattern mining helps the STM better understand passengers transit patterns and consequently allows the STM to adjust its network geometry and schedules in order to better utilize its existing resources.

Let $\mathcal{L} = \{L_1, L_2, \ldots, L_{|C|}\}$ be the universe of locations.
Locations are considered as discrete spatial area on a map.
For the STM transit data, $\mathcal{L}$ represent all stations in the STM network.
Each record on a sequential database consists of a sequence of time-ordered locations drawn from $\mathcal{L}$.

Definition

A sequence of locations is an ordered list

$$S = L_1 \to L_2 \to \cdots \to L_n,$$

where n is the length of the sequence (also denoted by $|S|$).

A location may occur multiple time in S and may occur consecutively in S.

Definition

A *sequential database* D of size $|D|$ is a multiset of sequences

$$D = \{S_1, S_2, \ldots, S_{|D|}\}.$$

Each sequence represents the movement history of a record owner.

Example

A sample sequential database is shown below:

Rec.#	Path
1	$L_1 \rightarrow L_2 \rightarrow L_3$
2	$L_1 \rightarrow L_2$
3	$L_3 \rightarrow L_2 \rightarrow L_1$
4	$L_1 \rightarrow L_2 \rightarrow L_4$
5	$L_1 \rightarrow L_2 \rightarrow L_3$
6	$L_3 \rightarrow L_2$
7	$L_1 \rightarrow L_2 \rightarrow L_4 \rightarrow L_1$
8	$L_3 \rightarrow L_1$

A prefix tree groups sequences with the same prefix into the same branch.
A sequence

$$S' = L'_1 \to L'_2 \to \cdots \to L'_m$$

is a prefix of a sequence

$$S = L_1 \to L_2 \to \cdots \to L_n$$

if $m \leqslant n$ and $L'_i = L_i$ for $1 \leqslant i \leqslant m$.
This is denoted by $S' \preceq S$.

Example

$S' = L_1 \to L_2$ is a prefix of $S = L_1 \to L_2 \to L_3$; $S'' = L_1 \to L_4$ is not a prefix of S.

Definition

A prefix tree of a sequential database $\mathcal{D}$ is a triplet $\mathcal{PT} = (V, E, \text{Root})$, where

- V is a set of nodes labelled with locations, each corresponding to a unique prefix in $\mathcal{D}$;
- E is the set of edges, representing transitions between nodes;
- $\text{Root} \in V$ is the virtual root of $\mathcal{PT}$.

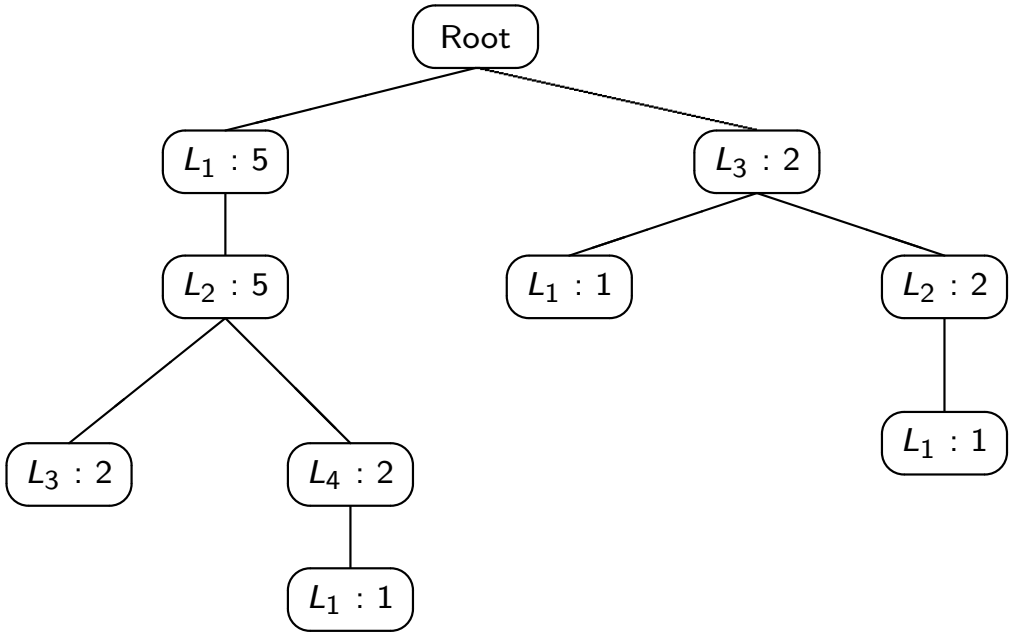
The unique prefix represented by a node $v \in V$, denoted by $\text{prefix}(v, \mathcal{PT})$ is a sequence of locations starting from Root to v.

Each node $v \in V$ keeps a pair $(\mathrm{tr}(v), c(v))$, where

- $\mathrm{tr}(v)$ is the set of sequences in $\mathcal{D}$ having $\mathrm{prefix}(v, \mathcal{T})$, that is, $\{S \in \mathcal{D} \mid \mathrm{prefix}(v, \mathcal{T}) \preceq S\}$, and
- $c(v)$ is a noisy version of $|\mathrm{tr}(v)|$ (e.g. $|\mathrm{tr}(v)| + X$, where X is Laplace noise.
- $\mathrm{tr}(\mathrm{Root})$ contains all sequence in $\mathcal{D}$.

The set of all nodes at a depth i is a level of $\mathcal{PT}$ and is denoted by $\mathrm{level}(i, \mathcal{T})$.
Root is at level 0.

Example



Rec.#	Path
1	$L_1 \rightarrow L_2 \rightarrow L_3$
2	$L_1 \rightarrow L_2$
3	$L_3 \rightarrow L_2 \rightarrow L_1$
4	$L_1 \rightarrow L_2 \rightarrow L_4$
5	$L_1 \rightarrow L_2 \rightarrow L_3$
6	$L_3 \rightarrow L_2$
7	$L_1 \rightarrow L_2 \rightarrow L_4 \rightarrow L_1$
8	$L_3 \rightarrow L_1$

Definition

Let $\mathcal{L}$ be a set of locations. A count query $Q_{\mathcal{L}}$ over a database $\mathcal{D}$ is defined as

$$Q_{\mathcal{L}}(\mathcal{D}) = |\{S \in \mathcal{D} \mid \mathcal{L} \subseteq \mathrm{loc}(S)\}|,$$

where $\mathrm{loc}(S)$ is the set of locations in S.

The order of locations is not considered in count queries, because the major users of count queries are the personnel of the marketing department of the STM, who are merely interested in users' presence in certain stations for marketing analysis, known as *passenger counting*, but not the order of visiting.

Utility of a Count Query

The utility of a count query $Q_{\mathcal{L}}$ over a sanitized database $\tilde{\mathcal{D}}$ is the relative error with respect to the original database $\mathcal{D}$:

$$err(Q_{\mathcal{L}}(\tilde{\mathcal{D}})) = \frac{Q_{\mathcal{L}}(\tilde{\mathcal{D}}) - Q_{\mathcal{L}}(\mathcal{D})}{\max\{Q_{\mathcal{L}}(\mathcal{D}), s\}},$$

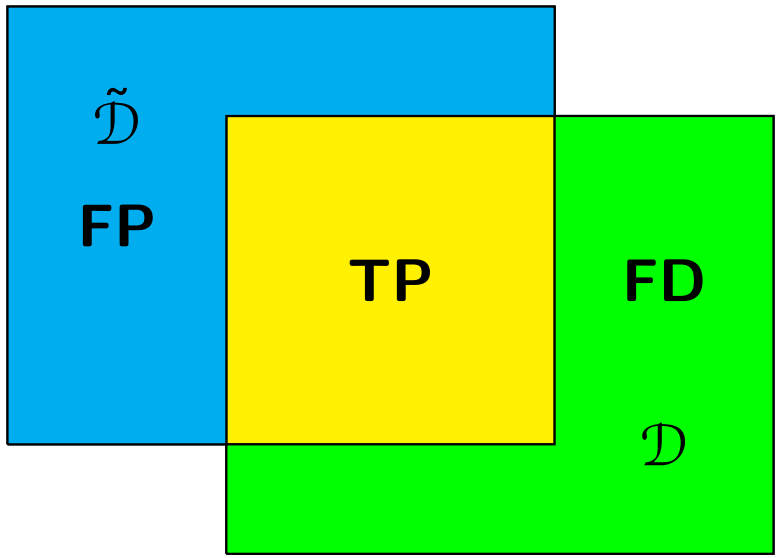
where s is a sanity bound that mitigates the influence of queries with very small selectivities.

Utility for Frequent Sequential Pattern Mining

For $k \in \mathbb{N}$, $k \geqslant 2$ the set of top k most frequent patterns in $\mathcal{D}$ is denoted by $\mathcal{F}_k(\mathcal{D})$; the corresponding set of the sanitized database $\mathcal{D}$ is $\mathcal{F}_k(\tilde{\mathcal{D}})$.

- The true positive rate is the number of frequent sequential patterns in $\mathcal{F}_k(\mathcal{D})$ that are correctly identified in $\mathcal{F}_k(\tilde{D})$, that is, $|\mathcal{F}_k(\mathcal{D}) \cap \mathcal{F}_k(\tilde{\mathcal{D}})|$.
- The false positive rate is the number of infrequent sequential patterns in $\mathcal{D}$ that are mistakenly included in $\mathcal{F}_k(\tilde{\mathcal{D}})$, that is, $|\mathcal{F}_k(\tilde{\mathcal{D}}) - \mathcal{F}_k(\mathcal{D})|$.
- The false drop rate is the number of frequent sequential patterns in $\mathcal{F}_k(\mathcal{D})$ that are wrongly omitted in $\mathcal{F}_k(\tilde{\mathcal{D}})$, that is, $|\mathcal{F}_F(\mathcal{D}) - \mathcal{F}_k(\tilde{D})|$.

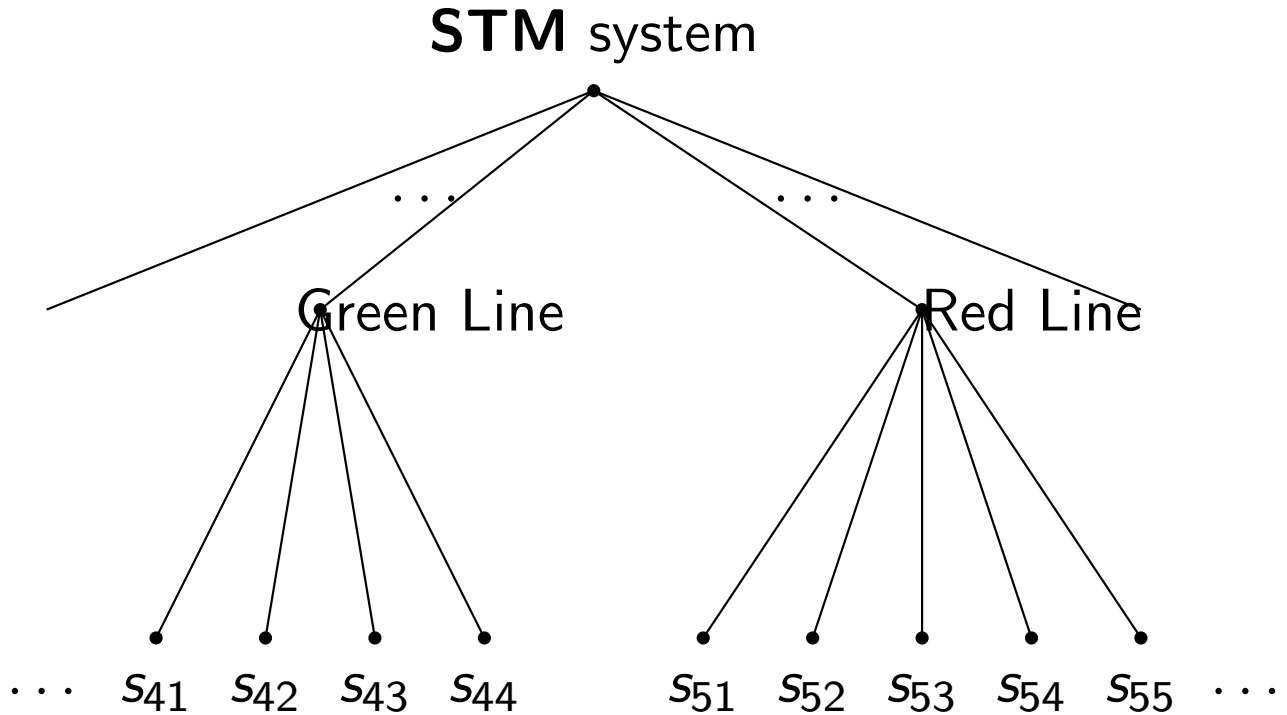
Since in our setting $|\mathcal{F}_k(\mathcal{D})| = |\mathcal{F}_k(\tilde{\mathcal{D}})| = k$, false positives always equal false drops.



TP : true positive, **FP**: false positive, **FD** : false drop

The taxonomy tree of STM

The taxonomy tree is a two-level tree $\mathcal{T}$ that specifies the strucure of the transportation network:



Input data:

raw data set $\mathcal{D}$;

privacy budget ϵ;

a user-specified height of prefix tree h;

a location taxonomy tree $\mathcal{T}$.

Output data:

a sanitized data set $\tilde{\mathcal{D}}$ satisfying ϵ-differential privacy.

Components of the algorithm:

BuildNoisyPrefixTree($\mathcal{D}, \epsilon, h, \mathcal{T}$);
GeneratePrivateRelease($\mathcal{PT}$);

- BuildNoisyPrefixTree($\mathcal{D}, \epsilon, h, \mathcal{T}$) constructs a noisy hybrid-granularity prefix tree $\mathcal{PT}$ of $\mathcal{D}$ using a set of count queries based on a given taxonomy tree $\mathcal{T}$.
- In STM case, $\mathcal{T}$ has two level: each station can be generalized to a metro or a bus line on which it is located.
- GeneratePrivateRelease($\mathcal{PT}$) uses utility-boosting techniques on $\mathcal{PT}$ based on two sets of consistency constraints, and then generates a differentially private release $\tilde{\mathcal{D}}$.

The Algorithm

Input: Raw sequential dataset $\mathcal{D}$;
Privacy budget ϵ;
Height of the prefix tree h;
Location taxonomy tree $\mathcal{T}$;

Output: Sanitized data set $\tilde{\mathcal{D}}$;

1. Noisy prefix tree $\mathcal{PT} \leftarrow$ BuildNoisyPrefixTree$(\mathcal{D}, \epsilon, h, \mathcal{T})$;
2. Sanitized data set $\tilde{\mathcal{D}} \leftarrow$ GeneratePrivateRelease$(\mathcal{PT})$;
3. **return** $\tilde{\mathcal{D}}$.

Noisy Prefix Tree Construction

Group recursively sequences in $\mathcal{D}$ into disjoint sub-databases based on their prefixes.

We begin with

Line no.	Code
1.	$i = 0$;
2.	Create a prefix tree $\mathcal{PT}$ with root *Root*;
3.	Add all sequence in $\mathcal{D}$ to tr(*Root*);

Noisy Prefix Tree Construction

In constructing $\mathcal{PT}$ we employ a uniform privacy budget allocation scheme, by dividing the total privacy budget ϵ into equal portions $\bar{\epsilon} = \frac{\epsilon}{h}$. Each portion is used for constructing a level of $\mathcal{PT}$.

4. $\bar{\epsilon} = \frac{\epsilon}{h}$;

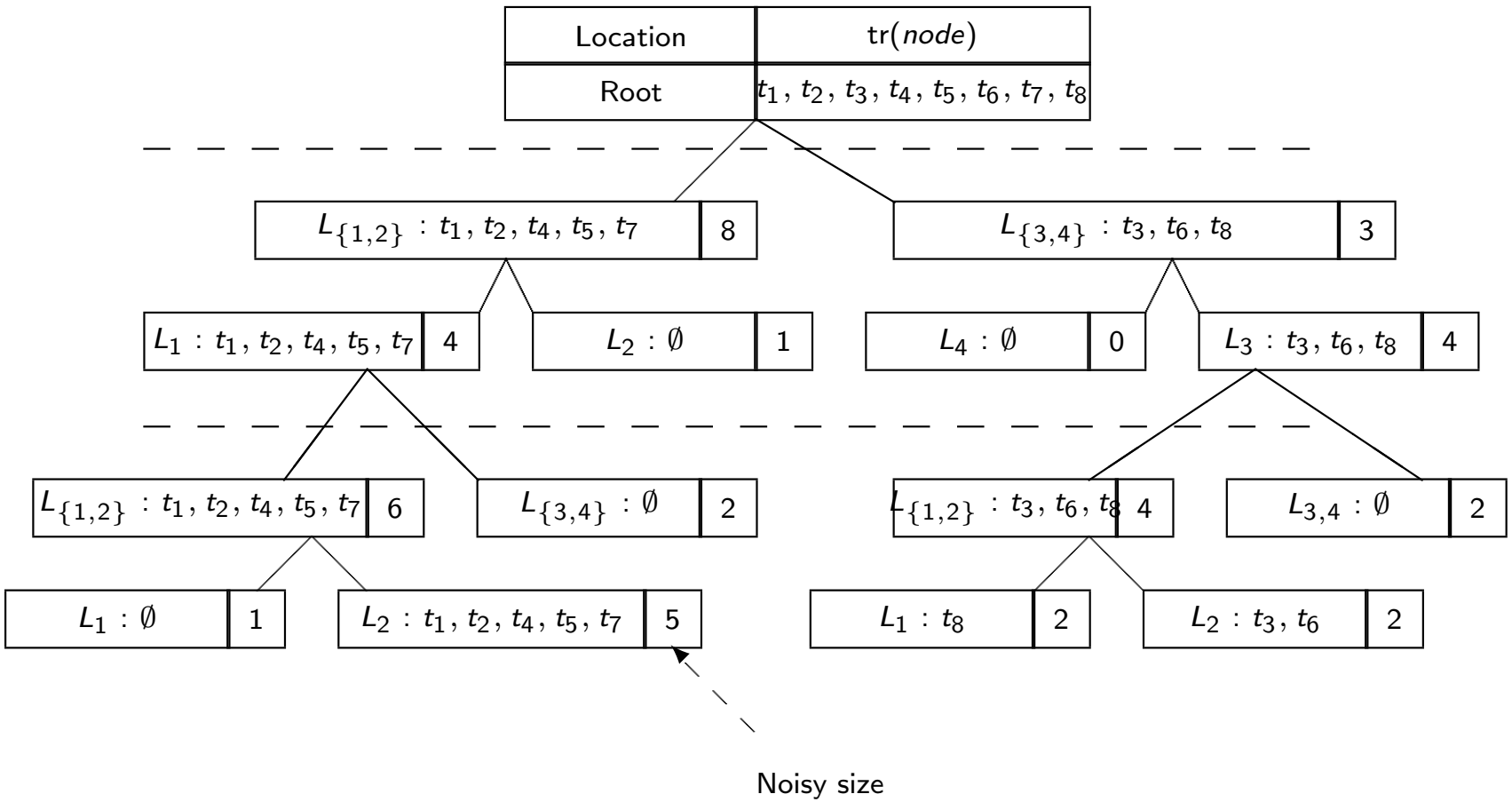
- To satisfy differential privacy every sequence that can be derived from the location universe (either in $\mathcal{D}$ or not) has a non-zero probability to appear in the noisy prefix tree. Therefore, at each level, for each node, we need to consider every possible location as a potential child.
- We need to identify children associated to non-empty nodes.

Each level of $\mathcal{PT}$ is divided into two sub-levels:

1. the first sub-level consists of nodes associated with generalized location information (generalized nodes);
2. depending on the noisy counts of generalized noise, we decide whether to further expand them to create a second sub-level in which nodes are associated with non-generalized locations (e.g., ask the noisy count of passengers in a metro line and then decide whether to ask the count of each station on that line).

- $\bar{\epsilon}$ is allocated to the two levels as a function of the fan-out of the location taxonomy tree $\mathcal{T}$; the first level receives $\bar{\epsilon}_1 = 2\frac{\bar{\epsilon}}{f}$ and the second $\bar{\epsilon}_2 = \frac{(f-2)\bar{\epsilon}}{f}$.
- All nodes on the same sub-level are associated with disjoint sequence subsets, so the privacy budget allocated to a sub-level can be used in full for each node.

5. calculate $\bar{\epsilon}_1$ and $\bar{\epsilon}_2$ such that $\bar{\epsilon}_1 + \bar{\epsilon}_2 = \bar{\epsilon}$;


Location
tr(node)
Root
$t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8$
$L_{\{1,2\}} : t_1, t_2, t_4, t_5, t_7$
8
$L_{\{3,4\}} : t_3, t_6, t_8$
3
$L_1 : t_1, t_2, t_4, t_5, t_7$
4
$L_2 : \emptyset$
1
$L_4 : \emptyset$
0
$L_3 : t_3, t_6, t_8$
4
$L_{\{1,2\}} : t_1, t_2, t_4, t_5, t_7$
6
$L_{\{3,4\}} : \emptyset$
2
$L_{\{1,2\}} : t_3, t_6, t_8$
4
$L_{3,4} : \emptyset$
2
$L_1 : \emptyset$
1
$L_2 : t_1, t_2, t_4, t_5, t_7$
5
$L_1 : t_8$
2
$L_2 : t_3, t_6$
2
Noisy size

6. **while** $i < h$ **do**
7. **foreach** non-generalized node $v \in level(i)$ **do**
8. $\mathcal{U}_g \leftarrow$ the set of generalized nodes from $\mathcal{T}$;
9. **foreach** node $u \in \mathcal{U}_g$ **do**
10. add sequences S with $prefix(u) \preceq S$ to $\mathrm{tr}(u)$;
11. $c(u) = NoisyCount(|\mathrm{tr}(u)|, \bar{\epsilon}_1)$;
12. **if** $c(u) \geqslant \theta_g$ **then**
13. add u to $\mathcal{PT}$;
14. $\mathcal{U}_{ng} \leftarrow u$'s non-generalized children in $\mathcal{T}$;
15. **foreach** node $w \in \mathcal{U}_{ng}$ **do**
16. add sequences S with $prefix(w) \preceq S$ to $\mathrm{tr}(w)$;
17. $c(w) = NoisyCount(|\mathrm{tr}(w)|, \bar{\epsilon}_2)$;
18. **if** $c(w) \geqslant \theta_{ng}$ **then**
19. add w to $\mathcal{PT}$;
20. i++;
21. **return** $\mathcal{PT}$;

- θ is a given threshold used to decide if a node u is empty;
- $c(u)$ is the noisy count of the node u;
- For a potential non-empty node u, Laplace noise is added to $|\text{tr}(u)|$ and use the noisy answer $c(u)$ to decide if u is non-empty.

If $c(u) > \theta$, u is deemed non-empty and inserted into $\mathcal{PT}$.

For a non-generalized node $\theta_{ng} = 2\frac{\sqrt{2}}{\bar{\epsilon}_2}$ (two standard deviations of noise).

For a generalized node, $\theta_g = 4\frac{\sqrt{2}}{\bar{\epsilon}_1}$

Empty Nodes

For each empty node tests are run to determine if

$$NoisyCount(|\mathrm{tr}(w)|, \bar{\epsilon}) = NoisyCount(0, \bar{\epsilon}) > \theta,$$

where $\bar{\epsilon}$ is the privacy budget (either $\bar{\epsilon}_1$ or $\bar{\epsilon}_2$).

Example

For the database $\mathcal{D}$ we have $h = 2$ and $\theta = 3$. If L_1, L_2 can be generalized to $L_{1,2}$ L_3, L_4 can be generalized to $L_{3,4}$. The construction of a hybrid prefix tree $\mathcal{PT}$ was shown previously.

- The sanitized database $\tilde{\mathcal{D}}$ can be obtained by traversing $\mathcal{PT}$ one is post-order ignoring the generalized nodes.
- For each node the number n of sequences terminating at each non-generalized node v and appending n copies of prefix$(v, \mathcal{PT})$ to output.
- The addition of noise to the counts may create inconsistencies in the release.

Consistency Constraints in the prefix tree

1. For any root-to-leaf path p and for every node v_i on this path,

$$|\mathrm{tr}(v_i)| \leqslant |\mathrm{tr}(v_{i+1})|,$$

   where v_i is a child of v_{i+1};
2. For each node v,

$$|\mathrm{tr}(v)| \geqslant \sum_{u \in \mathrm{children}(v)} |\mathrm{tr}(u)|.$$