

Generation Matrix: An Embeddable Matrix Representation for Hierarchical Trees

Jianping Cai^a, Ximeng Liu^{a,*}, Jiayin Li^a, Shuangyue Zhang^b

^aCollege of Computer and Data Science, Fuzhou University, Fuzhou 350108, China

^bCollege of information and Smart Electromechanical Engineering, Xiamen Huaxia University, Xiamen 361024, China

Abstract

Starting from the local structures to study hierarchical trees is a common research method. However, the cumbersome analysis and description make the naive method challenging to adapt to the increasingly complex hierarchical tree problems. To improve the efficiency of hierarchical tree research, we propose an embeddable matrix representation for hierarchical trees, called Generation Matrix. It can transform the abstract hierarchical tree into a concrete matrix representation and then take the hierarchical tree as a whole to study, which dramatically reduces the complexity of research. Mathematical analysis shows that Generation Matrix can simulate various recursive algorithms without accessing local structures and provides a variety of interpretable matrix operations to support the research of hierarchical trees. Applying Generation Matrix to differential privacy hierarchical tree release, we propose a Generation Matrix-based optimally consistent release algorithm (GMC). It provides an exceptionally concise process description so that we can describe its core steps as a simple matrix expression rather than multiple complicated recursive processes like existing algorithms. Our experiments show that GMC takes only a few seconds to complete a release for large-scale datasets with more than 10 million nodes. The calculation efficiency is increased by up to 100 times compared with the state-of-the-art schemes.

Keywords: Generation Matrix, Matrix Representation, Hierarchical Tree, Differential Privacy, Consistency
2020 MSC: 05C62, 05C05

1. Introduction

As a fundamental data structure, hierarchical trees are widely used in different areas, including file systems [1], census [2], evolution [3], etc. For example, in the U.S. Census Bureau's plan to apply differential privacy to protect privacy[2], designing a novel hierarchical tree releasing algorithm is one of the important challenges[4]. The scale of the census data is so large that we can organize them into a hierarchical tree with more than 10 million nodes. Hence, the hierarchical tree release algorithms must be specially designed and highly efficient to ensure the timely release of such large-scale data. However, the design of efficient algorithms usually requires a large amount of hierarchical tree research as a theoretical basis.

Most hierarchical tree research works naturally regard the hierarchical tree as a collection of nodes and relationships. Starting from the perspective of individual nodes and local relationships to study hierarchical trees is a common research method, called Naive Research Method. Empirically, Naive Research Method often leads to overly cumbersome analysis and abstract algorithm descriptions, mainly reflected in two aspects. On the one hand, hierarchical trees contain rich relationships, such as father, son, ancestor, descendant, sibling, or cousin, but these relationships usually lack a concrete enough description. When multiple node relationships occur in an algorithm simultaneously, the intricate node relationships will make the algorithm challenging to understand. On the other hand, Naive Research

^{*}This work was supported in part by the National Natural Science Foundation of China (project numbers 62072109 and U1804263).

^{*}Corresponding author.

Email addresses: jpingcai@163.com (Jianping Cai), nbnix@qq.com (Ximeng Liu), lijiaiyin2019@gmail.com (Jiayin Li), zhangsy@hxxxy.edu.cn (Shuangyue Zhang)

Method focuses on the local structure of hierarchical trees rather than the overall structure. The local structure is part of the overall structure, but the studies falling into local structures may prevent researchers from solving problems from a macro perspective. Worse, the additional auxiliary symbols or indexes for describing relationships and local structures pose a significant challenge to the researchers' data analysis capabilities. Imagine a researcher facing a half-page expression with various randomly labeled symbols and subscripts; how should the next step go? Therefore, Naive Research Method is too cumbersome to solve the increasingly complex hierarchical tree problems effectively.

Considering the complexity of Naive Research Method, we adopted a Matrixing Research Method for hierarchical tree problems. Its core idea is to transform the hierarchical tree into a specific matrix representation and then embed it into the research works or algorithm designs, making the initially abstract and indescribable hierarchical tree concrete and analyzable. As a more advanced research method, similar ideas widely exist in many fields such as graph theory [5, 6, 7, 8], group theory [9], and deep learning[10, 11]. Unlike Naive Research Method, the research object of Matrixing Research Method is the hierarchical tree itself rather than the local structure. It emphasizes avoiding visiting individual nodes as much as possible but implementing operations of the hierarchical tree by the matrix representation. Therefore, the matrix representation design is critical and directly determines whether the research can proceed smoothly. The challenges of matrix representation design are as follows.

- 1) A non-negligible problem is the universality of recursion in hierarchical tree algorithms, while the access of individual nodes and the description of local relationships are almost inevitable in recursion. It violates the core idea of the Matrixing Research Method. Some matrix representations [5, 6, 7, 8] have been used in the spectral theory of trees, but there is no achievement to show that the existing matrix representations can implement recursion without accessing local structures. So that whether supporting recursion is critical to the matrix representation.
- 2) We hope that the matrix representation can directly serve algorithm designs, not just a theoretical analysis tool. Therefore, the matrix representation should be succinct to ensure the efficiency of algorithms. Specifically, the space overhead of each hierarchical tree node should be constant rather than the dense matrices like Distance Matrix [7] or Ancestral Matrix [8].

1.1. Our contributions

Considering the challenges above, we propose an embeddable matrix representation called Generation Matrix. Generation Matrix is a lower triangular matrix containing only $2n - 1$ non-zero elements (i.e., the weights of nodes and edges). Applying sparse storage technologies[12], we only need to store non-zero elements, satisfying the succinctness. Compared with others [5, 6, 7, 8], Generation Matrix emphasizes the application in the hierarchical tree algorithms. Our analysis of properties shows that many calculations on Generation Matrix have specific mathematical meanings. We can explain them and combine them to design complex hierarchical tree algorithms. More importantly, we demonstrate that the inverse of the Generation Matrix contains the inherent logic of recursion. Therefore, we can use Generation Matrix to simulate the top-down and bottom-up recursions without accessing the local structures. Besides, we study the relationship between Generation Matrix and some existing matrix representations and find it can be easily converted to others. It implies that Generation Matrix can be combined with the theories from other matrix representations to solve hierarchical tree problems.

To demonstrate the practicability of Generation Matrix, we introduce an application on the differentially private hierarchical tree release above. Considering the consistency problem[13, 14], we design a Generation Matrix-based optimally consistent release algorithm (GMC) for differentially private hierarchical trees. To our knowledge, GMC is the first solution to the problem by using matrix theory. It has an exceptionally concise process description so that just a simple matrix expression can summarize the core process. GMC embodies the advantages of Generation Matrix in solving problems in cross-domain. Therefore, Generation Matrix has positive significance for promoting the development of hierarchical tree-related research and applying matrix theories to solve hierarchical tree problems.

1.2. Organization of paper

The rest of the paper is organized as follows. Section 2 reviews the related works of existing hierarchical tree representations and differentially private hierarchical tree release. Section 3 introduces the preliminaries of hierarchical trees and the optimally consistent release of differentially private hierarchical trees. Section 4 defines Generation

Matrix, then analyzes its mathematical properties and the conversion relationship with other matrix representations. In Section 5, we show the application of Generation Matrix on differentially private hierarchical tree release and design GMC. Finally, Section 6 compares GMC with the existing technology through experiments and demonstrates its efficiency.

2. Related Works

The research on the hierarchical tree representations mainly concentrates on data structure and graph theory fields. In the data structure field, researchers have achieved better performance in storage [15], query [16], or structure updating [17]. However, these representations are mainly for storage in computers but do not support mathematical analysis. We can not symbolize them and use them as tools for hierarchical tree researches. In the graph theory field, many works adopt matrix representations to represent trees, including Adjacency Matrix [5], Laplacian Matrix [3, 6], Distance Matrix [7], etc. Among them, Adjacency Matrix only describes the edge information, so that it is challenging to undertake the complex model analysis. Laplacian matrix and Distance matrix are two meaningful matrix representations widely used in spectral graph theory. However, they are for undirected graphs or trees but not suitable for representing rooted hierarchical trees. To summarize, none of the three matrix representations is the best choice for representing hierarchical trees. Subsequently, Eric et al. [8] proposed a new matrix representation for rooted trees (i.e., hierarchical trees), named Ancestor Matrix. It represents the structure of a hierarchical tree by describing the number of overlapping edges on the path from any two leaves to the root. Studying Ancestral Matrix, Eric et al. [8] obtained many essential conclusions, such as the maximum spectrum radius and the determinants of the characteristic polynomial. However, it is also not the best choice for the calculations of hierarchical trees. First, the dense Ancestral Matrix is not succinct enough. Secondly, Ancestral Matrix is a kind of matrix with a high degree of feature summary. Although it can deterministically express the structure of a hierarchical tree only by describing the leaves, it is very unintuitive and cannot simulate the operations of hierarchical trees. Therefore, the existing matrix representations are not suitable for the analysis and calculation of hierarchical tree models. Significantly, the broad application of the Laplacian matrix in deep learning [10, 11] in recent years implies that matrix representation has essential value for solving complex scientific problems. It motivates us to design a new matrix representation to solve hierarchical tree problems and design algorithms.

Differentially private hierarchical tree release is a data releasing technology that organizes the data into a hierarchical tree and applies differential privacy (DP) [18] to protect individual privacy. It is widely used in many scenarios, such as histogram publishing [13, 19], location privacy release based on spatial partitioning [20], trajectory data publishing [21], frequent term discovery [14]. By adding random noise to the data, DP provides a provable and quantifiable guarantee of individual privacy. However, the random noise will destroy the consistency that the hierarchical tree should satisfy, i.e., “the sum of the children’s values equals the value at the parent” [13]. Therefore, ensuring that the released results meet consistency and obtain a higher accuracy is one of the leading research goals. Hay et al. [13] first applied a hierarchical tree to improve the accuracy of range query and designed Boosting for the consistency of histogram release. However, Boosting can only support complete k -ary trees, which significantly limits its application. Moreover, Hay et al.’s error analysis [13] of the released results is rough, and only qualitative error results are obtained. Subsequently, Wahlbeh et al. analyzed the error of Boosting and designed an algorithm to calculate the error. However, it also can only support complete k -ary trees. In the differentially private frequent term discovery problem studied by Ning et al. [14], the hierarchical tree is arbitrary. Therefore, it is impossible to apply Boosting. For this reason, Ning et al. designed an optimally consistent release algorithm for arbitrary hierarchical trees in their proposed algorithms PrivTrie [14]. Its implementation is based on multiple complex recursions, which is not easy to understand and a large number of function calls result in significant additional computational overhead. Applying the idea of maximum likelihood estimation, Lee et al. [22] proposed a general solution for differentially private optimally consistent release. It solves the optimally consistent release by establishing a quadratic programming equation and has a closed-form matrix expression. Theoretically, it can apply to arbitrary optimally consistent release, but the computational overhead is so significant that it can only be processed for small-scale releases. However, Lee et al.’s research work [22] is inspiring. It motivates us to try to analyze the issues of differentially private hierarchical tree release from matrix analysis.

Under the representation of Generation Matrix, we can introduce many matrix analysis methods to solve hierarchical tree problems. One of them is QR decomposition [23]. In QR decomposition, we can transform any matrix into

a triangular matrix by orthogonal transformation. Compared with the original form, the triangular matrix is simpler, has many exciting properties [24]. The orthogonal transformation methods commonly used for QR decomposition include Householder transformation [25], Gram-Schmidt orthogonalization [26], Givens rotation [27], etc. Among them, Householder transformation is the simplest and more suitable for sparse matrices.

3. Preliminaries

In this section, we describe some preliminaries of our work. Before formally describing, we first introduce some of the main notation definitions shown in Tab 1.

Table 1: Notations Descriptions in Our Work

Notations	Descriptions
\mathcal{T}	Hierarchical tree with arbitrary structure
$\mathcal{T}^{(k)}$	k -order subtree of \mathcal{T}
f_i, C_i	Parent of node i ; the set of children of node i
n, n_k	The number of nodes of the hierarchical tree; the k -order subtree
h, h_i	The height of the hierarchy tree; The height of node i
m	The number of unit counts
$\mathbf{G}_{\mathcal{T}}^{(\mathbf{w}_{node}, \mathbf{w}_{edge})} \in \mathbb{R}^{n \times m}$	The Generation Matrix defined by a \mathcal{T} with the node weights \mathbf{w}_{node} and the edge weights \mathbf{w}_{edge}
$\mathbf{G}_{\mathcal{T}} \in \mathbb{R}^{n \times n}$	The structure matrix of \mathcal{T}
$\mathbf{M}_{\mathcal{T}} \in \mathbb{R}^{n \times m}$	The consistency constraint matrix of \mathcal{T}
$\mathbf{G}_{\mathcal{T}^{(1)} \leftarrow \mathcal{T}} \in \mathbb{R}^{n_1 \times n_1}$	The Generation Matrix inner-product equivalent to $\mathbf{M}_{\mathcal{T}}$
$\mathbf{A}_{\mathcal{T}}, \mathbf{L}_{\mathcal{T}}, \mathbf{D}_{\mathcal{T}} \in \mathbb{R}^{n \times n}, \mathbf{C}_{\mathcal{T}} \in \mathbb{R}^{m \times m}$	The Adjacency Matrix, Laplacian Matrix, Distance Matrix and Ancestral Matrix of \mathcal{T}
$\mathbf{x} \in \mathbb{R}^{m \times 1}$	The vector composed of unit counts x_i
$\mathbf{v} \in \mathbb{R}^{n \times 1}$	The vector composed of the values of nodes of the hierarchical tree arranged in order
$\bar{\mathbf{v}} \in \mathbb{R}^{n \times 1}, \bar{\mathbf{x}} \in \mathbb{R}^{m \times 1}$	The noisy \mathbf{v} and \mathbf{x} satisfying DP
$\hat{\mathbf{v}} \in \mathbb{R}^{n \times 1}, \hat{\mathbf{x}} \in \mathbb{R}^{m \times 1}$	Optimally consistent release after post-processing and the vector restored from $\bar{\mathbf{v}}$
$\mathbf{H}_h \in \{0, 1\}^{m \times n}$	The mapping matrix representing the mapping relationship between x_i and v_i

3.1. Hierarchical Tree

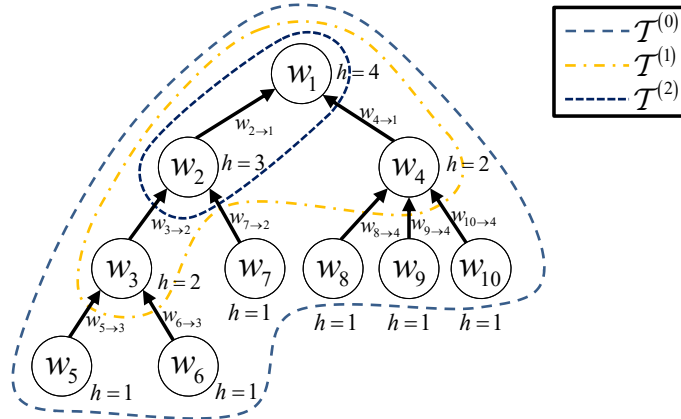


Figure 1: Hierarchical Tree and Its k -Order Subtree

We first recall the definition of the hierarchical tree.

Definition 1 (Hierarchical Tree[28]). *The hierarchical tree \mathcal{T} is a collection of nodes numbered $1, 2, \dots, n$, which satisfies*

- 1) \mathcal{T} contains a specially designated node called the root.
- 2) The remaining nodes are divided into several non-empty collections called the subtrees of the root.

In the hierarchical tree, we denote the relationships between the nodes as $i \rightarrow j$, indicating node j is the parent of node i . Fig. 1 shows a weighted hierarchical tree with 10 nodes. The node weights and edge weights are denoted as w_i and $w_{i \rightarrow j}$, respectively. By the height of each node, we can obtain a good quasi-ranking, which is defined as follows.

Definition 2 (Node Height). The node height of a hierarchical tree refers to the height of the subtree rooted at the current node. Let the height of node i be denoted as h_i , then h_i can be calculated by the following recursive expression.

$$h_i = \begin{cases} 1 & , \text{node } i \text{ is leaf} \\ \max_{j \in C_i} h_j + 1 & , \text{otherwise} \end{cases} . \quad (1)$$

By the height of the nodes, we define a kind of induced subtree of hierarchical trees, called k -Order Subtree.

Definition 3 (k -Order Subtree). For a hierarchical tree \mathcal{T} under the descending order of height, the k -Order Subtree $\mathcal{T}^{(k)}$ is defined as an induced subtree retained after \mathcal{T} deletes all leaves k times. $\mathcal{T}^{(k)}$ satisfies

$$\mathcal{T}^{(k)} = \{i \mid i \in \mathcal{T} \wedge h_i > k\} . \quad (2)$$

As a bottom-up induced subtree, $\mathcal{T}^{(k)}$ satisfies transitive, i.e., $\mathcal{T}^{(a)(b)} = \mathcal{T}^{(a+b)}$. It can help us determine whether the subtrees obtained in different ways are equivalent. In subsequent applications, we use the concept of k -Order Subtree to simplify the description of the tree structure. In Fig. 1, we use dotted circles to mark the subtrees of \mathcal{T} from 0 to 2 orders. It can be seen that the 0-Order Subtree is \mathcal{T} itself actually; $\mathcal{T}^{(1)}$ is a subtree composed of non-leaf nodes of \mathcal{T} . Let n_k denote the number of nodes contained in $\mathcal{T}^{(k)}$, then there are $n_0 \equiv n$ and n_1 equal to the number of non-leaf nodes of \mathcal{T} .

3.2. Optimally Consistent Release of Differentially Private Hierarchical Tree

Before describing the optimally consistent releasing of the differentially private hierarchical tree, we first recall the hierarchical tree releasing model. Consider a set of unit counts $x_i : \mathcal{D} \rightarrow \mathbb{N} (1 \leq i \leq m)$ for private dataset \mathcal{D} , where x_i indicates the number of records in \mathcal{D} that satisfies the mutually exclusive unit condition φ_i . The unit count x_i satisfies

$$x_i = |\{t \in \mathbf{D} \mid \varphi_i(t) = \text{True}\}| . \quad (3)$$

Since φ_i is mutually exclusive, any $t \in \mathcal{D}$ satisfies and only satisfies one φ_i . Therefore, organizing x_i into the form of a vector, we will get $\mathbf{x} = [x_1, x_2, \dots, x_m]^T$.

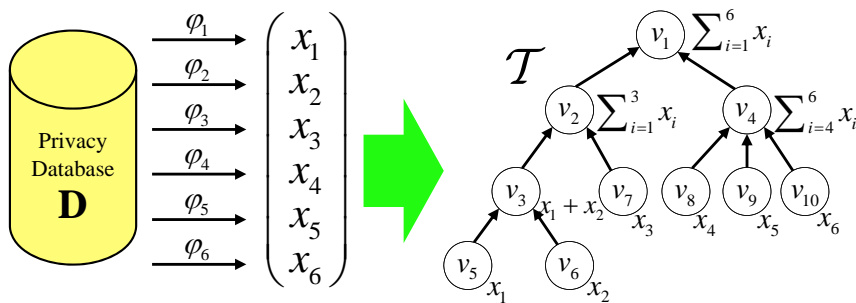


Figure 2: The Process of Hierarchical Tree Release

As shown in Fig. 2, each leaf corresponds to a x_i . The non-leaf node's value equals the sum of the leaves' value of the subtree rooted at that node. Therefore, the results of the hierarchical tree meet the consistency, i.e., "the sum of the values of the child nodes is equal to the value of the parent node". In Fig. 2, we denote the value of the i -th node as v_i . Then, organizing v_i into the form of $\mathbf{v} = [v_1, v_2, \dots, v_n]^T$ in turn, we can get the to-be-released data \mathbf{v} .

However, several works[13, 19, 20, 21, 14] have demonstrated that releasing an unprotected hierarchical tree will result in privacy disclosure. To protect individual privacy, DWork et al. [18] proposed differential privacy, defined as follows.

Definition 4 (ϵ -Differential Privacy[18]). *If a random algorithm \mathcal{M} satisfies ϵ -difference privacy, then for any two neighboring datasets \mathcal{D} and \mathcal{D}' , all outputs $O \in \text{Range}(\mathcal{M})$ satisfies*

$$\Pr(\mathcal{M}(\mathcal{D}) = O) \leq e^\epsilon \Pr(\mathcal{M}(\mathcal{D}') = O). \quad (4)$$

Under differential privacy, the process of hierarchical tree releasing can be described as

$$\tilde{\mathbf{v}} = \mathbf{v} + \xi, \quad (5)$$

where $\tilde{\mathbf{v}}$ is the \mathbf{v} after noise addition, which satisfies differential privacy. ξ is the random vector for the noise addition. Each element ξ_i is i.i.d and satisfies $\xi_i \sim \text{Lap}(\Delta/\epsilon)$, where Lap represents a Laplacian distribution and Δ is data sensitivity. In hierarchical tree releasing, Δ equals the height of \mathcal{T} [13].

To keep the consistency of the hierarchical tree after adding noise, we can get the optimally consistent release $\bar{\mathbf{v}}$ by following the optimization equation according to maximum likelihood post-processing proposed by Lee et al. [22].

$$\min_{\bar{\mathbf{v}}} \|\bar{\mathbf{v}} - \tilde{\mathbf{v}}\| \quad \text{s.t.} \quad \mathbf{M}_{\mathcal{T}}^T \bar{\mathbf{v}} = \mathbf{0}, \quad (6)$$

where $\mathbf{M}_{\mathcal{T}}$ is the consistency constraint matrix of a hierarchical tree, defined as follows.

Definition 5 (Consistency Constraint Matrix of Hierarchical Tree). *Given a hierarchical tree \mathcal{T} containing n nodes. Let n_1 denote the number of non-leaf nodes in \mathcal{T} . The value m_{ij} in row i and column j of the consistency constraint matrix $\mathbf{M}_{\mathcal{T}} \in \mathbb{R}^{n \times n_1}$ is defined as follows:*

$$m_{ij} = \begin{cases} 1 & , i = j \\ -1 & , j = f_i \\ 0 & , \text{otherwise} \end{cases}, \quad (7)$$

where f_i is the parent of node i .

The optimization equation (6) has the following closed-form expression:

$$\bar{\mathbf{v}} = \tilde{\mathbf{v}} - \mathbf{M}_{\mathcal{T}}(\mathbf{M}_{\mathcal{T}}^T \mathbf{M}_{\mathcal{T}})^{-1} \mathbf{M}_{\mathcal{T}}^T \tilde{\mathbf{v}}. \quad (8)$$

Since Formula (8) involves the inner product and inverse operations of the matrix, the time complexity of the direct solution is as high as $O(n^3)$. The amount of calculation is too large to obtain an efficient enough algorithm directly by the expressions. On the surface, Formula (8) is not a good choice for solving optimally consistent releases, but under the theories of Generation Matrix, we can convert it into another form and apply the properties of Generation Matrix to obtain an efficient algorithm.

4. Generation Matrix Model for Hierarchical Tree

4.1. Generation Matrix

Before defining Generation Matrix, we number the nodes of \mathcal{T} by descending order of height firstly.

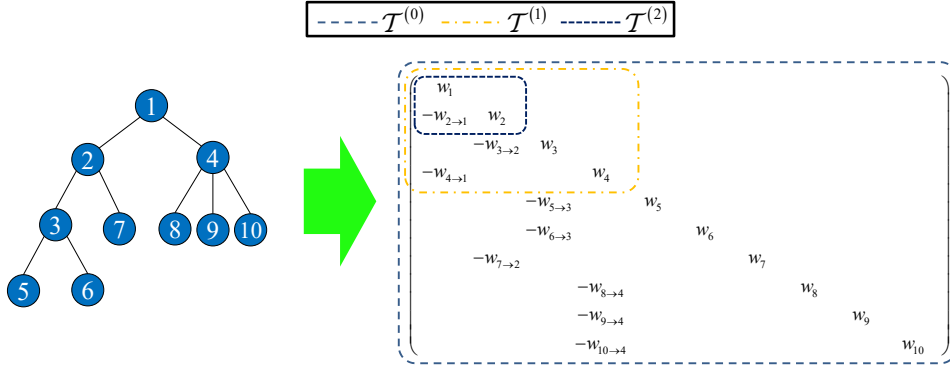


Figure 3: Generation Matrix and Its k -Order Submatrix

Definition 6 (Descending Order of Height). Let h_i denote the height of node i defined by Def. 2. If any two nodes i and j in \mathcal{T} satisfy

$$i < j \Rightarrow h_i \geq h_j, \quad (9)$$

we say that \mathcal{T} satisfies Descending Order of Height.

Under the descending order of height, we define Generation Matrix for \mathcal{T} .

Definition 7 (Generation Matrix). Considering a non-zero weighted hierarchical tree \mathcal{T} under descending order of height, let w_i and $w_{i \rightarrow f_i}$ ($w_i, w_{i \rightarrow f_i} \neq 0$) denote the weight values of the node i and the edge $i \rightarrow f_i$. Organizing them into a vector, denoted as \mathbf{w}_{node} and \mathbf{w}_{edge} , the generation matrix is denoted as $\mathbf{G}_{\mathcal{T}}^{(\mathbf{w}_{\text{node}}, \mathbf{w}_{\text{edge}})} \in \mathbb{R}^{n \times n}$, whose element $g_{i,j}$ in row i and column j is defined as follows,

$$g_{i,j} = \begin{cases} w_i & , i = j \\ -w_{i \rightarrow f_i} & , i \rightarrow f_i \\ 0 & , \text{otherwise} \end{cases}. \quad (10)$$

As shown in Fig. 3, since \mathcal{T} satisfies Descending Order of Height, the number of any non-root node i in \mathcal{T} is always bigger than its parent. It ensures Generation Matrix is always a lower triangular matrix. According to Def. 7, there is a one-to-one mapping between arbitrary non-zero weighted hierarchical tree and $\mathbf{G}_{\mathcal{T}}^{(\mathbf{w}_{\text{node}}, \mathbf{w}_{\text{edge}})} \in \mathbb{R}^{n \times n}$, i.e., the matrix representation of the non-zero weighted hierarchical tree is unique. When we only need to describe the structure of the hierarchical tree, we can use a Generation Matrix with all weights of 1 to represent it, i.e., $\mathbf{G}_{\mathcal{T}}^{(1,1)}$. We call $\mathbf{G}_{\mathcal{T}}^{(1,1)}$ **Structure Matrix**, which is abbreviated as $\mathbf{G}_{\mathcal{T}}$. If two hierarchical trees have the same structure and arrangement, the positions of non-zero elements in their Generation Matrices are always the same, which we call Similar Generation Matrices.

Definition 8 (Similar Generation Matrices). If \mathbf{G}_1 and \mathbf{G}_2 are two Generation Matrices defined by the hierarchical trees with the same structure and arrangement (or the same tree), we call them similar Generation Matrices, which are denoted as $\mathbf{G}_1 \sim \mathbf{G}_2$.

Every Generation Matrix from the same hierarchical tree is always similar. We can use $\mathbf{G} \sim \mathbf{G}_{\mathcal{T}}$ as sufficient to judge whether the hierarchical tree represented by \mathbf{G} has the same structure as \mathcal{T} .

According to Def. 3, $\mathcal{T}^{(k)}$ is an induced subtree composed of nodes i with $h_i \geq k$ in \mathcal{T} . Under Descending Order of Height, these nodes are always ranked first. In Fig. 3, k -Order Submatrix that represents the k -Order Subtree is denoted as $\mathbf{G}_{\mathcal{T}^{(k)}}^{(\mathbf{w}_{\text{node}}, \mathbf{w}_{\text{edge}})}$. We can obtain it by taking the n_k -order leading principal minor of $\mathbf{G}_{\mathcal{T}}^{(\mathbf{w}_{\text{node}}, \mathbf{w}_{\text{edge}})}$ (i.e., the elements in rows and columns from 1 to n_k). In particular, $\mathbf{G}_{\mathcal{T}^{(1)}}^{(\mathbf{w}_{\text{node}}, \mathbf{w}_{\text{edge}})}$ represents the sub-tree composed of non-leaf nodes of \mathcal{T} .

Considering a specific application, one problem we may encounter is that the nodes of the hierarchical tree are numbered but not in Descending Order of Height. Under the matrix representation, the problem is elementary to solve. We can adopt a sparse mapping matrix to convert the original number into Descending Order of Height.

Definition 9 (Mapping Matrix). *Given an ordered set $\mathcal{S} = \langle s_1, s_2, \dots, s_m \rangle$ represents the mapping relationship between integers, satisfying $s_i \in \mathbb{N}^+ \wedge s_i \leq n$, the mapping matrix defined by \mathcal{S} is denoted as $\mathbf{H}_{\mathcal{S}} \in \{0, 1\}^{m \times n}$. The value $h_{i,j}$ in row i and column j of $\mathbf{H}_{\mathcal{S}}$ satisfies*

$$h_{i,j} = \begin{cases} 1 & , s_i = j \\ 0 & , \text{otherwise} \end{cases} . \quad (11)$$

By the definition of the mapping matrix, \mathcal{S} always represents an injection and satisfies $\mathbf{H}_{\mathcal{S}}\mathbf{H}_{\mathcal{S}}^T = \mathbf{I}$. If \mathcal{S} represents a bijection, $\mathbf{H}_{\mathcal{S}}$ will be a permutation matrix. When the basis vector \mathbf{e}_i acts on $\mathbf{H}_{\mathcal{S}}$, the following equations holds.

$$\mathbf{H}_{\mathcal{S}}^T \mathbf{e}_i = \mathbf{e}_{s_i}, \quad (12)$$

$$\mathbf{H}_{\mathcal{S}} \mathbf{e}_i = \begin{cases} \mathbf{e}_{s_i^{-1}} & , i \in \mathcal{S} \\ \mathbf{0} & , i \notin \mathcal{S} \end{cases} , \quad (13)$$

where s_i^{-1} represents the inverse mapping of s_i , which satisfies $s_{s_i^{-1}} = i$.

To describe the mapping relationship between the nodes before and after sorting by Descending Order of Height, we only need to define the ordered set $\mathcal{S} = \langle s_1, s_2, \dots, s_n \rangle$, where s_i represents the sorted number of the node initially numbered i . Then, we can use $\mathbf{H}_{\mathcal{S}}\mathbf{v}$ to get the vector before sorting from \mathbf{v} after sorting.

In addition, the mapping matrix can be used to represent other mapping relationships, such as the mapping of v_i to x_i . For example, in Fig. 1, if we use an ordered set $\mathcal{H} = \langle h_1, h_2, \dots, h_m \rangle$ to represent the mapping relationship between v_i and x_i , then $h_i = i + 5$, and we have the mapping matrix $\mathbf{H}_{\mathcal{H}}$ to represent their mapping relationship.

4.2. Properties of Generation Matrix

Our research shows that Generation Matrix has many mathematical properties that deserve attention. These properties can help us solve various problems about the analysis and calculation of hierarchical trees. According to Def. 7, it is not difficult to find that Generation Matrix satisfies sparsity.

Property 1 (Sparsity). *Considering a hierarchical tree \mathcal{T} consists of n nodes, $\mathbf{G}_{\mathcal{T}}^{(\mathbf{w}_{node}, \mathbf{w}_{edge})}$ has and only has $2n - 1$ non-zero elements. Its first row has only 1 non-zero element, and the remaining $n - 1$ rows have 2 non-zero elements.*

Due to the sparsity of Generation Matrix, we can apply various sparse matrix technologies such as COO (Coordinate Format) and CSR (Compressed Sparse Row) to calculate hierarchical tree models efficiently. Under the sparse representations, the storage and calculation of Generation Matrix are both only $O(n)$. Therefore, the application based on Generation Matrix does not cause more computational overhead. Currently, the computing technologies of sparse matrices are very mature and widely used in various high-performance platforms [29, 30, 31].

One of the fundamental properties of Generation Matrices is invertibility. By solving the equations $\mathbf{G}_{\mathcal{T}}^{(\mathbf{w}_{node}, \mathbf{w}_{edge})^T} \mathbf{z} = \mathbf{v}$ and $\mathbf{G}_{\mathcal{T}}^{(\mathbf{w}_{node}, \mathbf{w}_{edge})} \mathbf{z} = \mathbf{v}$ about \mathbf{z} , we find two interesting and important mathematical properties of Generation Matrices. We collectively call them the propagation of Generation Matrix.

Property 2 (Upward Propagation). *Let $g_{i,j}$ denotes the element in row i and column j of $\mathbf{G}_{\mathcal{T}}^{(\mathbf{w}_{node}, \mathbf{w}_{edge})}$, and v_i is the value of node i of \mathcal{T} . Organize v_i into vector $\mathbf{v} = [v_1, v_2, \dots, v_n]^T$, then $\mathbf{z} = \mathbf{G}_{\mathcal{T}}^{(\mathbf{w}_{node}, \mathbf{w}_{edge})^{-T}} \mathbf{v}$ is an upward propagation on \mathbf{v} . The value z_i of \mathbf{z} satisfies*

$$z_i = \begin{cases} v_i / g_{i,i} & , \text{node } i \text{ is leaf} \\ \left(v_i - \sum_{j \in \mathcal{C}_i} g_{j,i} z_j \right) / g_{i,i} & , \text{otherwise} \end{cases} . \quad (14)$$

Property 3 (Downward Propagation). If $\mathbf{G}_{\mathcal{T}}^{(\mathbf{w}_{\text{node}}, \mathbf{w}_{\text{edge}})}$ and \mathbf{v} have the exact definition as Prop. 2, then $\mathbf{z} = \mathbf{G}_{\mathcal{T}}^{(\mathbf{w}_{\text{node}}, \mathbf{w}_{\text{edge}})^{-1}} \mathbf{v}$ is a downward propagation on \mathbf{v} . The value z_i of \mathbf{z} satisfies

$$z_i = \begin{cases} v_i / g_{i,i} & , \text{node } i \text{ is root} \\ (v_i - g_{i,f_i} z_{f_i}) / g_{i,i} & , \text{otherwise} \end{cases} . \quad (15)$$

Prop. 2 and Prop. 3 show that Generation Matrix can simulate multiple recursive operations of hierarchical trees. According them, Cor. 1 analyzes the elements affected by the propagations of Generation Matrix.

Corollary 1. For the upward propagation, affected z_j by v_i satisfies $j = i$, or j is the ancestor of i in \mathcal{T} ; for the downward propagation, affected z_j by v_i satisfies $j = i$, or j is a descendant of i in \mathcal{T} .

Combining the propagations, we further study a variety of matrix operations of Generation Matrices. They have strong interpretability and provide crucial theoretical support for the research of hierarchical trees.

Property 4. Let the vector $\mathbf{z} = (\mathbf{I} - \mathbf{G}_{\mathcal{T}}^T) \mathbf{1}$, then the i -th element z_i of \mathbf{z} represents the number of children of the node i , i.e., $z_i = |C_i|$.

Property 5. If the vector $\mathbf{z} = \mathbf{G}_{\mathcal{T}}^{-T} \mathbf{1}$, the i -th element z_i of \mathbf{z} represents the number of nodes contained in the subtree rooted at node i .

Property 6. Let the vector $\mathbf{z} = \mathbf{G}_{\mathcal{T}}^{-1} \mathbf{1}$, then the i -th element z_i of \mathbf{z} represents the depth of node i , where the depth of the root is 1.

The properties above indicate that Generation Matrix is an effective and easy-to-use tool for various hierarchical tree analyses. In addition to the conclusions about vectors discussed above, there are some conclusions about matrices as follows. Compared with conclusions about vectors, they focus on describing the characteristics between nodes.

Property 7. Let $g_{ij}^{(-1)}$ denote the element in row i and column j of $\mathbf{G}_{\mathcal{T}}^{-1}$, then $g_{ij}^{(-1)}$ satisfies

$$g_{ij}^{(-1)} = \begin{cases} 1 & , i = j \vee j \text{ is an ancestor of } i \\ 0 & , \text{otherwise} \end{cases} . \quad (16)$$

Proof. See Appendix A.1. □

Prop. 7 shows that $\mathbf{G}_{\mathcal{T}}^{-1}$ is a matrix indicating the relationship between ancestors and descendants. Although $\mathbf{G}_{\mathcal{T}}^{-1}$ is denser than $\mathbf{G}_{\mathcal{T}}$, in most cases, $\mathbf{G}_{\mathcal{T}}^{-1}$ is still sparse.

Property 8. Let $\mathbf{M} = \mathbf{G}_{\mathcal{T}} \mathbf{G}_{\mathcal{T}}^T$ and $m_{i,j}$ denote the element in row i and column j of \mathbf{M} . Except for $m_{11} = 1$, other elements satisfy “ $m_{ij} = 1 \Leftrightarrow i$ and j are sibling nodes”.

Property 9. Let $\mathbf{M} = (\mathbf{G}_{\mathcal{T}}^T \mathbf{G}_{\mathcal{T}})^{-1}$ and m_{ij} is the element in row i and column j of \mathbf{M} , then the value of m_{ij} represents the number of common ancestors of the node pair i and j , and m_{ii} represents the depth of i .

Proof. See Appendix A.2. □

Similar to Prop. 7, Prop. 8 can also be used as an indicator matrix to describe the relationship between nodes. Prop. 9 is an important property, which describes an effective method of calculating common ancestors. As an essential feature to describe the correlation between nodes, the number of common ancestors has an important application value for hierarchical tree analyses[8].

In the study of spectral graph theory, feature analysis is usually indispensable. Our research shows that the eigenvalues and eigenvectors of a Generation Matrix satisfy the following properties.

Property 10 (Eigenvalues and Eigenvectors). Let $\lambda = [\lambda_1, \lambda_2, \dots, \lambda_n]^T$ denote the eigenvalues of $\mathbf{G}_{\mathcal{T}}^{(\mathbf{w}_{node}, \mathbf{w}_{edge})}$, then the i -th eigenvalue λ_i is w_i .

Let the left eigenvector and the right eigenvector corresponding to the i -th eigenvalue denote as \mathbf{u}_i and \mathbf{v}_i , respectively. The premise of the existence of \mathbf{u}_i is that each ancestor j of i satisfies $w_i \neq w_j$, and the premise of the existence of \mathbf{v}_i is that each descendant j of i satisfies $w_i \neq w_j$.

Let the j -th element of \mathbf{u}_i and \mathbf{v}_i denote as $u_j^{(i)}$ and $v_j^{(i)}$, respectively. If \mathbf{u}_i exists, then $u_j^{(i)} = 0$ for any $j > i$. Let $u_i^{(i)} = 1$. The remaining elements $u_j^{(i)}$ ($j < i$) can be obtained by

$$u_j^{(i)} = \begin{cases} \frac{\sum_{k \in C_j} w_{k \rightarrow j} u_k^{(i)}}{w_j - w_i} & , j \text{ is a ancestor of } i \\ 0 & , \text{otherwise} \end{cases} \quad (17)$$

If \mathbf{v}_i exists, then $v_j^{(i)} = 0$ for any $j < i$. Let $v_i^{(i)} = 1$. The remaining elements $v_j^{(i)}$ ($j > i$) can be obtained by

$$v_j^{(i)} = \begin{cases} \frac{w_{j \rightarrow f_j} v_{f_j}^{(i)}}{w_j - w_i} & , j \text{ is a descendant of } i \\ 0 & , \text{otherwise} \end{cases} \quad (18)$$

Prop. 10 shows that the eigenvalues and eigenvectors of the Generation Matrix have many interesting properties. For example, the eigenvalue of Generation Matrix is the weights of the nodes, which is much easier to solve than other matrix representations; the eigenvectors also satisfy some propagation properties similar to Prop. 2 and 3. Notably, the eigenvectors are conditional, which means that Generation Matrix is not always diagonalizable. Some Generation Matrices, especially the eigenvectors of $\mathbf{G}_{\mathcal{T}}$, still have many problems waiting to be studied. Although feature analysis is not the main focus in this paper, Prop. 10 still provides some valuable references for the subsequent research works.

Considering the relationship between $\mathbf{G}_{\mathcal{T}}^{(\mathbf{w}_{node}, \mathbf{w}_{edge})}$ and corresponding $\mathbf{G}_{\mathcal{T}}$, we find that $\mathbf{G}_{\mathcal{T}}^{(\mathbf{w}_{node}, \mathbf{w}_{edge})}$ satisfies a particular decomposition form, which we call the diagonal decomposition of Generation Matrix.

Property 11 (Diagonal Decomposition). Given arbitrary $\mathbf{G}_{\mathcal{T}}^{(\mathbf{w}_{node}, \mathbf{w}_{edge})}$, there is always a pair of vectors $\alpha, \beta \in \mathbb{R}^n$, making the following decomposition hold for $\mathbf{G}_{\mathcal{T}}^{(\mathbf{w}_{node}, \mathbf{w}_{edge})}$ and the structure matrix $\mathbf{G}_{\mathcal{T}}$.

$$\mathbf{G}_{\mathcal{T}}^{(\mathbf{w}_{node}, \mathbf{w}_{edge})} = \text{diag}(\beta) \mathbf{G}_{\mathcal{T}} \text{diag}(\alpha) \quad (19)$$

Let α_i and β_i denote the i -th element of them, respectively. Then a pair of legal α and β can be obtained by

$$\begin{cases} \alpha = \exp(\mathbf{G}_{\mathcal{T}}^{-1} (\ln(\mathbf{w}_{node}) - \ln(\mathbf{w}_{edge}))) \\ \beta = \mathbf{w}_{node} \oslash \alpha \end{cases} \quad (20)$$

Where “ \oslash ” denote the element-wise division of vectors. Note, since the root numbered 1 has no parent, we set $w_{1 \rightarrow \emptyset} = 1$ as the first element of \mathbf{w}_{edge} , and $w_{i \rightarrow f_i}$ is the i -th element of \mathbf{w}_{edge} in the remaining elements.

Proof. See Appendix A.3. □

By the diagonal decomposition of Generation Matrix, we can express arbitrary $\mathbf{G}_{\mathcal{T}}^{(\mathbf{w}_{node}, \mathbf{w}_{edge})}$ as an expression with $\mathbf{G}_{\mathcal{T}}$. Using Prop. 11, we can extend some mathematical properties of $\mathbf{G}_{\mathcal{T}}$ to $\mathbf{G}_{\mathcal{T}}^{(\mathbf{w}_{node}, \mathbf{w}_{edge})}$ to solve more problems effectively.

4.3. The Conversion between Generation Matrix and Other Matrix Representations

Our research shows that Generation Matrix is not an isolated matrix representation from others. Through proper operations, we can convert Generation Matrix into other matrix representations. Fig. 4 shows the four matrix representations that can be transformed by the Generation Matrix constructed by the hierarchical tree in Fig. 1, including Adjacency Matrix, Laplacian Matrix, Distance Matrix, and Ancestor Matrix.

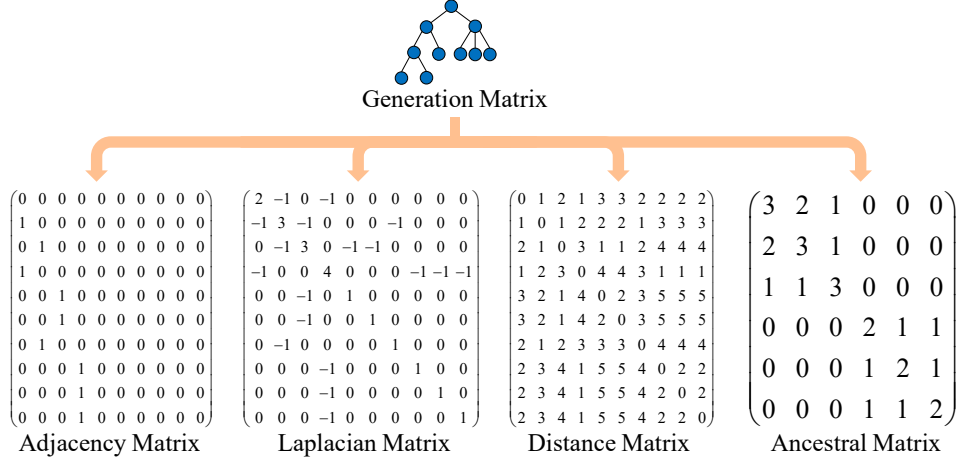


Figure 4: Conversion Relationships from Generation Matrix to Other Matrix Representations

Theorem 1. Let $\mathbf{A}_{\mathcal{T}}$ be Adjacency Matrix of \mathcal{T} , then $\mathbf{A}_{\mathcal{T}}$ can be obtained by the following expression of $\mathbf{G}_{\mathcal{T}}$:

$$\mathbf{A}_{\mathcal{T}} = \mathbf{I} - \mathbf{G}_{\mathcal{T}}. \quad (21)$$

Theorem 2. Let $\mathbf{L}_{\mathcal{T}}$ be Laplacian Matrix of \mathcal{T} , then $\mathbf{L}_{\mathcal{T}}$ can be obtained by the following expression of $\mathbf{G}_{\mathcal{T}}$:

$$\mathbf{L}_{\mathcal{T}} = \mathbf{G}_{\mathcal{T}}^T \mathbf{G}_{\mathcal{T}} - \mathbf{e}_1 \mathbf{e}_1^T. \quad (22)$$

Theorem 3. Let $\mathbf{D}_{\mathcal{T}}$ be Distance Matrix of \mathcal{T} , then $\mathbf{D}_{\mathcal{T}}$ can be obtained by the following expression of $\mathbf{G}_{\mathcal{T}}$:

$$\mathbf{D}_{\mathcal{T}} = \mathbf{G}_{\mathcal{T}}^{-1} \mathbf{H}^T + \mathbf{H}^T \mathbf{G}_{\mathcal{T}}^{-T} - 2(\mathbf{G}_{\mathcal{T}}^T \mathbf{G}_{\mathcal{T}})^{-1}. \quad (23)$$

Theorem 4. Let $\mathbf{C}_{\mathcal{T}}$ be Ancestral Matrix[8] of \mathcal{T} , then $\mathbf{C}_{\mathcal{T}}$ can be obtained by the following expression of $\mathbf{G}_{\mathcal{T}}$:

$$\mathbf{C}_{\mathcal{T}} = \mathbf{H}_{\mathcal{H}}(\mathbf{G}_{\mathcal{T}}^T \mathbf{G}_{\mathcal{T}})^{-1} \mathbf{H}_{\mathcal{H}}^T - \mathbf{1}. \quad (24)$$

Proof. The proofs of Thm. 2-4 in Appendix A.4 to A.6. □

It can be seen from the theorems above that we can convert Generation Matrix into other matrix representations by simple matrix expressions. However, the reverse is not easy. Except for Adjacency Matrix, other matrix representations cannot be directly converted back to Generation Matrix. Therefore, we can use Generation Matrix to construct other matrix representations. Besides, it also implies that the theories of Generation Matrix have a particular internal connection with the matrix represented. We can combine the theories of Generation Matrix and other matrix representations to solve more problems about hierarchical trees.

5. The Application on Differentially Private Hierarchical Tree Release

5.1. Hierarchical Tree Release Based on Generation Matrix

In this section, we introduce how to apply Generation Matrix to efficiently and concisely achieve an optimally consistent release on differentially private hierarchical tree release. Since each leaf of \mathcal{T} corresponds to a x_i , we use

the mapping matrix $\mathbf{H}_{\mathcal{H}}$ to represent the mapping relationship between leaf nodes and hierarchical tree nodes. Using Prop. 2, the hierarchical tree building process $\text{BuildTree}_{\mathcal{T}}$ can be described as

$$\mathbf{v} = \text{BuildTree}_{\mathcal{T}}(\mathbf{x}) = \mathbf{G}_{\mathcal{T}}^{-T} \mathbf{H}_{\mathcal{H}}^T \mathbf{x}. \quad (25)$$

At the same time, we also define the inverse tree-building process $\text{BuildTree}_{\mathcal{T}}^{-1}$ as the following expression, which is taking the leaves of \mathcal{T} and restoring them to \mathbf{x} .

$$\mathbf{x} = \text{BuildTree}_{\mathcal{T}}^{-1}(\mathbf{v}) = \mathbf{H}_{\mathcal{H}} \mathbf{v}. \quad (26)$$

Although most works[13, 19, 20, 21, 14] do not take matrix analysis as the theoretical basis for optimally consistent release, there are many advantages to applying matrix analysis. One of them is error analysis. Using matrix analysis, we can quickly calculate the overall mean square error of the “Node Query” after the post-processing for optimally consistent release.

Theorem 5. *Given the privacy budget ε and the to-be-released hierarchical tree \mathcal{T} containing n nodes and m leaves, whose height is h , the overall mean square error before and after post-processing $\text{mse}(\tilde{\mathbf{v}})$ and $\text{mse}(\bar{\mathbf{v}})$ satisfy*

$$\text{mse}(\tilde{\mathbf{v}}) = \sum_{i=1}^n \mathbb{E}(\tilde{v}_i - v_i)^2 = 2nh^2/\varepsilon^2, \quad (27)$$

$$\text{mse}(\bar{\mathbf{v}}) = \sum_{i=1}^n \mathbb{E}(\bar{v}_i - v_i)^2 = 2mh^2/\varepsilon^2. \quad (28)$$

Proof. See Appendix A.7. □

According to Thm. 5, the overall mean square error depends on the number of leaves after post-processing. Generally, n is much less than the number of leaves m , so post-processing will significantly reduce the error. As the proof shown in Appendix A.7, We applied the property of the trace of the matrix to obtain a concrete and concise demonstration, which embodies matrix analysis’s great potential for solving problems.

Next, we will introduce how to apply Generation Matrix to achieve an efficient enough algorithm.

5.2. “LO”–QR Decomposition on $\mathbf{M}_{\mathcal{T}}$

To obtain an efficient release algorithm, we apply QR decomposition to analyze Formula (8). Unfortunately, the traditional QR decomposition[23] cannot meet our analysis requirements, so we propose another QR decomposition form, namely the “LO”–QR decomposition.

Definition 10 (“LO”–QR Decomposition). *For matrix $\mathbf{M} \in \mathbb{R}^{n \times m}$ ($n \geq m$), QR decomposition looks for an orthogonal matrix \mathbf{Q} , which converts \mathbf{M} into a form composed of lower triangular matrix \mathbf{L} and zero matrix. i.e.,*

$$\mathbf{M} = \mathbf{Q} \begin{bmatrix} \mathbf{L} \\ \mathbf{0} \end{bmatrix}. \quad (29)$$

Correspondingly, we call the traditional QR decomposition the “UO”–QR decomposition, which decomposes a matrix into an upper triangular matrix. Although the two have different forms, they both achieve decomposition through a series of basic orthogonal transformations. Householder transformation is the most widely used among various orthogonal transformation techniques because of its high efficiency, easy implementation, and applicability to sparse matrices. Completing QR decomposition once requires m householder transformations. To describe the QR decomposition process in more detail, we define (\mathcal{S}, j) –Householder transformation to describe each transformation.

Definition 11 (\mathcal{S}, j) –Householder Transformation). *For matrix \mathbf{M} , given an ordered set $\mathcal{S} = \langle s_1, s_2, \dots, s_r \rangle$ and column number j , (\mathcal{S}, j) –Householder Transformation selects the rows s_1, s_2, \dots, s_r and column j of \mathbf{M} as the reference*

for householder transformation, $\mathbf{Y} = \mathbf{QM}$. The transformation result will make \mathbf{Y} satisfy $y_{s_1,j} = \sqrt{\sum_{s \in \mathcal{S}} m_{s,j}^2}$ and $y_{s_i,j} = 0, 2 \leq i \leq r$, where \mathbf{Q} satisfies the following expression:

$$\begin{cases} \mathbf{m} = \mathbf{H}_S \mathbf{M} \mathbf{e}_j \\ \boldsymbol{\omega} = \mathbf{m} - \|\mathbf{m}\| \mathbf{e}_1 \\ \mathbf{Q} = \mathbf{I} - 2\mathbf{H}_S^T \frac{\boldsymbol{\omega} \boldsymbol{\omega}^T}{\boldsymbol{\omega}^T \boldsymbol{\omega}} \mathbf{H}_S \end{cases}. \quad (30)$$

For \mathbf{M} , we denote (\mathcal{S}, j) -Householder transformation as $\mathbf{Y} = \text{House}_{\mathcal{S},j}(\mathbf{M})$. The QR decomposition based on the Householder transformation can be expressed as

$$\mathbf{R} = \text{House}_{\zeta_{n_1}} \cdot \cdots \cdot \text{House}_{\zeta_2} \cdot \text{House}_{\zeta_1}(\mathbf{M}), \quad (31)$$

where \mathbf{R} is the result of QR decomposition and “ \cdot ” represents the operation of function composition such as $f_2 \cdot f_1(x) = f_2(f_1(x))$. For “UO”-QR decomposition, the parameter $\zeta_i = (\langle i, \dots, n \rangle, i)$; for “LO”-QR decomposition, the parameter $\zeta_i = (\langle m-i+1, 1, \dots, m-i, m+1, \dots, n \rangle, m-i+1)$.

Next, we apply the “LO”-QR decomposition on $\mathbf{M}_{\mathcal{T}}$. To understand how “LO”-QR decomposition affects $\mathbf{M}_{\mathcal{T}}$, we divide $\mathbf{M}_{\mathcal{T}}$ into the following forms,

$$\mathbf{M}_{\mathcal{T}} \equiv \begin{bmatrix} \mathbf{M}_{\uparrow} \\ \mathbf{M}_{\downarrow} \end{bmatrix}. \quad (32)$$

The upper half $\mathbf{M}_{\uparrow} \in \mathbb{R}^{n_1 \times n_1}$ of $\mathbf{M}_{\mathcal{T}}$ consists of the first n_1 rows; The second half $\mathbf{M}_{\downarrow} \in \mathbb{R}^{m \times n_1}$ consists of the remaining m rows. According to Def. 5, we have $\mathbf{M}_{\uparrow} = \mathbf{G}_{\mathcal{T}^{(1)}}$ and $\mathbf{M}_{\downarrow} = -\mathbf{H}_{\mathcal{S}^{(f)}}$, where the i -th element of ordered set $\mathcal{S}^{(f)}$ satisfies $s_i^{(f)} = f_{i+n_1}$. Considering the effects of Householder transformation on $\mathbf{M}_{\mathcal{T}}$, we denote the matrix obtained after the k -th Householder transformation as $\mathbf{M}_{\mathcal{T}}^{(k)}$ ($0 \leq k \leq n_1$), whose upper half and second half are denoted as $\mathbf{M}_{\uparrow}^{(k)}$ and $\mathbf{M}_{\downarrow}^{(k)}$. $\mathbf{M}_{\mathcal{T}}^{(0)}$ is the form before Householder transformation, satisfying $\mathbf{M}_{\mathcal{T}}^{(0)} = \mathbf{M}_{\mathcal{T}}$; $\mathbf{M}_{\mathcal{T}}^{(n_1)}$ is the result after “LO”-QR decomposition. Thm. 6 demonstrates that, in the process of Householder transformation, $\mathbf{M}_{\mathcal{T}}^{(k)}$ satisfies some invariant properties.

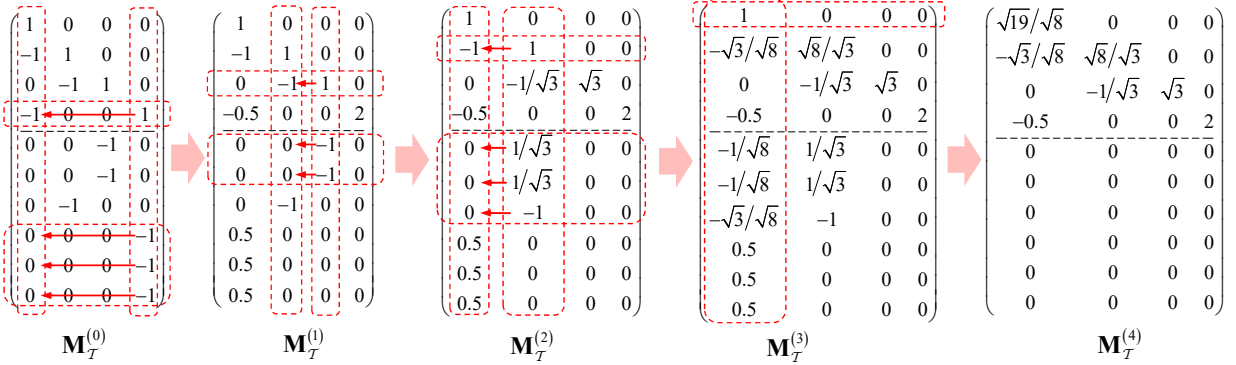


Figure 5: Householder Transformations in “LO”-QR Decomposition on $\mathbf{M}_{\mathcal{T}}$

Theorem 6. For the “LO”-QR decomposition on $\mathbf{M}_{\mathcal{T}}$, after k ($0 \leq k \leq n_1 - 1$) times of Householder transformation, $\mathbf{M}_{\mathcal{T}}^{(k)}$ always keeps the following three properties unchanged:

- a) $\mathbf{M}_{\uparrow}^{(k)} \sim \mathbf{G}_{\mathcal{T}^{(1)}}$;
- b) Each line of $\mathbf{M}_{\downarrow}^{(k)}$ has one and only one non-zero value;
- c) The last k columns of $\mathbf{M}_{\downarrow}^{(k)}$ (i.e., the columns from $n_1 - k + 1$ to n_1) are all 0.

Proof. See Appendix A.8. \square

As shown in Fig. 5, with the Householder transformation proceeds, the non-zero elements in $\mathbf{M}_{\downarrow}^{(k)}$ shift from right to left, and the position of the non-zero elements of $\mathbf{M}_{\uparrow}^{(k)}$ remains unchanged. Specifically, in the k -th Householder transformation, all non-zero elements in the $(n_1 - k + 1)$ -th column of $\mathbf{M}_{\downarrow}^{(k)}$ are transferred to the f_{n_1-k+1} -th column. Combining Thm. 6, we can infer the specific form of $\mathbf{M}_{\mathcal{T}}^{(n_1-1)}$ after $n_1 - 1$ times of Householder transformation. After the last householder transformation, the result satisfies the following theorem.

Theorem 7. After “LO”-QR decomposition, there is a $\mathbf{G}_{\mathcal{T}^{(1)}}^{(\mathbf{w}_{node}, \mathbf{w}_{edge})}$ and an orthogonal matrix \mathbf{Q} , which let $\mathbf{M}_{\mathcal{T}}$ satisfies:

$$\mathbf{M}_{\mathcal{T}} = \mathbf{Q} \begin{bmatrix} \mathbf{G}_{\mathcal{T}^{(1)}}^{(\mathbf{w}_{node}, \mathbf{w}_{edge})} \\ \mathbf{O} \end{bmatrix}. \quad (33)$$

Proof. See Appendix A.9. \square

Thm. 7 implies that the calculation $(\mathbf{M}_{\mathcal{T}}^T \mathbf{M}_{\mathcal{T}})^{-1}$ about $\mathbf{M}_{\mathcal{T}}$ in expression (8) can be replaced by $\mathcal{T}^{(1)}$. Since it is related to \mathcal{T} and $\mathcal{T}^{(1)}$ simultaneously, we denote it as $\mathbf{G}_{\mathcal{T}^{(1)} \leftarrow \mathcal{T}}$. Thm. 8 shows that $\mathbf{G}_{\mathcal{T}^{(1)} \leftarrow \mathcal{T}}$ and $\mathbf{M}_{\mathcal{T}}$ are inner-product-equivalent.

Theorem 8. For arbitrary $\mathbf{M}_{\mathcal{T}}$, there is a $\mathbf{G}_{\mathcal{T}^{(1)} \leftarrow \mathcal{T}}$ inner-product-equivalent to it, which satisfies

$$\mathbf{M}_{\mathcal{T}}^T \mathbf{M}_{\mathcal{T}} = \mathbf{G}_{\mathcal{T}^{(1)} \leftarrow \mathcal{T}}^T \mathbf{G}_{\mathcal{T}^{(1)} \leftarrow \mathcal{T}}, \quad (34)$$

where $\mathbf{G}_{\mathcal{T}^{(1)} \leftarrow \mathcal{T}}$ is the upper half $\mathbf{M}_{\downarrow}^{(n_1)}$ of $\mathbf{M}_{\mathcal{T}}$ after “LO”-QR decomposition.

Proof. See Appendix A.10. \square

5.3. Generation Matrix-based Optimally Consistent Release Algorithm

The property of inner product equivalence provides us with a vital optimization idea for an optimal release, as shown in Cor. 2. Using matrix analysis, we convert the expression (8) into an expression about $\mathbf{G}_{\mathcal{T}^{(1)} \leftarrow \mathcal{T}}$ and then use the mathematical properties of Generation Matrix to improve the efficiency of optimal release.

Corollary 2. The expression $\mathbf{y} = (\mathbf{M}_{\mathcal{T}}^T \mathbf{M}_{\mathcal{T}})^{-1} \mathbf{x}$ can be obtained by performing an upward propagation and a downward propagation successively about $\mathbf{G}_{\mathcal{T}^{(1)} \leftarrow \mathcal{T}}$. That is

$$\mathbf{y} = (\mathbf{M}_{\mathcal{T}}^T \mathbf{M}_{\mathcal{T}})^{-1} \mathbf{x} = \mathbf{G}_{\mathcal{T}^{(1)} \leftarrow \mathcal{T}}^{-1} (\mathbf{G}_{\mathcal{T}^{(1)} \leftarrow \mathcal{T}}^{-T} (\mathbf{M}_{\mathcal{T}}^T \mathbf{x})). \quad (35)$$

Proof. See Appendix A.11. \square

According to Cor. 2, we get another optimal release form as follows.

$$\bar{\mathbf{v}} = \tilde{\mathbf{v}} - \mathbf{M}_{\mathcal{T}} (\mathbf{G}_{\mathcal{T}^{(1)} \leftarrow \mathcal{T}}^{-1} (\mathbf{G}_{\mathcal{T}^{(1)} \leftarrow \mathcal{T}}^{-T} (\mathbf{M}_{\mathcal{T}}^T \tilde{\mathbf{v}}))). \quad (36)$$

We illustrate with parentheses that Formula (36) is calculated from right to left, ensuring that all multiplication and solution equations are for matrices and vectors. According to the sparsity of $\mathbf{M}_{\mathcal{T}}$, the time complexity of $\mathbf{M}_{\mathcal{T}}$ -related multiplication calculation in Formula (36) is $O(n)$. Besides, according to Prop. 2 and Prop. 3, the time complexity of solving the linear equation of Generation Matrix is also $O(n_1)$. Therefore, the overall time complexity of Formula (36) is $O(n)$. Formula (36) has completely summarized the core process of optimally consistent release. So long as we execute the formula directly after constructing $\mathbf{G}_{\mathcal{T}^{(1)} \leftarrow \mathcal{T}}$, we can efficiently obtain the optimal consistent release. The algorithm description with Generation Matrices is very concise and easy to implement.

However, only ensuring that the calculation of $\mathbf{G}_{\mathcal{T}^{(1)} \leftarrow \mathcal{T}}$ is also highly efficient, the whole process is efficient. So, we further proposed Thm. 9 to calculate it.

Theorem 9. Let $w_i (1 \leq i \leq n_1)$ and $w_{i \rightarrow f_i} (2 \leq i \leq n_1)$ as the weights of the nodes and edges in $\mathbf{G}_{\mathcal{T}^{(1)} \leftarrow \mathcal{T}}$, respectively. They satisfy

$$\begin{cases} w_i = \sqrt{1 + \theta_i} \\ w_{i \rightarrow f_i} = w_i^{-1} \end{cases}, \quad (37)$$

where $\theta_i (1 \leq i \leq n_1)$ satisfies

$$\theta_i = |C_i| - \sum_{j \in C_i \wedge j \leq n_1} (1 + \theta_j)^{-1}. \quad (38)$$

Proof. See Appendix A.12. □

Thm. 9 shows that w_i and $w_{i \rightarrow f_i}$ can be directly calculated by only requiring θ_i , and the calculation of θ_i only needs to traverse from n_1 to 1 once. Since this process involves the calculation of $|C_i|$, the overall time complexity of constructing $\mathbf{G}_{\mathcal{T}^{(1)} \leftarrow \mathcal{T}}$ is $O(n)$. The specific construction process is shown in Alg. 1.

Algorithm 1 Construct $\mathbf{G}_{\mathcal{T}^{(1)} \leftarrow \mathcal{T}}$ from $\mathbf{M}_{\mathcal{T}}$

Input: Consistency Constraint Matrix $\mathbf{M}_{\mathcal{T}}$

Output: the inner-product-equivalent $\mathbf{G}_{\mathcal{T}^{(1)} \leftarrow \mathcal{T}}$

- 1: Initialize $\boldsymbol{\theta} \in \mathbb{R}^{n_1 \times 1}$, satisfying $\theta_i = |C_i|$.
 - 2: **for** $i = n_1$ to 2 **do** $\theta_{f_i} \leftarrow \theta_{f_i} - (1 + \theta_i)^{-1}$;
 - 3: **for** $i = 1$ to n_1 **do** Let $w_i = \sqrt{1 + \theta_i}$;
 - 4: **for** $i = 2$ to n_1 **do** Let $w_{i \rightarrow f_i} = w_i^{-1}$;
 - 5: Construct $\mathbf{G}_{\mathcal{T}^{(1)} \leftarrow \mathcal{T}}$ by w_i and $w_{i \rightarrow f_i}$;
 - 6: **return** $\mathbf{G}_{\mathcal{T}^{(1)} \leftarrow \mathcal{T}}$;
-

In summary, we propose a Generation Matrix-based optimally consistent release algorithm (GMC) for differentially private hierarchical trees, described as Alg. 2. Note that Step 1 to Step 3 in Alg. 2 are the normal hierarchical tree release process, and Step 4 is the call of Alg. 1. Only step 5 is the core step, which uses formula (36) to achieve optimally consistent release. In the previous, the time complexity of formula (36) has been proved as $O(n)$. Therefore, the overall time complexity of GMC is also $O(n)$. Besides, it can be seen that GMC is a two-stage algorithm, i.e., the construction of $\mathbf{G}_{\mathcal{T}^{(1)} \leftarrow \mathcal{T}}$ and post-processing. If the same hierarchical tree is used for multiple releases, we only need to construct $\mathbf{G}_{\mathcal{T}^{(1)} \leftarrow \mathcal{T}}$ once.

Algorithm 2 Generation Matrix-based Optimally Consistent Release Algorithm

Input: hierarchical tree \mathcal{T} , dataset \mathcal{D} , privacy parameters ε

Output: the optimally consistent release $\bar{\mathbf{v}}$

- 1: Construct a vector \mathbf{x} from \mathcal{D} , which satisfies $x_i = \phi_i(\mathbf{D})$.
 - 2: Build a hierarchical tree with $\mathbf{v} = \mathbf{G}_{\mathcal{T}}^{-T} \mathbf{H}_{\mathcal{H}}^T \mathbf{x}$;
 - 3: Calculate the height of \mathcal{T} , then get $\tilde{\mathbf{v}}$ by adding noise to \mathbf{v} , where $\tilde{v}_i = v_i + \xi_i, \xi_i \sim \text{Lap}(h/\varepsilon)$.
 - 4: Construct $\mathbf{M}_{\mathcal{T}}$, then substitute it into Alg. 1 to obtain $\mathbf{G}_{\mathcal{T}^{(1)} \leftarrow \mathcal{T}}$;
 - 5: Calculate $\bar{\mathbf{v}} = \tilde{\mathbf{v}} - \mathbf{M}_{\mathcal{T}} \left(\mathbf{G}_{\mathcal{T}^{(1)} \leftarrow \mathcal{T}}^{-1} \left(\mathbf{G}_{\mathcal{T}^{(1)} \leftarrow \mathcal{T}}^{-T} (\mathbf{M}_{\mathcal{T}}^T \tilde{\mathbf{v}}) \right) \right)$;
 - 6: **return** $\bar{\mathbf{v}}$;
-

In addition, we can further optimize the algorithm. Considering that the construction of $\mathbf{G}_{\mathcal{T}^{(1)} \leftarrow \mathcal{T}}$ involves square root extraction, which may cause more calculation overhead, we propose an improved version of Alg. 2 to avoid any square root. The main idea is shown as follows.

$$\bar{\mathbf{v}} = \tilde{\mathbf{v}} - \mathbf{M}_{\mathcal{T}} \left(\mathbf{G}_{\mathcal{T}^{(1)}}^{(\boldsymbol{\theta}', \mathbf{1})^{-1}} \left(\boldsymbol{\theta}' * \left(\mathbf{G}_{\mathcal{T}^{(1)}}^{(\boldsymbol{\theta}', \mathbf{1})^{-T}} (\mathbf{M}_{\mathcal{T}}^T \tilde{\mathbf{v}}) \right) \right) \right). \quad (39)$$

Where the vector $\boldsymbol{\theta}' = [\theta_1, \theta_2, \dots, \theta_{n_1}]^T + 1$, and “*” is Hadamard product [32]. The new version without square root extraction can slightly improve the calculation efficiency, which shows that the algorithm under the matrix description has high scalability. The improvement of the algorithm only needs to make some slight modifications on Formula (36). Furthermore, we can directly extend the existing model to solve other hierarchical tree problems, such as the hierarchical tree release with the non-uniform privacy budget.

6. Experiment

We conducted experiments to verify the performance of GMC our proposed for differentially private hierarchical tree release. All our experiments are run on a computer with Dual 4 Core 3.9 GHz AMD Ryzen CPUs, 32GB RAM, and MATLAB’s development software. To improve the reliability of our experimental results, we repeatedly run the same experimental setup 100 times and then take the average of multiple experimental results as the final results. In addition, we denote the output of our algorithm by $\mathbf{v}^{(out)}$ or $\mathbf{x}^{(out)}$, $v_i^{(out)}$, and their i -th outputs are denoted as $v_i^{(out)}$ and $x_i^{(out)}$, respectively.

6.1. Datasets and Comparison Algorithms

Our experiments run on 3 large datasets with more than 10 million nodes. They are Census2010, NYCTaxi, and SynData, with the following details.

Census2010[33]: Due to the plan that the U.S. Census Bureau announced using differential privacy[2], we adopt the 2010 U.S. Census dataset as our first dataset, which contains demographic information of all Americans. The statistical results are divided into 8 levels by the geographic components, i.e., “United States - State - County - County Subdivision - Place/Remainder - Census Tract - Block Group - Block”. The dataset contains 312, 471, 327 individuals, which construct a hierarchical tree containing 11, 802, 162 nodes. One of its typical applications is to provide users with queries on the population of the area of interest, called “Node Query”. For example, a user submits a query request “What is the population of Albany County in New York, USA?”. The system will return the value at the node “USA - New York - Albany County”. After verification, we ensured that the data before adding noise satisfies consistency.

NYCTaxi[34]: This data set comes from car ride records in New York City in 2013. In order to ensure the uniqueness of the data, we selected the data provided by Creative Mobile Technologies. According to the start time of the taxi, we constructed a statistical histogram of the frequency of people taking a taxi in seconds and provided a range query. The process of building a hierarchical tree is random, which ensures that our algorithm can handle any hierarchical tree structure. The data contains 86, 687, 775 individuals. Since we count the ride frequency by seconds, the number of histograms (leaf nodes) totals 31, 536, 000. The fan-out of nodes is random. In our experiments, we set the proportion of nodes with fan-outs of 2, 3, 4 and 5 to {40%, 30%, 20%, 10%}. In this way, the hierarchical tree we build contains approximately 45 to 50 million nodes.

SynData: To test the hierarchical tree with special structures, we adopt a randomly synthesized dataset for our experiments. In synthetic data, the hierarchical tree structure is a complete binary, and the number of nodes n is controlled by the tree height h , where $n = 2^h - 1$. We generate x_i by Poisson distribution, i.e., $x_i \sim \text{Poi}(\lambda)$. In our experiments, we set $\lambda = 100$ and take $h = 24$ as the complete dataset, containing 16, 777, 215 nodes. Like NYCTaxi, we focus on the “Range Query” on SynData.

Table 2: Description of the Algorithms

Name	Description
Processless	No processing after adding noise.
Boosting	Boosting[13], the classic post-processing of only for the complete trees. For arbitrary structure, we take the fan-out of root as the fan-out of the algorithm.
PrivTrie	The post-processing in PrivTrie[14], which has been proven to achieve the optimally consistent release for arbitrary hierarchical tree.
GMC	Our post-processing for arbitrary hierarchical tree, which achieve the optimally consistent release based on Generation Matrix.

Our experiments compare 4 different algorithm settings. Their details are shown in Tab. 2. Because our experiment is for large-scale hierarchical trees with more than 10 million nodes, the time complexity of the selected algorithm settings is both $O(n)$.

6.2. Verifying the Effectiveness for GMC

In the first experiment, we verify the effectiveness of GMC with two main aspects, i.e., error and consistency.

The error is measured by the root mean square error (*rmse*), whose calculation methods are different for “Node Query” and “Range Query”. We denote the *rmse* of “Node Query” (For Census2010) by $rmse_{NQ}$, and its formula is

$$rmse_{NQ} = \sqrt{\frac{1}{n} \sum_{i=1}^n (v_i^{(out)} - v_i)^2}. \quad (40)$$

And the *rmse* of “Range Query” (For NYCTaxi and SynData) is denoted as $rmse_{RQ}$, calculated by

$$rmse_{RQ} = \sqrt{\frac{1}{q} \sum_{i=1}^q \left(\sum_{j=a_i}^{b_i} x_i^{(out)} - \sum_{j=a_i}^{b_i} x_j \right)^2}. \quad (41)$$

Where q is the number of range queries selected randomly and without repetition, and the range of the i -th query is recorded as $[a_i, b_i]$. The main reason for random sampling is that all range queries are up to C_m^2 so that we cannot test all range queries. In our experiment, we take $q = 10^5$.

The consistency of the outputs is measured by consistency bias. Let $\Delta = \mathbf{M}^T \mathbf{v}^{(out)}$, and Δ_i as the i -th element of Δ , then *bias* satisfies

$$bias = \sqrt{\sum_{i=1}^{n_1} \Delta_i^2 / n_1}. \quad (42)$$

Besides, we adopt the complete datasets in the experiment and take $\varepsilon = 1$.

Table 3: Experimental Results on Multiple Algorithm Settings and Large-scale Datasets

	Census2010		NYCTaxi		SynData	
	<i>rmse</i>	<i>bias</i>	<i>rmse</i>	<i>bias</i>	<i>rmse</i>	<i>bias</i>
Processless	11.32	49.61	138.42	48.38	166.37	61.24
Boosting	11.30	159.01	120.75	22.42	74.94	0.00
PrivTrie	11.00	0.00	68.69	0.00	74.94	0.00
GMC	11.00	0.00	68.69	0.00	74.94	0.00

Since the optimally consistent release problem of the differentially private hierarchical tree is a convex optimization problem, its solution is unique. If the outputs of GMC also satisfy optimally consistent, they should be the same as PrivTrie. The results in Tab. 3 confirm this point. They show that GMC is effective and correct. However, it does not mean that PrivTrie and GMC are entirely equivalent. Their implementations are entirely different, so we need further to analyze the algorithms’ performance in the subsequent experiments.

In addition, the results also show the major drawback of Boosting. i.e., it can only guarantee the consistency of complete trees. If the fan-outs of nodes are different, Boosting cannot guarantee that the results are consistent. Especially for Census2010, where the fan-out of nodes is highly different, Boosting results in more significant consistency bias after post-processing.

6.3. Performance Testing

In this section, we focus on the algorithms’ performance and test the running time of the algorithms above from small-scale data to large-scale data. To construct hierarchical trees with different scales, we adopted the following different methods according to the characteristics of each dataset. For Census2010, we obtain a smaller hierarchical tree by k -order subtree. For example, in its 4-order subtree, the leaves represent the “County Subdivision” level. For NYC taxi, we divide the data of 2013 into 12 months. The data of the first k months as the k -th subset. For SynData, we control the data scale by directly setting the tree height. In the experiment, we used the six different tree heights {8, 12, 16, 20, 24} to generate hierarchical trees with different scales. Since Processless does not perform any processing for consistency, we omit it in the experiment.

In Fig. 6, the experimental results show that all three algorithms can complete the post-processing within some time, but their running times are quite different. Boosting and PrivTrie need more than 200 seconds to process the

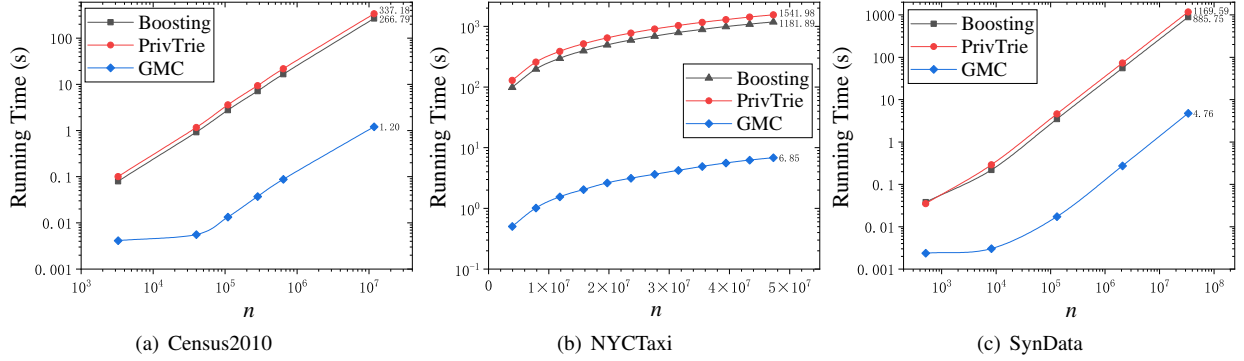


Figure 6: The Comparison of Running Time for Various Different Data Scales

hierarchical trees with tens of millions of nodes, while GMC only needs about 2 seconds to process the same data (See Fig. 6a). Their performance gap is up to 100 times. It shows that even if Boosting and PrivTrie have reached the lowest time complexity with $O(n)$, they still have much space for performance improvement. In addition to the relatively inefficient recursive algorithms, another important reason is that GMC uses standard matrix operations to complete the post-processing after establishing the matrix model. These standard matrix operations introduce many optimization techniques in the underlying design, which can make full use of computer resources and significantly improve computing efficiency. Although the results above do not deny that Boosting and PrivTrie can handle large-scale hierarchical trees, applying them to some scenarios, such as real-time data updates or more complex models, may cause considerable challenges in computational efficiency.

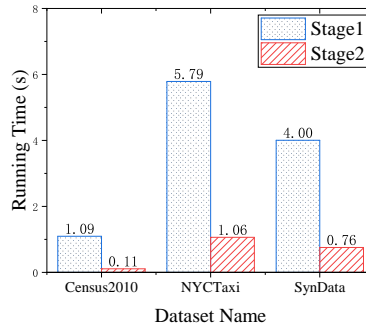


Figure 7: Running Time for Two Stages of GMC with Complete Datasets

Finally, we test the running time for the two stages of GMC, i.e., the Generation Matrix construction (Stage 1) and the post-processing (Stage 2). In Fig. 7, we can see that the time overhead in Stage 1 is much more significant than in Stage 2. The reason is the Generation Matrix construction includes the sorting by Descending Order of Height, counting the number of children, and some high-cost operations such as division, square root (in the process of calculating θ_i). It is worth noting that the Generation Matrix construction is independent of the data to be released and does not involve individual privacy. When the same hierarchical tree structure is reused in multiple releases, we only need to perform the construction process once, further reducing the overall running time of the optimally consistent releases.

7. Conclusions and Future Work

In the previous, we successfully defined Generation Matrix and demonstrated many of its critical mathematical properties. Using Generation Matrix, we can implement various hierarchical tree operations without accessing the

local structure, which provides crucial theoretical support for the Matrixing Research Method of hierarchical trees. The application on the differentially private hierarchical tree release reflects the Generation Matrix's practicability. The proposed GMC based on Generation Matrix provides a concise algorithm for optimally consistent release. Our experiments show that GMC has achieved a significant performance improvement of up to 100 times compared with the state-of-the-art schemes.

The scientific problem that we solve in this paper is not very complicated, but it is classic and is suitable as an example to show the practicability of the Generation Matrix. However, the hierarchical tree problems that Generation Matrix can solve are far more than the application. We can use it to explore more complex hierarchical tree release problems and even the problems that have not been solved so far. For example, the issue of non-negative consistent release of hierarchical trees currently does not exist any closed-form solution with time complexity of $O(n)$, which makes it very difficult to solve the optimal release satisfying both consistency and non-negativity for large-scale data. Nonetheless, Generation Matrix provides us with a critical analysis tool to challenge the problem.

Appendix A. Partial Proofs

Appendix A.1. Proof of Property 7

Proof. According to the basic properties of the lower triangular matrix[24], the inverse $\mathbf{G}_{\mathcal{T}}^{-1}$ of the lower triangular matrix $\mathbf{G}_{\mathcal{T}}$ is also a lower triangular matrix.

Considering $\mathbf{G}_{\mathcal{T}}^{-1}$'s j -th column vector $\mathbf{g}_j^{(-1)} = \mathbf{G}_{\mathcal{T}}^{-1}\mathbf{e}_j$, whose i -th element is denoted as $g_{ij}^{(-1)}$, we have $g_{ij}^{(-1)} = 0, i < j$. Since only the j -th element of \mathbf{e}_j is 1, but the rest are all 0, $g_{ij}^{(-1)}$, for $i \geq j$, satisfies

$$g_{ij}^{(-1)} = \begin{cases} 1 & , i = j \\ g_{f_j, j}^{(-1)} & , i > j \end{cases} . \quad (\text{A.1})$$

Therefore, $g_{jj}^{(-1)} = 1$. Next, we adopt the contradiction method to prove. Suppose j is the ancestor of i , but $g_{ij}^{(-1)} \neq 1$.

Let $t_0^{(i)}, t_1^{(i)}, \dots, t_p^{(i)}$ denote the node i and its p ancestors respectively.

Since j is an ancestor of i , there is a q such that $j = t_q^{(i)}$. Therefore, we have $g_{jj}^{(-1)} = g_{t_q^{(i)}, j}^{(-1)} = \dots = g_{t_1^{(i)}, j}^{(-1)} = g_{ij}^{(-1)} \neq 1$ according to the recurrence formula (A.1).

The conclusion contradicts $g_{jj}^{(-1)} = 1$. Therefore, when j is an ancestor of i , $g_{ij}^{(-1)} = 1$.

If j is not an ancestor of i , obviously j is not the root, i.e., $j > 1$. According to the formula (A.1), we have $g_{1,j}^{(-1)} = g_{t_p^{(i)}, j}^{(-1)} = \dots = g_{t_1^{(i)}, j}^{(-1)} = g_{ij}^{(-1)}$. Since $g_{1,j}^{(-1)}, j > 1$ has been proved to be 0, thus $g_{ij}^{(-1)} = 0$. \square

Appendix A.2. Proof of Property 9

Proof. Considering the i -th row of $\mathbf{G}_{\mathcal{T}}^{-1}$, we denote $U_i = \{k \mid g_{ik}^{(-1)} \neq 0\}$ as the set formed by the column subscripts of the non-zero elements in the i -th row. According to $\mathbf{M} = (\mathbf{G}_{\mathcal{T}}^T \mathbf{G}_{\mathcal{T}})^{-1} = \mathbf{G}_{\mathcal{T}}^{-1} \mathbf{G}_{\mathcal{T}}^{-T}$, we have m_{ij} satisfied

$$m_{ij} = \sum_{k=1}^n g_{ik}^{(-1)} g_{jk}^{(-1)} = \sum_{k \in U_i \cap U_j} g_{ik}^{(-1)} g_{jk}^{(-1)} = |U_i \cap U_j| \quad (\text{A.2})$$

According to Prop. 7, $U_i = \{k \mid k \text{ is an ancestor of } i\}$. Therefore, $U_i \cap U_j$ is the common ancestor of i and j and m_{ij} records the number of their common ancestors.

Specifically, if $i = j$, $m_{ii} = |U_i|$, i.e., the number of ancestors of i plus 1 (itself). Let the depth of the root be 1, m_{ii} is the depth of i . \square

Appendix A.3. Proof of Property 11

Proof. Let $\alpha = [\alpha_1, \alpha_2, \dots, \alpha_n]^T$, $\beta = [\beta_1, \beta_2, \dots, \beta_n]^T$. According to Equation (19), we have $w_i = \alpha_i \beta_i$, $w_{i \rightarrow f_i} = \alpha_{f_i} \beta_i$. That is, $\ln w_i = \ln \alpha_i + \ln \beta_i$, $\ln w_{i \rightarrow f_i} = \alpha_{f_i} + \beta_i$. Therefore, we have the following matrix equation holds.

$$\begin{bmatrix} \mathbf{I} & \mathbf{I} \\ \mathbf{I} - \mathbf{G} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \ln \alpha \\ \ln \beta \end{bmatrix} = \begin{bmatrix} \ln \mathbf{w}_{node} \\ \ln \mathbf{w}_{edge} \end{bmatrix}. \quad (\text{A.3})$$

According to the property of Block Matrix Inversion, we have

$$\begin{bmatrix} \mathbf{I} & \mathbf{I} \\ \mathbf{I} - \mathbf{G} & \mathbf{I} \end{bmatrix}^{-1} = \begin{bmatrix} \mathbf{G}^{-1} & -\mathbf{G}^{-1} \\ \mathbf{I} - \mathbf{G}^{-1} & \mathbf{G}^{-1} \end{bmatrix}. \quad (\text{A.4})$$

And then,

$$\begin{bmatrix} \ln \alpha \\ \ln \beta \end{bmatrix} = \begin{bmatrix} \mathbf{G}_{\mathcal{T}}^{-1} (\ln \mathbf{w}_{node} - \ln \mathbf{w}_{edge}) \\ \ln \mathbf{w}_{node} + \mathbf{G}_{\mathcal{T}}^{-1} (\ln \mathbf{w}_{edge} - \ln \mathbf{w}_{node}) \end{bmatrix} = \begin{bmatrix} \ln \alpha \\ \ln \mathbf{w}_{node} - \ln \alpha \end{bmatrix}. \quad (\text{A.5})$$

Performing the exponential operations $\exp(*)$ on both sides of the equation (A.5) at the same time, we have formula (20) holds. \square

Appendix A.4. Proof of Theorem 2

Proof. Let l_{ij} denote the element in row i and column j of $\mathbf{L}_{\mathcal{T}}$. According to the definition of Laplacian Matrix[6], we have the diagonal elements l_{ii} representing the number of adjacent nodes of i ; and if j is the adjacent node of i , i.e., $j = f_i$ or $i = f_j$, we have $l_{ij} = -1$; otherwise, $l_{ij} = 0$.

According to the expression (22), we have

$$l_{ij} = \begin{cases} \left(\sum_{k \in U_1} g_{k1} g_{k1} \right) - 1 & , i = j = 1 \\ \sum_{k \in U_i \cap U_j} g_{ki} g_{kj} & , \text{otherwise} \end{cases} \quad (\text{A.6})$$

where $U_j = \{k | g_{kj} \neq 0\} = \{j\} \cup C_j$ denotes the set formed by the row subscripts of non-zero elements in the j -th column of $\mathbf{G}_{\mathcal{T}}$.

Next, we discuss the following situations:

For $i = j = 1$, under the definition of Laplacian Matrix, l_{11} should be equal to the number of children of the root, i.e., $l_{11} = |C_1|$. According to the expression (22), there is $k \in \{1\} \cup C_1$ and $l_{11} = |\{1\} \cup C_1| - 1 = |C_1|$. It conforms to the definition of Laplacian Matrix.

For $i = j > 1$, under the definition of Laplacian Matrix, l_{ii} should be equal to the number of i 's children and parent, i.e., $l_{ii} = |C_i \cup f_i| = |C_i| + 1$. According to the expression (22), $l_{ii} = \sum_{k \in U_i} g_{ki} g_{ki} = |\{i\} \cup C_i| = |C_i| + 1$. It conforms to the definition of Laplacian Matrix.

For $i \neq j$ but $i \in C_j$, under the definition of Laplacian Matrix, $l_{ij} = -1$. According to the expression (22), $k \in U_i \cap U_j = (\{i\} \cup C_i) \cap (\{j\} \cup C_j) = \{i\} \cap C_j = \{i\}$, we have $l_{ij} = g_{ii} g_{ij} = -1$. Again, It conforms to the definition of Laplacian Matrix. Similarly, if $i \neq j$ and $j \in C_i$, $l_{ij} = -1$. Finally, when $i \neq j$ and i do not contain a parent-child relationship, $U_i \cap U_j = \emptyset$, $l_{ij} = 0$, which also conforms to the definition of Laplacian Matrix.

Therefore, in any case, the expression(22) always conforms to the definition of the Laplacian Matrix. \square

Appendix A.5. Proof of Theorem 3

Proof. Let d_{ij} denote the element in row i and column j of $\mathbf{D}_{\mathcal{T}}$.

According to the definition of distance matrix[7], d_{ij} is the distance from node i to j . As shown in Fig. A.8, c (red node) is the nearest common ancestor of nodes i and j . d_c denotes the depth of node c (i.e., the distance from node C to the root (orange node) plus 1); d_1 denotes the distance from c to i ; d_2 denotes the distance from c to j . Obviously, the distance $d_{ij} = d_1 + d_2$.

Considering expression (23), let $\mathbf{M}_1 = \mathbf{G}_{\mathcal{T}}^{-1} \mathbf{H}^T$, $\mathbf{M}_2 = \mathbf{H}^T \mathbf{G}_{\mathcal{T}}^{-T}$, $\mathbf{M}_2 = \mathbf{H}^T \mathbf{G}_{\mathcal{T}}^{-T}$ and $m_{ij}^{(k)}$ denotes the element in the row i and column j of \mathbf{M}_k . According to Prop. 6, $m_{ij}^{(1)} = m_{ji}^{(2)}$ is the depth of the node i , i.e., $m_{ij}^{(3)} = d_c$. According to Prop. 9, $m_{ij}^{(3)}$ is the number of common ancestors of i and j (the depth of c), i.e., $m_{ij}^{(3)} = d_c$.

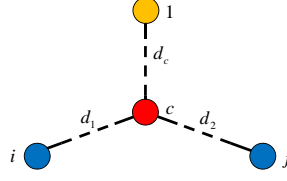


Figure A.8: The Distance of Node i, j and Their Nearest Common Ancestor

Substituting them into expression (23), we have

$$d_{ij} = m_{ij}^{(1)} + m_{ij}^{(2)} - 2m_{ij}^{(3)} = (d_1 + d_c) + (d_2 + d_c) - 2d_c = d_1 + d_2 \quad (\text{A.7})$$

The expression (A.7) is consistent with the definition, so $\mathbf{D}_{\mathcal{T}}$ can be calculated by expression (23). \square

Appendix A.6. Proof of Theorem 4

Proof. Let c_{ij} denote the element in row i and column j of $\mathbf{C}_{\mathcal{T}}$.

According to the definition of Ancestral Matrix[8], c_{ij} represents the distance from the nearest common ancestor of leaves i and j to the root.

Note, Prop. 9 has been proved that the element of $(\mathbf{G}_{\mathcal{T}}^T \mathbf{G}_{\mathcal{T}})^{-1}$ is the number of the common nodes, i.e., the distance from the nearest common ancestor to the root node plus 1.

Therefore, we can get $\mathbf{C}_{\mathcal{T}}$ by taking the sub-matrix corresponding to the leaves in $(\mathbf{G}_{\mathcal{T}}^T \mathbf{G}_{\mathcal{T}})^{-1}$ and then subtracting 1. \square

Appendix A.7. Proof of Theorem 5

Proof. According to formula (5), the noise we add to each element in $\tilde{\mathbf{v}}$ is i.i.d, and satisfies $\text{Lap}(h/\varepsilon)$. Since the variance of $\text{Lap}(b)$ is $2b^2$, the covariance matrix of $\tilde{\mathbf{v}}$ is $D(\tilde{\mathbf{v}}) = 2(h^2/\varepsilon^2) \mathbf{I}$.

According to the mean square error analysis of differential privacy, we have

$$\text{mse}(\tilde{\mathbf{v}}) = \text{trace}(D(\tilde{\mathbf{v}})) = 2(h^2/\varepsilon^2) \text{trace}(\mathbf{I}) = 2nh^2/\varepsilon^2 \quad (\text{A.8})$$

In addition, according to formula (8),

$$D(\tilde{\mathbf{v}}) = D\left(\left(\mathbf{I} - \mathbf{M}_{\mathcal{T}}(\mathbf{M}_{\mathcal{T}}^T \mathbf{M}_{\mathcal{T}})^{-1} \mathbf{M}_{\mathcal{T}}^T\right) \tilde{\mathbf{v}}\right) = 2(h^2/\varepsilon^2) \left(\mathbf{I} - \mathbf{M}_{\mathcal{T}}(\mathbf{M}_{\mathcal{T}}^T \mathbf{M}_{\mathcal{T}})^{-1} \mathbf{M}_{\mathcal{T}}^T\right) \quad (\text{A.9})$$

Then,

$$\begin{aligned} \text{mse}(\tilde{\mathbf{v}}) &= \text{trace}(D(\tilde{\mathbf{v}})) = 2(h^2/\varepsilon^2) \text{trace}\left(\mathbf{I} - \mathbf{M}_{\mathcal{T}}(\mathbf{M}_{\mathcal{T}}^T \mathbf{M}_{\mathcal{T}})^{-1} \mathbf{M}_{\mathcal{T}}^T\right) \\ &= 2(h^2/\varepsilon^2) \left(n - \text{trace}\left(\mathbf{M}_{\mathcal{T}}^T \mathbf{M}_{\mathcal{T}} (\mathbf{M}_{\mathcal{T}}^T \mathbf{M}_{\mathcal{T}})^{-1}\right)\right). \end{aligned} \quad (\text{A.10})$$

Since $\mathbf{M}_{\mathcal{T}} \in \mathbb{R}^{n \times n_1}$, we have

$$\text{trace}\left(\mathbf{M}_{\mathcal{T}}^T \mathbf{M}_{\mathcal{T}} (\mathbf{M}_{\mathcal{T}}^T \mathbf{M}_{\mathcal{T}})^{-1}\right) = \text{trace}(\mathbf{I}) = n_1. \quad (\text{A.11})$$

Substituting it into (A.10), we have

$$\text{mse}(\tilde{\mathbf{v}}) = 2(h^2/\varepsilon^2) (n - n_1) = 2mh^2/\varepsilon^2 \quad (\text{A.12})$$

\square

Appendix A.8. Proof of Theorem 6

Proof. By Def. 10, it is obvious that the process of Householder transformation satisfies property c). Because the process of “LO”–QR decomposition is the process of setting 0 column by column from the last column to the first column.

According to Def. 5, $\mathbf{M}_{\mathcal{T}}^{(0)}$ satisfies both properties a), b) and c).

Assume that \mathbf{M} satisfies both properties a), b) and c). By the formula (31), the expression of the k -th Householder transformation is as follows:

$$\mathbf{M}_{\mathcal{T}}^{(k)} = \text{House}_{S_k, t_k}(\mathbf{M}_{\mathcal{T}}^{(k-1)}), \quad (\text{A.13})$$

where $t_k = n_1 - k + 1$. Considering that $\mathbf{M}_{\mathcal{T}}^{(k-1)}$ satisfies the property a) and is a lower triangular matrix, the values of S_k can be simplified, taking $S_k = \langle t_k, n_1 + 1, \dots, n \rangle$.

According to Def. 11, during the transformation process, \mathbf{m}_k and ω_k satisfy

$$\begin{cases} \mathbf{m}_k = [m_{t_k, t_k}^{(k-1)}, m_{n_1+1, t_k}^{(k-1)}, m_{n_1+2, t_k}^{(k-1)}, \dots, m_{n, t_k}^{(k-1)}]^T \\ \omega_k = [m_{t_k, t_k}^{(k-1)} - \rho, m_{n_1+1, t_k}^{(k-1)}, m_{n_1+2, t_k}^{(k-1)}, \dots, m_{n, t_k}^{(k-1)}]^T \end{cases}, \quad (\text{A.14})$$

where $\rho_k = \|\mathbf{m}_k\| = \sqrt{\sum_{s \in S_k} m_{s, t_k}^{(k)2}}$. By calculating $\omega_k^T \mathbf{m}_k$, we have

$$\omega_k^T \mathbf{m}_k = (m_{t_k, t_k}^{(k-1)} - \rho_k) m_{t_k, j}^{(k-1)} + \sum_{s=n_1+1}^n m_{s, t_k}^{(k-1)} m_{s, j}^{(k-1)}. \quad (\text{A.15})$$

Due to $\mathbf{M}_{\mathcal{T}}^{(k-1)}$ satisfies the property b), for the same row $s > n_1$, at most only one of $m_{s, t_k}^{(k-1)}$ and $m_{s, j}^{(k-1)}$ in $\mathbf{M}_{\mathcal{T}}^{(k-1)}$ is non-zero. Therefore,

$$\omega_k^T \mathbf{m}_k = (m_{t_k, t_k}^{(k-1)} - \rho_k) m_{t_k, j}^{(k-1)}. \quad (\text{A.16})$$

Simplifying according to formula (30), we can get $\mathbf{M}_{\mathcal{T}}^{(k)}$ after k -th Householder transformation, whose element $m_{i, j}^{(k)}$ in row i and column j satisfies

$$m_{i, j}^{(k)} = \begin{cases} \rho_k & , i = t_k, j = t_k \\ 0 & , n_1 + 1 \leq i \leq n, j = t_k \\ m_{t_k, j}^{(k-1)} m_{t_k, t_k}^{(k-1)} / \rho_k & , i = t_k, j \neq t_k \\ m_{i, j}^{(k-1)} + m_{i, t_k}^{(k-1)} m_{t_k, j}^{(k-1)} / \rho_k & , n_1 + 1 \leq i \leq n, j \neq t_k \\ m_{i, j}^{(k-1)} & , \text{otherwise} \end{cases} \quad (\text{A.17})$$

According to the recursive expression (A.17), consider the properties of $\mathbf{M}_{\mathcal{T}}^{(k)}$. First, consider the property a). Since $\mathbf{M}_{\uparrow}^{(k-1)} \sim \mathbf{G}_{\mathcal{T}^{(1)}}$, and only the row $t_k = n_1 - k + 1$ in the first n_1 rows of $\mathbf{M}_{\mathcal{T}}^{(k-1)}$ is affected, according to the sparsity of GM, it can be known that the t_k -th row of $\mathbf{M}_{\mathcal{T}}^{(k-1)}$ satisfies $m_{t_k, j}^{(k-1)} \neq 0$ if and only if $j \in \{t_k, f_{t_k}\}$.

Since $m_{t_k, t_k}^{(k-1)} / \rho_k \neq 0$, according to $m_{i, j}^{(k)} = m_{i, j}^{(k-1)} m_{t_k, t_k}^{(k-1)} / \rho_k$ for $j \neq t_k$, we have $m_{i, j}^{(k)} \neq 0 \Leftrightarrow m_{i, j}^{(k-1)} \neq 0$.

Therefore, there is no mutual conversion between non-zero elements and zero elements in the only affected t_k -th row. $\mathbf{M}_{\mathcal{T}}^{(k)}$ satisfies property a).

Next consider property b). For the rows $n_1 + 1 \sim n$, according to the property c), all the values of the t_k -th column of $\mathbf{M}_{\downarrow}^{(k)}$ are 0. Consider the j -th column ($j \neq t_k$) of $\mathbf{M}_{\downarrow}^{(k)}$, we have

$$m_{i, j}^{(k)} = m_{i, j}^{(k-1)} + m_{i, t_k}^{(k-1)} m_{t_k, j}^{(k-1)} / \rho_k. \quad (\text{A.18})$$

If $j \neq f_{t_k}$, there is $m_{i, j}^{(k-1)} = 0$, i.e., $m_{i, j}^{(k)} = m_{i, j}^{(k-1)}$; If $j = f_{t_k}$, there is $m_{i, j}^{(k-1)} \neq 0$, i.e., $m_{i, j}^{(k)} \neq 0 \Leftrightarrow m_{i, j}^{(k-1)} \neq 0 \vee m_{i, t_k}^{(k-1)} \neq 0$.

Therefore, the k -th Householder transformation is equivalent to transferring non-zero elements from the t_k -th column of $\mathbf{M}_{\downarrow}^{(k-1)}$ to the f_{t_k} -th column of $\mathbf{M}_{\downarrow}^{(k-1)}$. The process keeps the property b) holds.

In summary, $\mathbf{M}_{\mathcal{T}}^{(k)}$ also satisfies the properties a), b) and c). Since $\mathbf{M}_{\mathcal{T}}^{(0)}$ satisfies the properties a), b) and c), all $\mathbf{M}_{\mathcal{T}}^{(k)}$ ($0 \leq k \leq n_1 - 1$) satisfy the properties a), b) and c). \square

Appendix A.9. Proof of Theorem 7

Proof. By Thm. 6, after $n_1 - 1$ Householder transformations, $\mathbf{M}_{\mathcal{T}}^{(n_1-1)}$ satisfies both the properties a), b), and c).

Consider $\mathbf{M}_{\mathcal{T}}^{(n_1)} = \text{House}_{\mathcal{S}_{n_1}, 1}(\mathbf{M}_{\mathcal{T}}^{(n_1-1)})$ for the n_1 -th Householder transformation. Due to $\mathbf{M}_{\uparrow}^{(n_1-1)} \sim \mathbf{G}_{\mathcal{T}^{(1)}}$ and only $m_{1,1}^{(n_1-1)}$ in the first row of $\mathbf{M}_{\uparrow}^{(n_1-1)}$ is non-zero, which is the only element of $\mathbf{M}_{\uparrow}^{(n_1-1)}$ affected by the last transformation. The transformed $m_{1,1}^{(n_1)}$ satisfies

$$m_{1,1}^{(n_1)} = \sqrt{m_{1,1}^{(n_1-1)2} + \sum_{s=n_1+1}^n m_{s,1}^{(n_1-1)2}}. \quad (\text{A.19})$$

Therefore, there is still $\mathbf{M}_{\uparrow}^{(n_1)} \sim \mathbf{G}_{\mathcal{T}^{(1)}}$ after the last transformation, i.e., there is a $\mathbf{G}_{\mathcal{T}^{(1)}}^{(\mathbf{w}_{\text{node}}, \mathbf{w}_{\text{edge}})}$ satisfying $\mathbf{G}_{\mathcal{T}^{(1)}}^{(\mathbf{w}_{\text{node}}, \mathbf{w}_{\text{edge}})} = \mathbf{M}_{\uparrow}^{(n_1)}$. Besides, according to Def. 11 and property c), it can be known that $\mathbf{M}_{\downarrow}^{(n_1)} = \mathbf{O}$ after the last transformation. \square

Appendix A.10. Proof of Theorem 8

Proof. Let $\mathbf{G}_{\mathcal{T}^{(1)} \leftarrow \mathcal{T}}$ be the upper half of $\mathbf{M}_{\mathcal{T}}$ after the “LO”-QR decomposition. According to Thm. 7, we have

$$\mathbf{M}_{\mathcal{T}} = \mathbf{Q} \begin{bmatrix} \mathbf{G}_{\mathcal{T}^{(1)} \leftarrow \mathcal{T}} \\ \mathbf{O} \end{bmatrix}. \quad (\text{A.20})$$

Substituting it into the expression $\mathbf{M}_{\mathcal{T}}^T \mathbf{M}_{\mathcal{T}}$, we have

$$\mathbf{M}_{\mathcal{T}}^T \mathbf{M}_{\mathcal{T}} = \begin{bmatrix} \mathbf{G}_{\mathcal{T}^{(1)} \leftarrow \mathcal{T}}^T & \mathbf{O} \end{bmatrix} \mathbf{Q}^T \mathbf{Q} \begin{bmatrix} \mathbf{G}_{\mathcal{T}^{(1)} \leftarrow \mathcal{T}} \\ \mathbf{O} \end{bmatrix} = \begin{bmatrix} \mathbf{G}_{\mathcal{T}^{(1)} \leftarrow \mathcal{T}}^T & \mathbf{O} \end{bmatrix} \begin{bmatrix} \mathbf{G}_{\mathcal{T}^{(1)} \leftarrow \mathcal{T}} \\ \mathbf{O} \end{bmatrix} = \mathbf{G}_{\mathcal{T}^{(1)} \leftarrow \mathcal{T}}^T \mathbf{G}_{\mathcal{T}^{(1)} \leftarrow \mathcal{T}}. \quad (\text{A.21})$$

\square

Appendix A.11. Proof of Corollary 2

Proof. Since $\mathbf{G}_{\mathcal{T}^{(1)} \leftarrow \mathcal{T}}$ is reversible, according to Thm. 8, substituting formula (34) into $\mathbf{y} = (\mathbf{M}_{\mathcal{T}}^T \mathbf{M}_{\mathcal{T}})^{-1} \mathbf{x}$, we have

$$\mathbf{y} = (\mathbf{M}_{\mathcal{T}}^T \mathbf{M}_{\mathcal{T}})^{-1} \mathbf{x} = \mathbf{G}_{\mathcal{T}^{(1)} \leftarrow \mathcal{T}}^{-1} (\mathbf{G}_{\mathcal{T}^{(1)} \leftarrow \mathcal{T}}^{-T} \mathbf{x}). \quad (\text{A.22})$$

\square

Appendix A.12. Proof of Theorem 9

Proof. Define a sequence θ_i ($1 \leq i \leq n_1$) satisfies

$$\theta_i = \sum_{s=n_1+1}^n m_{s,i}^{(t_i)2}, \quad (\text{A.23})$$

where $t_i = n_1 - i$. The $(t_i + 1)$ -th Householder transformation is the Householder transformation on the i -th column. And let j denote the child of i , i.e., ($j \in C_i$).

By recursive expression (A.17), in the first t_i Householder transformations, only the Householder transformation on the j -th column (i.e., the $(t_j + 1)$ -th transformation) will cause the value of $m_{s,i}$ ($n_1 + 1 \leq s \leq n$) to change. Therefore, $m_{s,i}^{(t_i)}$ satisfies

$$m_{s,i}^{(t_i)} = m_{s,i}^{(0)} + \sum_{j \in C_i \wedge j \leq n_1} \left(m_{s,j}^{(t_j)} m_{j,i}^{(t_j)} / m_{j,j}^{(t_j+1)} \right). \quad (\text{A.24})$$

For a non-leaf node i , since the values of $m_{j,i}^{(t_j)}$ and $m_{j,j}^{(t_j)}$ haven't changed in the first t_j Householder transformations, there are $m_{j,j}^{(t_j)} = 1$ and $m_{j,i}^{(t_j)} = -1$. Therefore, $m_{j,j}^{(t_j+1)}$ satisfies

$$m_{j,j}^{(t_j+1)} = \sqrt{m_{j,j}^{(t_j)2} + \sum_{s=n_1+1}^n m_{s,j}^{(t_j)2}} = \sqrt{1 + \theta_j}. \quad (\text{A.25})$$

Therefore, for $n_1 + 1 \leq s \leq n$, we have

$$m_{s,i}^{(t_i)} = m_{s,i}^{(0)} - \sum_{j \in C_i \wedge j \leq n_1} m_{s,j}^{(t_j)} / \sqrt{1 + \theta_j}. \quad (\text{A.26})$$

According to formula (A.23), θ_i satisfies

$$\theta_i = \sum_{s=n_1+1}^n m_{s,i}^{(t_i)^2} = \sum_{s=n_1+1}^n \left(m_{s,i}^{(0)} - \sum_{j \in C_i \wedge j \leq n_1} m_{s,j}^{(t_j)} / \sqrt{1 + \theta_j} \right)^2. \quad (\text{A.27})$$

According to Thm. 6, the process of “LO”–QR decomposition always satisfies the property b), so at most only one item in $m_{s,i}^{(0)} - \sum_{j \in C_i \wedge j \leq n_1} m_{s,j}^{(t_j)} / \sqrt{1 + \theta_j}$ is non-zero, and we have

$$\theta_i = \left(m_{s,i}^{(0)} - \sum_{j \in C_i \wedge j \leq n_1} m_{s,j}^{(t_j)} / \sqrt{1 + \theta_j} \right)^2 = m_{s,i}^{(0)^2} + \sum_{j \in C_i \wedge j \leq n_1} m_{s,j}^{(t_j)^2} / (1 + \theta_j). \quad (\text{A.28})$$

Therefore,

$$\begin{aligned} \theta_i &= \sum_{s=n_1+1}^n \left(m_{s,i}^{(0)} - \sum_{j \in C_i \wedge j \leq n_1} m_{s,j}^{(t_j)} / \sqrt{1 + \theta_j} \right)^2 = \sum_{s=n_1+1}^n m_{s,i}^{(0)^2} + \sum_{j \in C_i \wedge j \leq n_1} \left(\sum_{s=n_1+1}^n m_{s,j}^{(t_j)^2} / (1 + \theta_j) \right) \\ &= \sum_{s=n_1+1}^n m_{s,i}^{(0)^2} + \sum_{j \in C_i \wedge j \leq n_1} \theta_j / (1 + \theta_j) = \sum_{s=n_1+1}^n m_{s,i}^{(0)^2} + \sum_{j \in C_i \wedge j \leq n_1} \left(1 - (1 + \theta_j)^{-1} \right). \end{aligned} \quad (\text{A.29})$$

According to the definition of \mathbf{M}_\downarrow , for $n_1 + 1 \leq s \leq n$, if $s \in C_i$, then $m_{s,i}^{(0)} = -1$, otherwise $m_{s,i}^{(0)} = 0$. Finally, θ_i is reduced to

$$\theta_i = \sum_{j \in C_i \wedge j > n_1} 1 + \sum_{j \in C_i \wedge j \leq n_1} 1 - \sum_{j \in C_i \wedge j \leq n_1} 1 / (1 + \theta_j) = \sum_{j \in C_i} 1 - \sum_{j \in C_i \wedge j \leq n_1} 1 / (1 + \theta_j) = |C_i| - \sum_{j \in C_i \wedge j \leq n_1} 1 / (1 + \theta_j). \quad (\text{A.30})$$

According to formula (A.25), we have

$$w_i = m_{i,i}^{(t_i+1)} = \sqrt{1 + \theta_i}. \quad (\text{A.31})$$

Then, according to formula (A.17), we have

$$w_{i \rightarrow f_i} = -m_{f_i,i}^{(t_i+1)} = -m_{i,f_i}^{(t_i)} m_{i,i}^{(t_i)} / m_{i,i}^{(t_i+1)} = -m_{i,f_i}^{(0)} m_{i,i}^{(0)} / m_{i,i}^{(t_i+1)} = -(-1 \times 1) / w_i = w_i^{-1}. \quad (\text{A.32})$$

□

References

- [1] S. Niazi, M. Ismail, S. Grohsschmiedt, M. Ronstrm, S. Haridi, J. Dowling, Hopsfs: Scaling hierarchical file system metadata using newsq databases.
- [2] J. Abowd, The u.s. census bureau adopts differential privacy, 2018, pp. 2867–2867. doi:10.1145/3219819.3226070.
- [3] T. Lima, M. de Aguiar, Laplacian matrices for extremely balanced and unbalanced phylogenetic trees (08 2020).
- [4] S. Garfinkel, J. Abowd, S. Powazek, Issues encountered deploying differential privacy (09 2018). doi:10.1145/3267323.3268949.
- [5] X. Li, Z. Wang, Trees with extremal spectral radius of weighted adjacency matrices among trees weighted by degree-based indices, Linear Algebra and its Applications 620. doi:10.1016/j.laa.2021.02.023.
- [6] S. Ganesh, S. Mohanty, Trees with matrix weights: Laplacian matrix and characteristic-like vertices (09 2020).
- [7] R. Bapat, S. Sivasubramanian, Squared distance matrix of a tree: Inverse and inertia, Linear Algebra and its Applications 491. doi:10.1016/j.laa.2015.09.008.
- [8] E. Andriantiana, K. Dadedzi, S. Wagner, The ancestral matrix of a rooted tree, Linear Algebra and Its Applications 575 (2019) 35–65. doi:10.1016/j.laa.2019.04.004.
- [9] W. Fulton, J. Harris, Graduate texts in mathematics, Representation Theory. A First Course, Readings in Mathematics 129.

- [10] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, M. Sun, Graph neural networks: A review of methods and applications, *AI Open* 1 (2020) 57–81. doi:10.1016/j.aiopen.2021.01.001.
- [11] X. A. Chen, Understanding spectral graph neural network.
- [12] M. Deveci, C. Trott, S. Rajamanickam, Multi-threaded sparse matrix-matrix multiplication for many-core and gpu architectures, *Parallel Computing* 78. doi:10.1016/j.parco.2018.06.009.
- [13] M. Hay, V. Rastogi, G. Miklau, D. Suciu, Boosting the accuracy of differentially-private histograms through consistency, *Proceedings of the VLDB Endowment* 3. doi:10.14778/1920841.1920970.
- [14] N. Wang, X. Xiao, Y. Yang, T. Hoang, H. Shin, J. Shin, G. Yu, Privtrie: Effective frequent term discovery under local differential privacy, 2018, pp. 821–832. doi:10.1109/ICDE.2018.00079.
- [15] J. Jansson, K. Sadakane, W.-K. Sung, Ultra-succinct representation of ordered trees with applications, *J. Comput. Syst. Sci.* 78 (2012) 619–631. doi:10.1016/j.jcss.2011.09.002.
- [16] D. Tsur, Succinct representation of labeled trees, *Theoretical Computer Science* 562. doi:10.1016/j.tcs.2014.10.006.
- [17] A. Farzan, J. Munro, Succinct representation of dynamic trees, *Theor. Comput. Sci.* 412 (2011) 2668–2678. doi:10.1016/j.tcs.2010.10.030.
- [18] C. Dwork, F. McSherry, K. Nissim, A. Smith, Calibrating noise to sensitivity in private data analysis, *Journal of Privacy and Confidentiality* 7 (2017) 17–51. doi:10.29012/jpc.v7i3.405.
- [19] W. Qardaji, W. Yang, N. Li, Understanding hierarchical methods for differentially private histograms, *Proceedings of the VLDB Endowment* 6 (2013) 1954–1965. doi:10.14778/2556549.2556576.
- [20] G. Cormode, M. Procopiuc, E. Shen, D. Srivastava, T. Yu, Differentially private spatial decompositions, *Computing Research Repository - CORR* doi:10.1109/ICDE.2012.16.
- [21] S. Yuan, D. Pi, X. Zhao, M. Xu, Differential privacy trajectory data protection scheme based on r-tree, *Expert Systems with Applications* 182 (2021) 115215. doi:10.1016/j.eswa.2021.115215.
- [22] J. Lee, Y. Wang, D. Kifer, Maximum likelihood postprocessing for differential privacy under consistency constraints, 2015, pp. 635–644. doi:10.1145/2783258.2783366.
- [23] G. Strang, *Linear algebra and learning from data*, Wellesley-Cambridge Press Cambridge, 2019.
- [24] G. Birkenmeier, H. Heatherly, J. Kim, J. Park, Triangular matrix representations, *Journal of Algebra* 230 (2000) 558–595. doi:10.1006/jabr.2000.8328.
- [25] L. Minah, A. Fox, G. Sanders, Rounding error analysis of mixed precision block householder qr algorithms, *SIAM Journal on Scientific Computing* 43 (2021) A1723–A1753. doi:10.1137/19M1296367.
- [26] P. Desai, S. Aslan, J. Saniie, Fpga implementation of gram-schmidt qr decomposition using high level synthesis, 2017, pp. 482–487. doi:10.1109/EIT.2017.8053410.
- [27] W. Fam, A. Alimohammad, Givens rotation-based qr decomposition for mimo systems, *IET Communications* 11. doi:10.1049/iet-com.2016.0789.
- [28] R. Stanley, *Enumerative combinatorics — volume 1*.
- [29] M. C. M. incorporates LAPACK, Increasing the speed and capabilities of matrix computation, [Online] (2000).
- [30] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, L. Kaiser, M. Kudlur, J. Levenberg, X. Zheng, *Tensorflow : Large-scale machine learning on heterogeneous distributed systems* (01 2015).
- [31] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, T. Darrell, Caffe: Convolutional architecture for fast feature embedding, *MM 2014 - Proceedings of the 2014 ACM Conference on Multimedia* doi:10.1145/2647868.2654889.
- [32] J. Magnus, *Matrix differential calculus with applications in statistics and econometrics* doi:10.1002/9781119541219.
- [33] U. C. Bureau, 2010 census summary file 1, <https://www.census.gov/prod/cen2010/doc/sf1.pdf> (2012).
- [34] New york city taxi data, <http://www.nyc.gov/html/tlc/html/about/triprecord/data.shtml>.