Privacy-aware Data Sharing in a Tree-based Categorical Clustering Algorithm [⋆]

Mina Sheikhalishahi[1,2], Mohamed Mejri[1], Nadia Tawbi[1] and Fabio Martinelli[2]

[1] Department of Computer science,Université Laval, Québec City, Canada,
{mohamed.mejri,nadia.tawbi}@ift.ulaval.ca,
[2] Istituto di Informatica e Telematica, Consiglio Nazionale delle ricerche, Pisa, Italy
{mina.sheikhalishahi,fabio.martinelli}@iit.cnr.it

Abstract. Despite being one of the most common approaches in unsupervised data analysis, a very small literature exists in applying formal methods to address data mining problems. This paper applies an abstract representation of a hierarchical categorical clustering algorithm (CCTree) to solve the problem of privacy-aware data clustering in distributed agents. The proposed methodology is based on rewriting systems, and automatically generates a global structure of the clusters. We prove that the proposed approach improves the time complexity. Moreover a metric is provided to measure the privacy gain after revealing the CCTree result. Furthermore, we discuss under what condition the CCTree clustering in distributed framework produces the comparable result to the centralized one.

Keywords: Distributed Clustering, Algebra, Rewriting, Formal Methods, privacy

1 Introduction

Clustering is a very well-known tool in unsupervised data analysis, which has been the focus of significant research in different domains, spanning from information retrieval, text mining, scientific data exploration, to medical diagnosis [1]. Clustering refers to the process of partitioning a set of data points into groups, such that the elements in the same group are more similar to each other rather than to the ones in other groups. Despite its benefit in a wide range of applications, very few works exist to express and solve the problems of clustering algorithms in terms of formal methods [13].

In the present work, we apply the abstract representation of a categorical clustering algorithm, named CCTree [12], to formalize the process of distributed clustering. Distributed clustering is mainly applied when the data are originally collected at different sites [6]. Generally, for global benefit, the distributed agents are interested in obtaining a global structure of clusters on the whole data, whilst for privacy issues they are unwilling to share their own datasets, except when a desired level of privacy is guaranteed [2]. For example, the Center for Disease Control (master agent) is interested to use the result of clustering on patients' records in different hospitals (agents) to identify

[⋆] This research has been partially supported by the EU Funded Projects H2020 C3IISP, GA #700294, H2020 NeCS, GA #675320, EIT Digital MCloudDaaS and partially by the Natural Sciences and Engineering Research Council of Canada (NSERC) the trends and patterns of diseases. The result on whole dataset brings the benefit for all agents to find the better treatment. However, for privacy concerns, the hospitals are unwilling to disclose the patients' records, unless that a privacy level is satisfied [9].

In this study, we address the problem of privacy-aware distributed CCTree clustering. To this end, first each agent clusters her own dataset with the use of CCTree algorithm [12]. Then, each agent sends the abstract structure of the clusters to a master agent (honest but curious). The abstract schema of clusters is published if it preserves the required privacy of data holder. The master agent aggregates the result of clusters to get a global structure of CCTree such that each agent is able to homogenize her own clusters based on the global structure. The whole process performed by the master agent is formalized with the use of a *rewriting system*. Rewriting system as a well established mathematical structure automatically creates a new desired final result applying the correctly specified rules [3].

The contributions of the present work can be summarized as follows:

– Two rewriting systems are provided in order to automatically verify the compliance of an element of our algebraic structure to a CCTree structure, and moreover to get automatically a global CCTree structure from the abstract schema of CCTrees collected from distributed agents.
– We prove that the proposed rewriting systems terminate and produce the unique result. Furthermore, we state that under which condition the result of our distributed CCTree clustering is comparable with the centralized one.
– A metric is provided for a data holder to measure the privacy gain after revealing the structure of CCTree on her data in order to decide weather to participate in data sharing or not.

The paper is organized as follows. In Section 2, the required background knowledge for the proposed methodology is provided. In Section 3, we apply the abstract CCTree representation to formalize CCTree distributed clustering in terms of rewriting systems. In Section 4, we present a review of the literature. We conclude and point to the future directions of the research in Section 5.

2 Background

In present section, we give some required background information.

2.1 CCTree Construction

CCTree [12] is constructed iteratively through a decision tree-like structure, where the leaves of the tree are the desired clusters. The root of the CCTree contains all the elements to be clustered. Each element is described through a set of *categorical* attributes. Being categorical, each attribute may assume a finite set of discrete values, constituting its domain. At each step, a new level of the tree is generated by splitting the nodes of the previous levels, when they are not homogeneous enough. *Shannon Entropy* is used both to define a homogeneity measure called *node purity*, and to select the attribute used to split a node. In particular non-leaf nodes are divided trough the attribute yielding the maximum value for Shannon entropy. The separation is represented through a branch for each possible outcome of the specific attribute. Each branch or edge extracted from the parent node is labeled with the selected feature which directs data to the child node. A node is considered as a leaf if it respects one of the stop conditions criteria, i.e. 1) the number of elements is fewer than a threshold "μ", or 2) the node purity is better than "ε". Figure 1 depicts a simple CCTree.

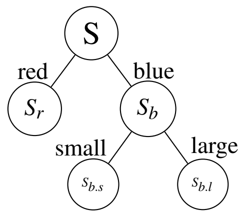

Fig. 1: A small CCTree

2.2 Rewriting Systems

A *rewriting rule* is an ordered pair, written as $x \to y$ of terms x and y. Similar to equations, rules are applied to replace instances of x by corresponding instances of y. Unlike equations, rules are not applied to replace instances of the right-hand side y [3]. A *term* over signature $\mathcal{G}$, constants $\mathcal{K}$, and variables $\mathcal{X}$ is either a variable $x \in \mathcal{X}$, a constant $k \in \mathcal{K}$, or an expression of the form $g(t_1, t_2, \ldots, t_n)$, where $g \in \mathcal{G}$ is a function symbol of n arguments, and t_i are terms [3]. A *derivation* for a rule "$\to$" is a sequence of the form "$t_0 \to t_1 \to \ldots$". An element t is *reducible* (with respect to $\to$), if there is an element u such that "$t \to u$"; otherwise it is considered as *irreducible*. A *rewrite system R* is a set of rewrite rules, $t \to u$, where t and u are term. The term u is "$\to$ *normal*" form of t, if "$t \to^* u$" and u is irreducible via $\to$, where "$\to^*$" means that continuously the transition is applied. A relation $\to$ is *terminating*, if there is no infinite derivations "$t_0 \to t_1 \to \ldots$" which means that it does not reach to a normal term. A relation $\to$ is *confluent*, if there is an element v such that "$s \to^* v$" and "$t \to^* v$" whenever "$u \to^* s$" and "$u \to^* t$" for some elements s, t and u. A relation $\to$ is *convergent*, if it is terminating and confluent. Convergent rewriting system are interesting, because all derivations lead to a unique normal form [3]. A *conditional rule* is an equational implication in which the term in the conclusion is oriented. We use the form "$x_1 = u_1 \wedge \ldots \wedge x_n = u_n \,|\, x \to y$" to show that under the conditions "$x_1 = u_1 \wedge \ldots \wedge x_n = u_n$" we have "$x \to y$".

2.3 Feature-Cluster Algebra

In this section, we present *feature-cluster algebra* proposed in [13] which abstracts CCTree representation in terms of a *term*. We proved in [13] that under the condition of having an order among the set of features, the proposed term *fully abstracts* CCTree structure. Full abstraction guarantees that a CCTree structure and its counterpart CC-Tree term can be applied one instead of the other. In what follows, we briefly present the notations of CCTree abstraction [13] which will be exploited in present study.

(I) Semiring of features: Assume that a *set of disjoint attributes*, denoted as $\mathcal{A}$, is given, where the *carrier set* of each attribute $A_i \in \mathcal{A}$ is denoted by $\mathcal{V}_{A_i}$. We call the *union* of the attributes, denoted as $\mathbb{V} = \bigcup_{A_i \in \mathcal{A}} \mathcal{V}_{A_i}$, the set of *values* or *features*. For example, the set of attributes could be $\mathcal{A} = \{color, size\}$, where $\mathcal{V}_{color} = \{red, blue\}$, and $\mathcal{V}_{size} = \{small, large\}$. Consequently, the set of features equals to $\mathbb{V} = \{red, blue, small, large\}$. Let $\mathbb{F} = \mathcal{P}(\mathcal{P}(\mathbb{V}))$ be the power set of the power set of $\mathbb{V}$. We denote $1 = \{\emptyset\} \in \mathbb{F}$ and $0 = \emptyset \in \mathbb{F}$, and the operations "+" and "$\cdot$" on $\mathbb{F}$ are respectively defined as "$F_i + F_j = F_i \cup F_j$" and "$F_i \cdot F_j = \{X_s \cup Y_t : X_s \in F_i, Y_t \in F_j\}$" for $F_i, F_j \in \mathbb{F}$ [13]. Hence, F belongs to $\mathbb{F}$, if it respects one of the following syntax forms: $F := 0 \,|\, \{\{f\}\} \,|\, F \cdot F \,|\, F + F \,|\, 1$, where $f \in \mathbb{V}$. Then, the quintuple $(\mathbb{F}, +, \cdot, 0, 1)$ constitutes a commutative semiring.

(II) Semiring of elements: Let us consider that the set of the attributes $\mathcal{A} = \{A_1, A_2, \ldots, A_k\}$ is given. We say s belongs to *the set of elements* $\mathbb{S}$, if $s \in \mathcal{V}_{A_1} \times \mathcal{V}_{A_2} \times \ldots \times \mathcal{V}_{A_k} \times \mathbb{N}$, where $\mathbb{N}$ is the set of *natural* numbers. Hence, $s \in \mathbb{S}$ can be written as $s = (x_1, x_2, \cdots, x_k, n)$, where $x_i \in \mathcal{V}_{A_i}$ for $1 \leq i \leq k$, and $n \in \mathbb{N}$ represents the *ID* of an element. For the sake of simplicity, we may use the alternative representation $x_i \in A_i$ instead of $x_i \in \mathcal{V}_{A_i}$. In our problem, $\mathbb{S}$ is the set of all elements that one desires to cluster. As the result of having different sets of elements to be clustered in distributed clustering, we define a semiring of the power set of all elements as follows. Two operations "+" and "$\cdot$" are defined on the elements of $\mathcal{P}(\mathbb{S})$ (the power set of $\mathbb{S}$) as "$S_i + S_j = S_i \cup S_j$" and "$S_i \cdot S_j = S_i \cap S_j$", respectively, for $S_i, S_j \in \mathcal{P}(\mathbb{S})$. Formally, we say S belongs to the *set of elements* $S \in \mathcal{P}(\mathbb{S})$, if it respects one of the syntax forms: $S := \emptyset \,|\, S' \,|\, S + S \,|\, S \cdot S \,|\, \mathbb{S}$, where $S' \subseteq \mathbb{S}$. Then, the quintuple $(\mathcal{S}, +, \cdot, \emptyset, \mathbb{S})$ is a commutative semiring [13].

Definition 1 (Feature-Cluster (Family) Term). *The set of* feature-cluster family terms *on* $\mathbb{V}$ *and* $\mathbb{S}$, *denoted as* $\mathbb{FC}_{\mathbb{V},\mathbb{S}}$ *(or simply* $\mathbb{FC}$ *if it is clear from the context), is the smallest set containing elements satisfying the following conditions:*

$$\begin{array}{llll} if & S \subseteq \mathbb{S} & then & S \in \mathbb{FC} \\ if & F \in \mathbb{F}_1, S \subseteq \mathbb{S} & then & F \diamond S \in \mathbb{FC} \\ if & \tau_1 \in \mathbb{FC}, \tau_2 \in \mathbb{FC} & then & \tau_1 + \tau_2 \in \mathbb{FC} \end{array}$$

In this case, we call S and $F \diamond S$ a feature-cluster term *and the addition of one or more feature-cluster terms is called* feature-cluster family term. *We may simply use* $\mathbb{FC}$*-term to refer to a feature-cluster family term. We define the* block *function, which receives an* $\mathbb{FC}$*-term and returns the set of its blocks as the following:*

$block : \mathbb{FC} \to \mathcal{P}(\mathbb{FC})$

$block(S) = \{S\}$, $block(F \diamond S) = \{F \diamond S\}$, $block(\tau_1 + \tau_2) = block(\tau_1) \cup block(\tau_2)$

In the case that no feature specifies S directly, it is called an atomic *term. The set of all atomic terms is denoted as* $\mathcal{A}$.

Definition 2 (CCTree Term). *A term resulting from a CCTree structure, or equivalently transformable to a CCTree structure, is called a* CCTree term.

Example 1. The CCTree term resulted from Figure 1 is written as follows:

$$\tau = red \diamond S + blue \cdot small \diamond S + blue \cdot large \diamond S$$

where the symbol "$\cdot$" is used to separate the features specifying a cluster, the symbol "$+$" is applied to separate different clusters from each other, and "$\diamond$" is exploited to represent that a cluster is resulted from which main dataset. The latter property is desirable in the process of distributed clustering, where data are clustered in different agents.

Definition 3 (Term). *We call* τ *a* term*, if it has one of the following forms:* $\tau := S \mid F \diamond S \mid \tau + \tau \mid \tau \cdot \tau$*, where* $S := \emptyset \mid S' \mid S + S \mid S \cdot S \mid \mathbb{S}$ *and* $F := 0 \mid \{\{f\}\} \mid F + F \mid F \cdot F \mid 1$. The set of terms *on* $\mathbb{S}$ *and* $\mathbb{F}$ *is denoted as* $\mathbb{C}_{\mathbb{S},\mathbb{F}}$*, or abbreviated as* $\mathbb{C}$*.*

Definition 4 (Feature-Cluster Algebra). *The quintuple* $(\mathbb{C}, \text{"}+\text{"}, \text{"}\cdot\text{"}, 0 \diamond \emptyset, 1 \diamond \mathbb{S})$ *is an idempotent commutative semiring which is called a* feature-cluster algebra.

Definition 5 (Order Rewriting Rule). *Let an ordered set of features* $(\mathbb{V}, <)$ *be given. An* $\mathbb{FC}$*-term is called an* ordered $\mathbb{FC}$-term *on* $(\mathbb{V}, <)$*, if it is the normal form of the following rewriting rule:*

$$f_1 \cdot f_2 \diamond S \rightarrow_O f_2 \cdot f_1 \diamond S \qquad if \qquad f_1 < f_2 \quad \forall f_1, f_2 \in \mathbb{V}$$

Moreover, we define a rewriting rule which orders the features of an $\mathbb{FC}$*-term based on an attribute* $A \in \mathcal{A}$ *as* $f_2 \cdot f_1 \diamond S \xrightarrow{A}_O f_1 \cdot f_2 \diamond S$ *for* $f_1 \in A$*. We represent the normal form of a term* τ *applying above rewriting rule, based on attribute* A*, as* $\tau \Downarrow_A$*.*

To avoid the confusion of different representations of an $\mathbb{FC}$-term, in what follows we present the definitions of *factorized* and *non factorized* terms. In the provided examples, attributes $Color = \{r(ed), b(lue)\}$, $Size = \{s(mall), l(arge)\}$, and $Shape = \{c(ircle), t(riangle)\}$ are used to describe the terms.

Definition 6 (Factorized Term). *We define the* factorization *rewriting rule through an attribute* $A \in \mathcal{A}$*, denoted as* $\xrightarrow{A}$*, from an* $\mathbb{FC}$*-term to its factorized form as the following:*

$$f \cdot \tau_1 + f \cdot \tau_2 \xrightarrow{A} f \cdot (\tau_1 + \tau_2) \qquad for \qquad f \in A$$

we denote the normal form of applying the factorization rewriting rule on term τ *applying factorized rewriting rule, through attribute* A *as* $\tau \downarrow_A$*, and the set of factorized forms of* $\mathbb{FC}$ *is denoted by* $\mathbb{FC}\downarrow$*. A term after factorization is called a* factorized term.

Definition 7 (Non Factorized Term). *We define the* defactorized *rewriting rule on an* $\mathbb{FC}$*-term as "*$f \cdot (\tau_1 + \tau_2) \rightarrow_d f \cdot \tau_1 + f \cdot \tau_2$*". A normal term resulted from defactorized rewriting rule is called a* non factorized term*. A non factorized form of the term* τ *is denoted as* $\tau\uparrow$*. The set of non factorized forms of the terms of* $\mathbb{FC}$ *are denoted by* $\mathbb{FC}\uparrow$*.*

Example 2. For factorization we have: $(r \cdot s \diamond S + r \cdot c \diamond S + b \cdot s \diamond S)\downarrow_{color} = r \cdot (s \diamond S + c \diamond S) + b \cdot s \diamond S$, and for defactorization we obtain: $r \cdot (s \diamond S + c \diamond S) + b \cdot s \diamond S \rightarrow_d r \cdot s \diamond S + r \cdot c \diamond S + b \cdot s \diamond S$.

In the following, we present a set of relations on feature-cluster algebra, introduced in [13], which are applicable in our methodology for distributed clustering.

Definition 8 (Attribute Division). Attribute division *(*$\mathcal{D}_{\mathcal{A}}$*) is a function from* $\mathcal{A} \times \mathbb{FC}$ *to* {True, False}*, which gets an attribute and a non factorized* $\mathbb{FC}$*-term as input; it returns* True *or* False *as follows:*

$$\begin{aligned}
&\mathcal{D}_{\mathcal{A}}(A,S) = \text{False} \\
&\mathcal{D}_{\mathcal{A}}(A,f\diamond S) = \text{True} \qquad if \quad f \in A \\
&\mathcal{D}_{\mathcal{A}}(A,f\diamond S) = \text{False} \qquad if \quad f \notin A \\
&\mathcal{D}_{\mathcal{A}}(A,f\cdot F\diamond S) = \mathcal{D}_{\mathcal{A}}(A,f\diamond S) \vee \mathcal{D}_{\mathcal{A}}(A,F\diamond S) \\
&\mathcal{D}_{\mathcal{A}}(A,\tau_1+\tau_2) = \mathcal{D}_{\mathcal{A}}(A,\tau_1) \wedge \mathcal{D}_{\mathcal{A}}(A,\tau_2)
\end{aligned}$$

The concept of attribute division is used to order the attributes presented in a term.

Definition 9 (Initial). *We define the* initial *(δ) function from $\mathcal{P}(\mathbb{FC}\uparrow)$ to $\mathcal{P}(\mathbb{F})$, which gets a set of ordered non factorized terms on $(\mathbb{V},<)$ and returns a set of the first features of each term as follows:*

$$\delta(\emptyset) = \{0\}\ ,\ \delta(\{S\}) = \{1\}\ ,\ \delta(\{f\cdot F\diamond S\}) = \{f\}\ ,\ \delta(\{\tau_1+\tau_2\}) = \delta(\{\tau_1\}) \cup \delta(\{\tau_2\})$$

In the case that the input set contains just one term, we remove the brackets, i.e. $\delta(\{\tau\}) = \delta(\tau)$ when $|\{\tau\}| = 1$. Moreover, when the output set also contains just one element, for the sake of simplicity we remove the brackets, i.e. $\delta(X) = \{f\} = f$ for $X \in \mathcal{P}(\mathbb{FC}\uparrow)$. The initial function will be used in the process of evaluating that if a term represents a CCTree term, considering that in a CCTree the sibling features (first features in an ordered term) belong to the same attribute.

Definition 10 (Derivative). *We define the* derivative, *denoted by ∂, as a function which gets an ordered non factorized $\mathbb{FC}$-term on $(\mathbb{V},<)$, i.e. $\partial : \mathbb{FC}\uparrow \to \mathcal{P}(\mathbb{FC})$; it returns the term (set of terms) by cutting off the first features as follows:*

$$\partial(S) = \emptyset\ ,\ \partial(f\diamond S) = \{S\}\ ,\ \partial(f\cdot F\diamond S) = \{F\diamond S\}\ ,\ \partial(\tau_1+\tau_2) = \partial(\tau_1)\cup\partial(\tau_2)$$

The derivative function is defied to be used in the process of evaluating if a term represents CCTree term. More precisely, if the first level features in an ordered term (siblings in tree) belong to the same attributes, with derivative function we remove first features to evaluate if the sub-terms (sub-trees) also represent CCTree term.
Note that the functions initial *(δ) and* derivative *(∂) are overloaded to the input, depending on the input that if it is a tree or a term.*

Definition 11 (Order of Attributes). *We say attribute B is smaller or equal to attribute A on the non factorized term $\tau \in \mathbb{FC}\uparrow$, denoted as $B \preceq_\tau A$, if the number of blocks of τ that B divides, is less than (equal to) the number of blocks that A divides. Formally, $B \preceq_\tau A$ implies that:*

$$|\{\tau_i \in block(\tau) \mid \mathcal{D}_{\mathcal{A}}(B,\tau_i) = True\}| \leq |\{\tau_i \in block(\tau) \mid \mathcal{D}_{\mathcal{A}}(A,\tau_i) = True\}|$$

Given a set of attributes $\mathcal{A}$ and a term τ, the set $(\mathcal{A},\preceq_\tau)$ is a lattice. We denote the upper bound *of this set as $\sqcap_{\mathcal{A},\tau}$. This means that we have $\forall A \in \mathcal{A} \Rightarrow A \preceq_\tau \sqcap_{\mathcal{A},\tau}$.*

Example 3. In the following we show how the order of attributes of a term is identified. Suppose the term $\tau = r\cdot s\diamond S + r\cdot c\diamond S + b\cdot s\diamond S$ is given. We have: $block(\tau) = \{r\cdot s\diamond S, r\cdot c\diamond S, b\cdot s\diamond S\}$.Consequently, we obtain:

$$\begin{aligned}
&|\{\tau_i \in block(\tau) | \mathcal{D}_{\mathcal{A}}(shape,\tau_i) = True\}| = 1 \\
\leq\ &|\{\tau_i \in block(\tau) | \mathcal{D}_{\mathcal{A}}(size\,,\ \tau_i)\ = True\}| = 2 \\
\leq\ &|\{\tau_i \in block(\tau) \mid \mathcal{D}_{\mathcal{A}}(color,\tau_i) = True\}| = 3
\end{aligned}$$

which means that we have "$shape \preceq_\tau size \preceq_\tau color$". Namely, the attribute "$color$" is the one which appears in all the blocks of term τ. We notify that in CCTree the first attribute from the root is the one which appears in all the blocks of the equivalent CCTree term. For example, in the CCTree term of Example 1, the attribute color exists in all blocks.

Features Ordering Recalling that not having the predefined order among features creates a problem in full abstraction of terms [13]. To this end, here we propose a way to order the set of features which is appropriate to our problem. Given an $\mathbb{FC}$-term τ, we find the order of attributes according to Definition 11, whilst if for two arbitrary attributes A and A', we have $A = A'$, without loss of generality, we choose a strict order among them, say $A \prec A'$. Then in each attribute we arbitrarily (and fix) order the features.

Definition 12 (Ordered Unification). Ordered unification *($\mathcal{F}$) is a partial function from $\mathcal{P}(\mathcal{A}) \times \mathbb{FC}\uparrow$ to $\mathbb{FC}\downarrow$, which gets a set of attributes and a non factorized term; it returns the normal form of rewriting rule $\xrightarrow{A}_O$ (Definition 5), iteratively, based on the order of attributes on received term as follows:*

$$\mathcal{F}(\emptyset,\tau\uparrow) = \tau \quad , \ \mathcal{F}(\{A\},\tau\uparrow) = \tau\Downarrow_A \ , \ \mathcal{F}(\mathcal{A},\tau) = \mathcal{F}(\sqcap_{\mathcal{A},\tau}, \mathcal{F}(\mathcal{A} - \{\sqcap_{\mathcal{A},\tau}\},\tau\uparrow))$$

The normal form of ordered unification is called a unified term. *By $\mathcal{F}^*(\tau)$ we mean that $\mathcal{F}$ is performed iteratively on the set of ordered attributes on τ to get the unified term.* Ordered unification function *by automatically ordering the features of a term, as explained before, directs the shape of a term to be easily verified if it is a CCTree term.*

Example 4. To find the unified form of $\tau_1 = r\cdot s \diamond S + r \cdot c \diamond S + b \cdot s \diamond S$, we have:

$$\begin{aligned} \mathcal{F}^*(\tau_1) = \mathcal{F}(\{shape, color, size\}, \tau_1\uparrow) &= \mathcal{F}(color, \mathcal{F}(size, \mathcal{F}(shape, \tau_1))) \\ &= r\cdot s \diamond S + r \cdot c \diamond + b \cdot s \diamond S \end{aligned}$$

Definition 13 (Well-formed Term). Well formed function, *denoted as W, is a binary function from $\mathbb{FC}\uparrow$ to* $\{$True, False$\}$, *which gets a unified non factorized $\mathbb{FC}$-term $\tau\uparrow$; it returns* True *if $\delta(\tau\uparrow)$ is equal to one of the attributes belonging to $\mathcal{A}$; it returns* False *otherwise. Formally:*

$$W(\tau\uparrow) = \begin{cases} \text{True} \ \ if & \exists A_i \in \mathcal{A} \quad s.t. \quad \delta(\tau\uparrow) = A_i \\ \text{False} & \qquad\qquad\qquad otherwise \end{cases}$$

A unified term τ is called a well formed term, *if $W(\tau) = True$.*

An atomic term is considered as a *well formed term*. Basically, well-form function verifies if in a unified non factorized term the first level features belong to the same attribute or not, as expected for CCTree structure.

Theorem 1. *A unified term represents a CCTree term, or it is transformable to a CCTree structure, if and only if, 1) it can be written in the form $\mathcal{F}^*(\tau) = \sum_i f_i \cdot \tau_i$, such that 1) " $W(\mathcal{F}^*(\tau)) = True$", i.e. the unified form of the received term is a well formed term; and 2) the unified form of each τ_i is a well formed term as well ($W(\tau_i) = True$) and 3) each τ_i respects above requirements [13].*

3 Distributed Private CCTree Clustering

Data clustering has become increasingly exploited in a wide range of applications, spanning from molecular biology to marketing [1]. In many areas, the data are collected in different sites, for instance different hospitals. Due to the privacy issues, each data holder may prevent to publish her own dataset, unless some privacy level is guaranteed [9]. To this end, each agent computes the amount of privacy leakage when the clustering algorithm result is published comparing to the publishing of the original dataset. If this difference is higher than her desired threshold, then she will share her CCTree; Otherwise she refuses to participate in collaborative CCTree construction.

Afterwards, the result of each CCTree in each participated agent is transformed to its equivalent CCTree term. The resulted CCTree terms are reported to the master agent (honest but curious) for composition. The CCTree terms are composed automatically based on our proposed composition rewriting rules (Table 2), which creates a final CCTree term that all terms can be homogenized to it. Therefore, the composition result is reported to each agent to homogenize CCTree terms, and consequently the structure of all CCTrees. Figure 2 depicts a high level representation of such an architecture.

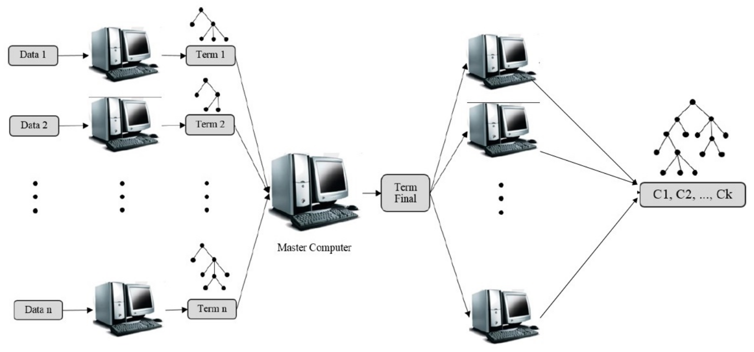

Fig. 2: Distributed Clustering Workflow.

In what follows, we first prove that if all agents fix the stop condition (the minimum number of elements in leaves) of CCTree equal to μ, then publishing the result as a CCTree term satisfies k-anonymous dataset (k is dependent to μ) [4]. Then, we propose *CCTree rewriting system* which automatically verifies the compliance of a term to a CCTree structure. Afterward, we come up with the *composition rewriting system*, which is exploited to find a global CCTree term from the addition of several CCTree terms (received from distributed agents). At the end of the section, we prove that the proposed methodology results in 1) a unique CCTree term from the addition of several CCTree terms, 2) the process of finding the global CCTree terminates, 3) the time complexity improves in distributed system, comparing to centralized one, and finally 4) under some condition the CCTree, resulted from the distributed architecture, is comparable to the centralized schema.

3.1 Privacy-aware Data Sharing

In [5], a statistical framework is proposed to measure the amount of privacy which is violated through publishing the result of a classifier. We extend the same notations

for publishing the result of our clustering algorithm. This privacy violation is formally measured as what follows. Given a distribution (P,U,S), where P is public data that everyone including the adversary can access, S refers to sensitive data we are trying to protect, and U is data not known by the adversary. The resulted structure of CCTree say $\mathcal{C}$, i.e. CCTree term, is available to adversary which can be used to predict U given P. Assume that t samples $\{(p_1,s_1),\dots,(p_t,s_t)\}$ are already available to adversary. The goal is to test whether revealing the resulted $\mathcal{C}$ increases the ability of adversary to predict S values for unseen samples. It is expected that $\mathcal{C}$ would not be much more accurate than random guess, and the adversary is not able to improve her own estimation about S applying $\mathcal{C}$. Formally, this concept is measured through Bayes error, and means that for all classifiers using P the Bayes error should bed the same as the Bayes error of all classifiers using $(P,\mathcal{C}(P))$. Formally, given $\mathcal{C}$ and t samples from P and S, if $\rho(t) = \rho\{t;P,S\}$ and $\rho(t;\mathcal{C}) = \rho\{t;P,\mathcal{C}(P),S\}$ be the Bayes errors for classifiers using P only and using $P,\mathcal{C}(P)$ respectively, and considering $\bar{\rho} = \lim_{t\to\infty}$, and $\bar{\rho}(\mathcal{C}) = \lim_{t\to\infty}\rho(t;\mathcal{C})$, we have the upcoming definition from [5].
For $0 \leq p \leq 1$, the result of CCTree $\mathcal{C}$ is (t,p) privacy violating if $\rho(t;\mathcal{C}) \leq \rho(t) - p$, and the clustering $\mathcal{C}$ is (∞,p)- privacy violating if $\bar{\rho}(\mathcal{C}) \leq \bar{\rho} - \rho$.
Under this definition, if an agent finds that distributing her CCTree violates her privacy requirement, she can refuse to participate in sharing the resulting CCTree on her dataset.

3.2 CCTree Rewriting System

In order to be able to define some rules on a term to verify if it satisfies CCTree structure, it is required to define the concept of *Component* of a term. Roughly speaking, a component of a CCTree term refers to sub-tree in CCTree structure. Hence, when we want to check the compliance of a term to a CCTree structure, we need to verify it iteratively through sub-terms, as the iterative structure of CCTree.

Definition 14 (Component). *Given two ordered non factorized $\mathbb{FC}$-terms τ_1 and τ_2 on $(\mathbb{V},<)$, we define the* component *relation, denoted by $\sim$, as the first level comparison of the terms as $\tau_1 \sim \tau_2 \Leftrightarrow \delta(\tau_1) = \delta(\tau_2)$.*

Let the ordered term $\tau \in \mathbb{FC}\uparrow$ on $(\mathbb{V},<)$ be given. The equivalence class of $\tau' \in block(\tau)$ is called a component *of τ, and it is formally defined as:*

$$[\tau']_\tau = \{\tau_i \in block(\tau) \mid \tau' \sim \tau_i\}$$

The set of all components of the term τ through the equivalence relation $\sim$, is denoted by "$block(\tau)/\sim$" or simply "$\tau/\sim$", i.e. we have: $\tau/\sim = \{[\tau_i]_\tau \mid \tau_i \in block(\tau)\}$.
We order the components of term τ according to the order of features in $\mathbb{V}$ as following:

$$[\tau']_\tau < [\tau'']_\tau \quad \Leftrightarrow \quad (\ \forall f' \in \delta([\tau'])\ ,\ \forall f'' \in \delta([\tau'']) \Rightarrow f' < f'')$$

since the features are ordered strictly, the components are also ordered strictly.

Example 5. Consider the ordered form of the term τ_1 in Example 4, i.e. $\tau_1 = r \cdot s \diamond S + r \cdot c \diamond S + b \cdot s \diamond S$. The components of $\tau_1 \uparrow$ are $\{r \cdot s \diamond S, r \cdot c \diamond S\}$ (all blocks begin with r) and $\{b \cdot s \diamond S\}$ (all blocks begin with b). Moreover if in the first feature ordering we have $r > b$, then $\{r \cdot s \diamond S, r \cdot c \diamond S\} > \{b \cdot s \diamond S\}$.

To verify automatically if a term is a CCTree term, a set of conditional rewriting rules are provided in Table 1. The term $\mathbb{0}$ in this table, refers to a *null* term. In this regard, the CCTree rewriting system is applied on a received term; the term is a CCTree term if the only irreducible term is $\emptyset$. In this rewriting system, $\big[[f(\tau)]\big]$ means that the semantics of $f(\tau)$ is replaced, whilst the result is considered as one unique term. Furthermore, $\tau_1 : \tau_2$ contains two terms τ_1 and τ_2, whilst each one is considered as a new term. Moreover, $[\tau]_i$ refers to the i'th component of "$\tau/\sim$" [13].

$$
\begin{array}{l}
(1)\ \ (\tau \in \mathcal{A}) \mid \tau \rightarrow \emptyset \\
(2)\ \ (\tau \neq \mathcal{F}^*(\tau)) \mid \tau \rightarrow \big[[\mathcal{F}^*(\tau)]\big] \\
(3)\ \ (\tau = \mathcal{F}^*(\tau)) \wedge (W(\tau))) \wedge (\tau \notin \mathcal{A}) \mid \tau \rightarrow \big[[\Sigma_{\tau_k \in [\tau]_1} \partial(\tau_k)]\big] \ :\ \ldots\ :\ \big[[\Sigma_{\tau_k \in [\tau]_{|\tau/\sim|}} \partial(\tau_k)]\big]
\end{array}
$$

Table 1: CCTree Rewriting System

The first rule of Table 1 specifies that if a term is an atomic term, it is directed to $\emptyset$. The second rule expresses that if a term is not in unified form, it is required to be transformed to its unified representation. The third rule specifies that if a non atomic unified term is well formed, it is divided to the derivative of its components. The last rule is used to verify whether the CCTree conditions satisfy for the following components or not, resulting from the iterative structure of CCTree. These rules are following the structure of Theorem 1 in identifying if a term is in compliance with a CCTree structure.

Example 6. Suppose that the term $\tau_1 = a_1 \diamond S + b_1 \diamond S$, with the set of attributes $A = \{a_1, a_2\}, B = \{b_1, b_2\}$, are given. We apply the CCTree rewriting rules to automatically verify if τ_1 is a CCTree term. The term τ_1 is not atomic. Moreover, we have $\tau_1 = \mathcal{F}^*(\tau_1)$ and "$W(\tau_1) = \mathit{False}$" since the first features of the components of τ_1 are not equal. There is no CCTree rewriting rule which can be applied, whilst this term is not $\emptyset$. This means that the received term τ_1 is not a CCTree term.
As another instance, we show that the term $\tau_2 = a_1 \diamond S + a_2 \diamond S$ with the set of attributes $A = \{a_1, a_2\}, B = \{b_1, b_2\}$, is a CCTree term.

$$(\tau_2 = \mathcal{F}^*(\tau_2)) \wedge (W(\tau_2)) \mid a_1 \diamond S + a_2 \diamond S \xrightarrow{(3)} S : S \xrightarrow{(1)} \emptyset : \emptyset$$

The condition of this conditional rewriting rule verifies that if the term is in unified form, and if in unified form all first features belong to the same attribute or not. If so, then the terms is broken to its components, to verify the same condition for following sub-terms. Since by removing the first features, atomic terms (terms without feature specifying them) are ontained each one is directed to a *null* term. There is no irreducible term except $\emptyset$, hence, τ_2 is a CCTree term.

3.3 Composition Rewriting System

To address the composition process, a set of *composition rewriting rule*s (Table 2) are proposed to obtain automatically a CCTree term when a term is not a CCTree term. The *split* relation (4'th rule of Table 2) is added to the rules of Table 1 to get CCTree term from non CCTree term.

Definition 15 (Split). *Suppose that a unified term $\tau \in \mathbb{FC}\uparrow$ on $(\mathbb{V},<)$ and the set of attributes $\mathcal{A}$, is given. Considering $\sqcap_{\mathcal{A},\tau}$ as the upper bound attribute of τ, we define the* split *relation as what follows:*

$$split(\tau) = \begin{cases} \tau & if \quad W(\tau) = True \\ \sum_{\tau_i \in block(\tau)} \zeta(\tau_i) & if \quad W(\tau) = False \end{cases}$$

where:

$$\zeta(\tau_i) = \begin{cases} \tau_i & if \quad \mathcal{D}_{\mathcal{A}}(A,\tau_i) = True \\ (\sum_{a_i \in \sqcap_{\mathcal{A},\tau}} a_i) \cdot \tau_i & if \quad \mathcal{D}_{\mathcal{A}}(A,\tau_i) = False \end{cases}$$

This means that all the blocks of τ which do not contain any feature of $\sqcap_{\mathcal{A},\tau}$ are multiplied to the addition of the features of $\sqcap_{\mathcal{A},\tau}$. In the following example we show how $split$ relation is applied.

Example 7. Suppose that the term $\tau_2 = r \cdot s \diamond S + c \diamond S + b \diamond S$ is given. We have $W(r \cdot s \diamond S + r \cdot c \diamond S + b \diamond S) = False$, hence, τ_2 is not a well formed term. Considering that $\sqcap_{\mathcal{A},\tau_2} = color$, and $\mathcal{D}_{\mathcal{A}}(colro, r \cdot s \diamond S) = True$, $\mathcal{D}_{\mathcal{A}}(colro, r \cdot c \diamond S) = True$, $\mathcal{D}_{\mathcal{A}}(colro, b \diamond S) = False$, we have: $split(r \cdot s \diamond S + c \diamond S + b \diamond S) = r \cdot s \diamond S + (r+b) \cdot c \diamond S + b \diamond S = r \cdot s \diamond S + r \cdot c \diamond S + b \cdot c \diamond S + b \diamond S$.

Actually, when a term is not a CCTree term, it is possible to infer it from its unified form when the first features of its components do not belong to the same attribute. Therefore, the split rule is proposed to generate a well formed term from a non CCTree term. In what follows, we add the split rule to the previous rewriting system, which is used when a term is not a CCTree term to obtain a CCTree term.

The composition rewriting rules to get a CCTree term from a non CCTree term is presented in Table 2. In the proposed rewriting system, $[[f(\tau)]]$ means that the semnatic of $f(\tau)$ is replaced, whilst the result is considered as one unique term, not several terms. Furthermore, $\tau_1 : \tau_2$ contains two terms τ_1 and τ_2, whilst each one is considered as a new term; and $[\tau]_i$ refers to the i'th component of $\tau/\sim$. Comparing to Table 1, just the

(1) $(\tau \in \mathcal{A}) \mid \tau \to \emptyset$
(2) $(\tau \neq \mathcal{F}^*(\tau)) \mid \tau \to [[\mathcal{F}^*(\tau)]]$
(3) $(\tau = \mathcal{F}^*(\tau)) \wedge (W(\tau)) \wedge (\tau \notin \mathcal{A}) \mid \tau \to [[\Sigma_{\tau_k \in [\tau]_1} \partial(\tau_k)]] \;:\; \ldots \;:\; [[\Sigma_{\tau_k \in [\tau]_{|\tau/\sim|}} \partial(\tau_k)]]$
(4) $(\tau = \mathcal{F}^*(\tau)) \wedge (\sim W(\tau)) \mid \tau \to [[split(\tau)]]$

Table 2: Composition Rewriting System

split rule is added. This rule guarantees that if a term is not a CCTree term, how by splitting the term based on the upper bound attribute we may get a CCTree term.

To this end, first of all, the set of attributes $\mathcal{A}$ describing the received term τ is provided. Note that in categorical clustering algorithm, the set of attributes are known beforehand. The set of attributes and non CCTree term are given to the composition rewriting system. When the conditions of the rule $(\tau = \mathcal{F}^*(\tau)) \wedge (W(\tau)) \mid \tau \rightarrow \big[[\Sigma_{\tau_k \in [\tau]_1} \partial(\tau_k)]\big]$: $\dots$: $\big[[\Sigma_{\tau_k \in [\tau]_{|\tau/\sim|}} \partial(\tau_k)]\big]$ respects for a term τ, we save τ. Then all $\big[[\Sigma_{\tau_k \in [\tau]_i}]\big]$ of τ are replaced by their own successive terms respecting this rule. This process is repeated iteratively till reaching to atomic terms in all derivations. The result of this term is the desired CCTree term.

Example 8. Suppose that the addition of two CCTree terms is given as $\tau = a_1 \diamond S + a_2 \diamond S + b_1 \diamond S' + b_2 \diamond S'$, with the set of attributes $A = \{a_1, a_2\}, B = \{b_1, b_2\}$. It is easy to verify that τ is not a CCTree term from the rules of Table 1. We are interested to find a CCTree term from the received non CCTree term τ, with the use of composition rewriting system. To this end we have:

(i) $(\tau = \mathcal{F}^*(\tau)) \wedge (\sim W(\tau)) \mid \tau \xrightarrow{(4)} \big[[split(\tau)]\big]$

(ii) $\big[[split(\tau)]\big] = \tau' = a_1 \diamond S + a_2 \diamond S + (a_1 + a_2) \cdot b_1 \diamond S' + (a_1 + a_2) \cdot b_2 \diamond S'$

(iii) $(\tau' \neq \mathcal{F}^*(\tau')) \mid \tau' \xrightarrow{(2)} \big[[\mathcal{F}^*(\tau')]\big] = (a_1 \cdot (S + b_1 \diamond S') + a_2 \cdot (S + b_1 \diamond S')) = \tau''$

(iv) $(\tau'' = \mathcal{F}^*(\tau'')) \wedge (W(\tau'')) \mid \tau'' \xrightarrow{*(3)*} S + b_1 \diamond S'(I)\ :\ S + b_1 \diamond S'(II)$

$$(I)\ S + b_1 \diamond S' \xrightarrow{(4)} (b_1 + b_2) \cdot S + b_1 \diamond S' \xrightarrow{(2)} b_1 \cdot (S + S') + b_2 \diamond S$$
$$\xrightarrow{*(3)*} S + S' : S \xrightarrow{(1)} \emptyset : \emptyset$$
$$(II)\ S + b_1 \diamond S' \xrightarrow{(4)} (b_1 + b_2) \cdot S + b_1 \diamond S' \xrightarrow{(2)} b_1 \cdot (S + S') + b_2 \cdot S$$
$$\xrightarrow{*(3)*} S + S' : S \xrightarrow{(1)} \emptyset : \emptyset$$

To find the resulted CCTree term, we consider the terms respecting the rule (3), shown with $*(3)*$. Hence, we have them as: $(*)\, a_1 \cdot (S + b_1 \diamond S') + a_2 \cdot (S + b_1 \diamond S')$, $(**)\, b_1 \cdot (S + S') + b_2 \diamond S$, $(***)\, b_1 \cdot (S + S') + b_2 \cdot S$.
Then, since $(**)$ results from this term $S + b_1 \diamond S'$ inside $(*)$, and $(***)$ from term $S + b_1 \diamond S'$ inside $(*)$, we replace them to their previous form:
$$a_1 \cdot (b_1 \cdot (S + S') + b_2 \diamond S) + a_2 \cdot (b_1 \cdot (S + S') + b_2 \cdot S)$$
Since there is no more term respecting rule (3), the above term is the desired CCTree term. It can automatically be verified that the resulted term is a CCTree term (Table 1).

After that the final CCTree term, resulting from the composition of two (or more) CCTree terms, is returned to the distributed devices, the CCTree term of each agent has to be extended to the final CCTree term. The extension of each CCTree term to a final CCTree term will homogenize the structure of all CCTrees. To this end, it is enough to add a CCTree term with the final CCTree term. Then, all *split* rules applied on CCTree term in the process of its composition with final CCTree term, shows the required split in the associated CCTree structure, following the procedure of transforming a term to tree provided in [13].

3.4 Confluent Rewriting Systems

In this section, we first present what the termination and confluence of a rewriting system mean. Furthermore, through several theorems, we prove our proposed rewriting systems are terminating and confluent. Termination and confluence are the interesting properties of a rewriting system, which guarantee that firstly, applying the rewriting rules of the proposed system, there is no infinite loop of rules, and furthermore, we always get a unique result.

Termination and Confluence of a Rewriting System A rewriting system is terminating, if there is no infinite derivation "$t_1 \to t_2 \to t_3 \to \ldots$" in R. This implies that every derivation eventually ends to a normal form [3]. Lankford theorem claims that a rewriting system R is terminating, if for some *reduction ordering* "$>$", we have "$x > y$" for all rules "$x \to y \in R$". An order is a reduction ordering, if it is *monotonic* and *fully invariant* [3]. A relation is monotonic if it preserves the order through adding or reduction a term in both sides, and it is fully invariant, if it preserves the order when a term is substituted in both sides of the relation [3]. An element t in the rewriting system R is locally confluent if for all $x, y \in R$ such that "$t \to x$" and "$t \to y$", there exists $u \in R$ such that "$x \to^* u$" and "$y \to^* u$". If every $t \in R$ is locally confluent, then $\to$ is called locally confluent. Newman's lemma expresses that a terminating rewriting system is confluent if and only if it is locally confluent [3].

Theorem 2. *The CCTree rewriting system is terminating.*

Proof. To prove this theorem we first define a *reduction order* on the rules of CCTree rewriting system. To this end, we define the *size* function which gets an $\mathbb{FC}$-term and returns the number of features appeared in the term as follows: $size : \mathbb{FC} \to \mathbb{N}$, such that $size(S) = 1$, $size(f \diamond S) = 1$, $size(F \cdot \tau) = |F| + size(\tau)$, $size(\tau_1 + \tau_2) = size(\tau_1) + size(\tau_2)$ and we consider $size(\emptyset) = 0$, and $size(\tau_1 : \tau_2) = size(\tau_1) + size(\tau_2)$.
We say $\mathbb{FC}$-term τ_1 is smaller than $\mathbb{FC}$-term τ_2, denoted by $\tau_1 \leq \tau_2$, if the number of features in τ_1 is less than the number of features in τ_2, or equally $size(\tau_1) \leq size(\tau_2)$. This partial ordering is well-founded, since there is no infinite descending chain (number of features are limited). It is monotonic, because the property of number of features in two terms is preserved when a term is added or reduced in both sides. Furthermore, the substitution in left and right sides, preserves the order of number of features, i.e. it is fully invariant. Therefore, the proposed ordering is a reduction ordering.

Considering that $\emptyset$ is a null term containing no feature, in the first rule we have $atomic\ term > \emptyset$. In the second one, the conditional rule is just applied when the term is not equal to its unified form; whilst the ordered unification function, if applied, does not change the number of features, i.e. $\tau \geq \mathcal{F}^*(\tau)$ for $\tau \neq \mathcal{F}^*(\tau)$, since $size(\tau) = size(\mathcal{F}^*(\tau))$. Worth noticing that this rule is a one step rule, such that when the term is unified, the other rules are exploited. In the third rule, the first features of all components of the left term are removed, i.e. the size (number of features) of the left-hand term is greater than the size (number of features) in the right-hand one. Hence, the proposed reduction ordering $\leq$ on CCTree rewriting system, so the system is terminating. □

Theorem 3. *The CCTree rewriting system is confluent.*

Proof. In CCTree rewriting system, all rules are conditional and there is no term for which two (or more) conditions are satisfied at the same time. This means that the possibility of having $\tau \to \tau_1$ and $\tau \to \tau_2$ where $\tau_1 \neq \tau_2$, does not happen. Hence, the rewriting system is locally confluent. According to Newman's lemma, the CCTree rewriting system being terminating (Theorem2) and locally confluent, it is confluent. □

Theorem 4. *The composition rewriting system is confluent.*

Proof. The only rule added to composition rewriting system comparing to CCTree rewriting system, is the rule *split*. We show that *split* rule is not contradicting the termination and confluence of rewriting system. First of all, the *split* rule is one step rule, i.e. the result of *split* rule, after one step application, is considered as the premise of other rules (which decreases the term). On the other hand, on each term, the *split* rule is applied at most equal to the number of attributes (finite). Hence, since the *split* by itself is one step rule, and for each term it is called finite times, the composition rewriting system is terminating.

On the other hand, there is no term respecting at the same time two (or more) conditions of composition rewriting system, i.e. there is no term τ for which $\tau \to \tau_1$ and $\tau \to \tau_2$, where $\tau_1 \neq \tau_2$. This means that composition rewriting system is locally confluent. Therefore, the composition rewriting system is terminating and locally confluent, and hence, from Newman's lemma, it is confluent. $\square$

3.5 Complexity and Result Comparison between Centralized and Distributed CCTree Clustering

In what follows, we discuss on time complexity improvement for CCTree clustering in distributed devices, compared to centralized one. Furthermore, we state under what condition the CCTree resulted from distributed schema equals to centralized one.

Theorem 5. *Let us consider N to be the total number of elements desired to be clustered, k be the number of attributes, v_{max} be the maximum number of values in an attribute, and K be the maximum number of non leaf nodes. The time complexity of constructing CCTrees in n distributed devices equals to $\frac{1}{n} \cdot O(K \times (N \times m + N \times v_{max}))$.*

Proof. In [11], the time complexity of constructing a CCTree has been calculated. Recalling again, consider N as the number of elements in whole dataset, N_i be the number of elements in node i, m be the total number of features, v_l the number of features of attribute A_l, k the number of attributes, and $v_{max} = max\{v_l\}$. For constructing a CCTree, if $K = m+1$ be the maximum number of non leaf nodes, which arise in a complete tree, then the maximum time required for constructing a CCTree with N elements equals to $O(K \times (N \times m + N \times v_{max}))$. Now if we equally divide the dataset containing N points to n devices, it takes $O(K \times ((N/n) \times m + (N/n) \times v_{max})) = \frac{1}{n} \cdot O(K \times (N \times m + N \times v_{max}))$ to create n CCTrees, i.e the whole required time will be divided to the number of devices. The other part of algebraic calculations requires constant time. $\square$

Definition 16 (Kullback-Leibler Divergence). *Let P and Q be two probability distributions on $A_1 \times A_2 \times \ldots \times A_k$, where $A_i \in \mathcal{A}$, $\forall \leq i \leq k$ (the set of attributes in CCTree clustering). Then,* Kullback-Leibler divergence *[7] from Q to P, denoted by $D_{KL}(P||Q)$, is a measure gained comparing the probability distribution Q with the probability distribution P, as follows:*

$$D_{KL}(P||Q) = \sum_{X_i \in A_1 \times A_2 \times \ldots \times A_k} p(X_i) \log \frac{p(X_i)}{q(X_i)}$$

whilst whenever $\log \frac{p(X_i)}{q(X_i)} \to \infty$, then we set $\log \frac{p(X_i)}{q(X_i)}$ to 1.

From the above definition, if for two datasets D_1 and D_2 with the probability distributions P_1 and P_2, respectively, we have the result of $D_{KL}(P_1||P_2)$ is close to zero, then it means that they have almost the same data distribution. Consequently, they will produce *similar* CCTrees.

Theorem 6. *Let D_1 and D_2 be two datasets with the probability distributions P_1 and P_2 (as Definition 16), respectively. If $D_{KL}(P_1||P_2) = 0$, then $CCTree(D_1) = CCTree(D_2)$.*

Proof. The proof is resulted from the fact that the structure of CCTree is dependent to the attribute selected for dividing the data in each node. If the distribution of elements in two dataset are almost equal, it means that the same attribute in both causes the highest Shannon entropy, and hence, the one selected for data division through branches labeled the features of selected attribute. □

Theorem 6 states that if for two datasets the probability distributions are equal, it results in having equal CCTree structure for both agents. In other words, when *Kullback-Leibler divergence* of two probability distribution tends to zero, the result of our distributed framework produces the more similar result to the centralized system. Extending the result to more than two datasets, it can be expressed as follows. Let N datasets $D_1, D_2, \ldots, D_n$ with the probability distributions $P_1, P_2, \ldots, P_n$, respectively, be given. If $D_{KL}(P_i||P_j) = 0$ for all $1 \leq i, j \leq n$, then the CCTree resulted from our distributed framework equals to CCTree of centralized system. The more the probability distribution is divergent (higher Kullback-Leibler divergence result), the more it is possible that the final CCTree in distributed framework be different from the centralized one. In future work, we plan to verify this topic in more detail through real use case experiments.

4 Related Work

The problem of knowledge extraction among multiple parties involved in a data mining task has been presented in [14]. The studied methodology aims at performing data mining without data disclosure between the parties. This methodology relies on homomorphic encryption and digital envelope techniques. These techniques suffer from the drawback of being applicable to only a small set of data analysis functions. Also they impose a considerable overhead. In [8], the basic paradigms and the notions of secure multiparty computation and its relation to privacy preserving data mining has been surveyed. Still it only works on data mining algorithms that apply the proposed computations. Clifton et. al. [2] propose a toolkit for different applications required in privacy preserving distributed data mining. Data transformation methods have been proposed in [10] for privacy preserving clustering. In [9], a general framework is proposed to formalize the architecture of privacy preserving data mining in collaborative system. However, to the best of our knowledge, the present study is amongst the very first efforts in applying algebraic structure for addressing data mining issues.

5 Conclusion

In the present work, an abstract representation of a categorical clustering algorithm, named CCTree, is used to address the problem of privacy-aware distributed clustering.

Generally for global benefit, the distributed agents are interested to share their information to get the global structure of their own data. However, for privacy concerns, the agents in distributed system are unwilling to disclose their datasets. To address the aforementioned challenges in distributed CCTree clustering, we proposed a rewriting system which automatically returns a CCTree term, in a way that all CCTrees in distributed agents can be homogenized. The termination and confluence of the proposed rewriting system have been proven, which guarantees first of all we have no infinite loop in applying the proposed rewriting systems, and moreover, the resulted final term is unique.
In future directions, we plan to apply the proposed methodology in realistic case studies to evaluate its efficiency in distributed clustering. Moreover, we plan to generalize the proposed approach for a wide range of distributed categorical clustering where the features play an important role in identifying the clusters, thence, applicable in abstraction.

References

1. Berkhin, P.: A survey of clustering data mining techniques. In: Grouping Multidimensional Data, pp. 25–71. Springer (2006)
2. Clifton, C., Kantarcioglu, M., Vaidya, J., Lin, X., Zhu, M.Y.: Tools for privacy preserving distributed data mining. SIGKDD Explor. Newsl. 4(2), 28–34 (2002)
3. Dershowitz, N., Jouannaud, J.: Handbook of theoretical computer science (vol. b). chap. Rewrite Systems, pp. 243–320. MIT Press, Cambridge, MA, USA (1990)
4. Fung, B.C.M., Wang, K., Chen, R., Yu, P.S.: Privacy-preserving data publishing: A survey of recent developments. ACM Comput. Surv. 42(4), 14:1–14:53 (2010)
5. Kantarcioğlu, M., Jin, J., Clifton, C.: When do data mining results violate privacy? In: Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 599–604. KDD ’04, ACM, New York, NY, USA (2004)
6. Kriegel, H.P., Kroger, P., Pryakhin, A., Schubert, M.: Effective and efficient distributed model-based clustering. In: Fifth IEEE International Conference on Data Mining (2005)
7. Kullback, S., Leibler, R.A.: On information and sufficiency. Ann. Math. Statist. pp. 79–86 (1951)
8. Lindell, Y., Pinkas, B.: Secure multiparty computation for privacy-preserving data mining (2008)
9. Martinelli, F., Saracino, A., Sheikhalishahi, M.: Modeling privacy aware information sharing systems: A formal and general approach. In: 15th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (2016)
10. Oliveira, S.R.M., Zaane, O.R.: Achieving privacy preservation when sharing data for clustering. In: In Proc. of the Workshop on Secure Data Management in a Connected World (SDM04) in conjunction with VLDB2004. pp. 67–82. Toronto,Canada (2004)
11. Sheikhalishahi, M., Mejri, M., Tawbi, N.: Clustering spam emails into campaigns. In: Library, S.D. (ed.) 1st Conference on Information Systems Security and Privacy (2015)
12. Sheikhalishahi, M., Saracino, A., Mejri, M., Tawbi, N., Martinelli, F.: Fast and effective clustering of spam emails based on structural similarity. In: 8th International Symposium on Foundations and Practice of Security (2015)
13. Sheikhalishahi, M., Mejri, M., Tawbi, N.: On the abstraction of a categorical clustering algorithm. In: Machine Learning and Data Mining in Pattern Recognition - 12th International Conference, MLDM 2016, New York USA, 16-21, 2016, Proceedings. pp. 659–675 (2016)
14. Zhan, Z.J.: Privacy-preserving collaborative data mining (2006), doctoral Dissertation