PurTreeClust: A Clustering Algorithm for Customer Segmentation from Massive Customer Transaction Data

Xiaojun Chen, *Member, IEEE*, Yixiang Fang, Min Yang, Feiping Nie, Zhou Zhao, and Joshua Zhexue Huang

Abstract—Clustering of customer transaction data is an important procedure to analyze customer behaviors in retail and e-commerce companies. Note that products from companies are often organized as a product tree, in which the leaf nodes are goods to sell, and the internal nodes (except root node) could be multiple product categories. Based on this tree, we propose the "personalized product tree", named purchase tree, to represent a customer's transaction records. So the customers' transaction data set can be compressed into a set of purchase trees. We propose a partitional clustering algorithm, named PurTreeClust, for fast clustering of purchase trees. A new distance metric is proposed to effectively compute the distance between two purchase trees. To cluster the purchase tree data, we first rank the purchase trees as candidate representative trees with a novel separate density, and then select the top k customers as the representatives of k customer groups. Finally, the clustering results are obtained by assigning each customer to the nearest representative. We also propose a gap statistic based method to evaluate the number of clusters. A series of experiments were conducted on ten real-life transaction data sets, and experimental results show the superior performance of the proposed method.

Index Terms—Customer segmentation, clustering transaction data, purchase tree, clustering trees

1 Introduction

Customer segmentation is very important for retail and e-commerce companies, because it is usually the first step towards the analysis of customer behaviors in these companies [1]. Early works use general variables like customer demographics and lifestyles [2], but such works are doubtful since the general variables are difficult to collect and some collected variables might be invalid soon without revision [3]. With the rapid increase of the collected customer behavior data, researchers now focus on clustering customers from transaction data [3], [4], [5].

Transaction data is the information recorded from daily transactions of customers, in which a transaction record contains a set of products (items) bought by a customer in one basket. There exist three problems for clustering of customer transaction data: 1) how to represent a customer with the associated transaction records, 2) how to compute the distance between different customers, and 3) how to segment the customers into specific number of customer groups. Recently, Hsu et. el. proposed a customer clustering method of transaction data [6]. In their method, the items are organized as a hierarchical tree structure which is called as concept hierarchy. They define a similarity measure based on path length and path depth on the concept hierarchy, and use hierarchical clustering to segment customers. However, the distance is defined on individual transaction record, thus the method suffers from the huge amount of transaction records. Moreover, the high computational complexity of hierarchical clustering hinders the using of their customer segmentation methods in real applications.

In this paper, we propose a novel PurTreeClust clustering algorithm for customer segmentation from massive customer transaction data. In the new method, we propose a new data representation method. We first build a *product tree*, where the leaf nodes usually denote the provided products and the internal nodes are multiple product categories. A product tree often consists of several levels and thousands of nodes, and the number of nodes increases rapidly from the top level to the bottom level. In transaction data, each product (item) bought by a customer corresponds to a leaf node in the product tree. To facilitate the analysis and visualization of customer's behavior, we build a "personalized product tree" for each customer, called *purchase tree*. The purchase tree can be built by aggregating all products in a customer's transactions, and pruning the

- *X. Chen and J. Z. Huang are with College of Computer Science and Software, Shenzhen University, Shenzhen 518060, P.R. China. E-mail: {xjchen, zx.huang}@szu.edu.cn.*
- *Y. Fang is with the Department of Computer Science, the University of Hong KongPokfulam Road, Hong Kong. E-mail: yxfang@cs.hku.hk.*
- *M. Yang is with Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen, Guangdong 518060, China. E-mail: min.yang1129@gmail.com.*
- *F. Nie is with School of Computer Science and Center for OPTical IMagery Analysis and Learning (OPTIMAL), Northwestern Polytechnical University, Xi'an, Shanxi 710072, P. R. China. E-mail: feipingnie@gmail.com.*
- *Z. Zhao is with School of Computing Science, Zhejiang University, Hangzhou, Zhejiang 310027, China. E-mail: zhaozhou@zju.edu.cn.*

Manuscript received 24 Nov. 2016; revised 15 Aug. 2017; accepted 2 Oct. 2017. Date of publication 26 Oct. 2017; date of current version 2 Feb. 2018. (Corresponding author: Xiaojun Chen.)
Recommended for acceptance by J. Gama.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TKDE.2017.2763620

product tree by only retaining the corresponding leaf nodes and all paths from root note to leaf nodes.

Since a customer usually buys a very small proportion of goods indicated by the product tree, the purchase tree is very sparse because it often contains a very smaller number of nodes compared to the product tree. For example, a retail chain supermarket may sell more than 30 thousands of different goods, but a customer might only buy less than 100 goods. The commonly-used distances such as Euclidean distance, Jaccard distance and cosine distance do not work for tree structure features. A natural way is to use the tree edit distance to compute the distance between two purchase trees, which computes the minimum cost of turning one tree to another tree by operations of deleting, inserting, and relabeling tree nodes [7]. However, the tree edit distance will produce high distance value between any two purchase trees because customers do not buy similar products. Therefore, it is difficult to recover cluster structure with the tree edit distance. To solve this problem, we have to effectively utilize the semantic in the product tree. In this paper, we define a new PurTree distance metric to compare customers from the entire levels of the product tree. The most important property of the new distance is that it is a metric, thus many advanced techniques can be used for the purchase tree data with the new distance.

Customer transaction data is usually very big, even after being compressed as a set of purchase trees. So the speed of the clustering method for customer transaction data is very important. The commonly-used hierarchical clustering method is hard to use in real applications due to its high computational complexity. Moreover, it is difficult to apply the fast k-means algorithm to the complex purchase tree data. In this paper, we propose PurTreeClust, a fast partitional clustering method for clustering of massive purchase tree data. In the new method, we propose a separate density to rank the purchase trees as candidate representative trees which are both dense and far from each other. Given a specified number of customer groups k, we select the top k customers as the representatives of k customer groups and perform clustering by assigning each customer to the nearest representative. To help the users selecting proper number of customer groups, we propose a gap statistic based method to evaluate the number of clusters.

The main contributions of this paper include:

- We define a purchase tree structure for compressing the customer transaction data set into a set of purchase trees.
- We propose a novel PurTree distance to compute the distance between two purchase trees. The new distance can consume both tree structure and semantic meaning of the tree hierarchy, and it is a metric.
- We propose a fast density estimation method for purchase tree data, and a separate density to rank the purchase trees as candidate representative trees which are both dense and far from each other.
- We propose PurTreeClust, a fast clustering method for clustering purchase tree data, which takes time at most $O((c^6 + k)n \log(n))$, where n is the number of purchase trees, k is the number of clusters and c is the expansion constant which is a measure of the intrinsic dimensionality [8].
- We propose a gap statistic based method for estimating the number of clusters from the purchase tree clustering results.

A series of experiments were conducted on ten real-life transaction data sets to investigate the properties of the new distance metric and the performance of the PurTreeClust algorithm. The experimental results have shown that the new algorithm is effective. We also compared our method with six clustering methods, the experimental results have demonstrated the superior performance of our method.

The rest of this paper is organized as follows. Section 2 presents a brief review of related work. Notations and preliminaries are given in Section 3. We present the PurTree distance metric in Section 4 for measure the distance between two Purchase trees. We define a separate density in Section 5.1 for selecting the representative trees and the PurTreeClust algorithm in Section 5 for clustering Purchase tree data sets with the separate density. The experimental results are reported in Sections 6. Conclusions and future work are given in Section 7.

2 Related Work

Clustering of customers, also called customer segmentation, is one of the most critical elements in achieving successful modern marketing and customer relationship management. The early segmentation methods use general variables like customer demographics, lifestyle, attitude and psychology, because such variables are intuitive and easy to operate [9]. With the rapid increase of customer behavior data, researchers turn to use product specific variables such as items purchased [3], [4], [10]. These methods often define distances on transaction records but computing distances with such distance functions are very slow due to the large number of transaction records. Recently, Hsu et. el. proposed a customer clustering method based on concept hierarchy of items, which defines a similarity measure based on path length and path depth on the concept hierarchy [6]. However, the distance is also computed based on transaction records. Given a distance function, hierarchical clustering and genetic algorithm are often used for clustering [3], [6], [11]. However, such methods cannot handle large-scale transaction data due to their high computational complexity.

In the past decade, many clustering methods have been proposed, including hierarchical clustering [12], subspace clustering [13], [14], [27], multi-view clustering [15], density-based clustering [16] and spectral clustering [17]. Since the distance functions defined on transaction data are different from the traditional distance such as Euclidean distance, the commonly-used k-means clustering method cannot be used for clustering [3], [6]. Given a similarity/distance matrix, hierarchical clustering, spectral clustering and density based clustering are three commonly-used clustering techniques.

Hierarchical clustering is a method of cluster analysis which seeks to build a hierarchy of clusters. Two commonly-used strategies for hierarchical clustering are agglomerative and divisive. In hierarchical agglomerative clustering, each object starts in a cluster, and pairs of clusters are recursively merged to one cluster. In hierarchical divisive clustering, all objects start in one cluster, and a cluster is split recursively

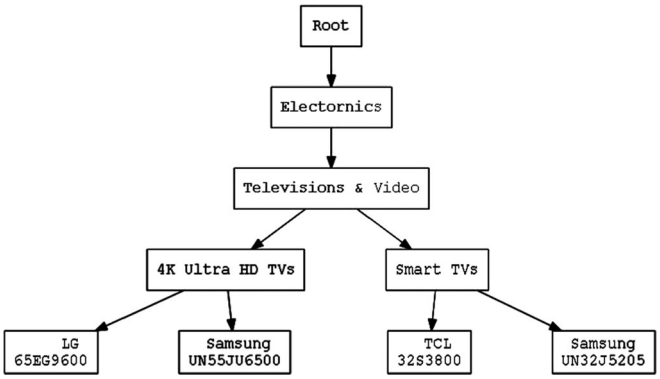


Fig. 1. A sample of product tree.

until it only contains one object. Hierarchical clustering often evolves merging or splitting subsets of objects rather than individual objects. Single link, average link, and complete link are three commonly-used distance functions to merge two clusters. Most hierarchical clustering algorithms have computational complexity at least $O(n^2)$ where n is the size of data. This high cost limits their applications in large-scale data. Although some improved hierarchical algorithms have computational complexity of $O(n \log(n))$ but they only work for numeric data [18].

Spectral clustering received a significant amount of attention in the last few years. Such methods make use of the spectrum (eigenvalues) of the similarity matrix of the data to perform dimensionality reduction, and then an iterative clustering algorithm is used to cluster the data with fewer dimensions [17], [19].

Density based clustering estimates the density of objects in a data set and uncovers clusters from high-density areas. In density-based clustering, a cluster is a set of data objects spread over a contiguous region of high density objects, and objects in low-density regions are typically considered noise or outliers [16]. The typical density based clustering is DBSCAN, which finds core objects with high density and connects close core objects to form clusters [20]. Density-based clustering can find arbitrary shape clusters and handle noise data well, but it is slow due to neighborhood search for each data object, and faces a difficulty in setting the density threshold parameters properly.

Recently, Rodriguez et al. proposed a density based clustering algorithm by fast searching and finding of density peaks [21]. The new method assumes that the central point of a cluster has higher density compared to its neighbors, and the cluster center is relatively far from other cluster centers compared to its local data points. For each data point, the new method computes a local density $\rho = \sum_{j=1}^{n} \chi(d_{ij} - d_c)$ where $\chi(a) = 1$ if $a < 0$ and $\chi(a) = 0$ otherwise and d_c is a cutoff distance, and a measure δ which is computed as the minimum distance between the point and any other point with higher density. To cluster the data set into k clusters, it first select k data points with the highest $\rho\delta$ as the cluster centers, and then performs clustering by assigning each data point to the nearest cluster center. However, it is difficult to set proper d_c. Moreover, it has a computational complexity of $O(n^2)$ where n is the number of objects.

3 Notations and Preliminaries

3.1 Notations

Let T be a tree with nodes $N(T)$ and edges $E(T) \in N(T) \times N(T)$. The root of T is denoted by $root(T)$. A node without children is a leaf node, and otherwise an internal node. For an edge $\{v, w\} \in E(T)$, node v is the parent and w is the child, i.e., $P_w(T) = v$ and $w \in C_w(T)$. The descendants of v, the nodes in all paths reachable from v to a leaf node, is denoted as $des(v)$. The level of a node is defined by 1 + (the number of edges between the node and the root). $N^l(T)$ represents nodes in the lth level of T. The height of tree T, denoted by $H(T)$, is the number of edges on the longest downward path between the root and a leaf node.

3.2 Transaction Data

Let $R = \{r_1, \ldots, r_m\}$ be a set of m transaction records and $I = \{item_1, \ldots, item_y\}$ be a set of y items. A transaction record r_i is an itemset, represented as $(x_1, \ldots, x_z)$ where $x_j \in I$ for $1 \leq j \leq z$.

3.3 Product Tree

Note that the products are often organized according to multiple types of categories. For example, a retailer may classify each product according to the business unit, the category, the subcategory, the product description and the brand. In this paper, we propose the product tree to systematically organize the products in a company. Let Ψ be a product tree, in which the root node $root(\Psi)$ is an empty node "Root", each leaf node represents an product (item) and the internal nodes represent categories used to organize the products. In this paper, we assume that all leaf nodes in Ψ have equal depth, which means that all products are organized with the same numbers of categories. Level l ($2 \leq l \leq H(\Psi)$) represents the lth category and the nodes in $N^l(\Psi)$ represent the items in the lth category. Usually, $|N^l(\Psi)| < |N^{l+1}(\Psi)|$. Here, $I = N^{H(\Psi)+1}(\Psi)$, i.e., a transaction record r is a subset of the leaf nodes in Ψ. Fig. 1 shows a sample of product tree which consists of two category levels.

3.4 Purchase Tree

With the product tree, we can compress a customer's transaction records as a purchase tree. A purchase tree φ is a subgraph of the product tree Ψ, i.e., $N(\varphi) \subseteq N(\Psi)$ and $E(\varphi) \subseteq E(\Psi)$. Given any node $v \in N(\varphi)$, there must exist a leaf node $w \in N(\varphi)$ such that the path from $root(\varphi)$ to w also exists in Ψ. For each purchase tree φ, $H(\varphi) = H(\Psi)$. For simplicity, we use $H(\Phi)$ to represent $H(\varphi)$. A purchase tree is used to illustrate the items bought by a customer, and it can be built from a customer's transaction data. Given a customer's m transaction records $R = \{r_1, \ldots, r_m\}$, the items in R can be aggregated as $I' \in I$. Then a purchase tree φ can be built from Ψ by only retaining the leaf nodes in I' and the internal nodes in the path from the root to each leaf node.

Fig. 2 shows an example of purchase tree, which is a subset of the product tree in Fig. 1. Usually, a purchase tree often contains a very small subset of products in a product tree. For example, the product tree in Fig. 1 consists of three products, but the purchase tree in Fig. 2 only contains two products. To evaluate the sparsity of a purchase tree, we

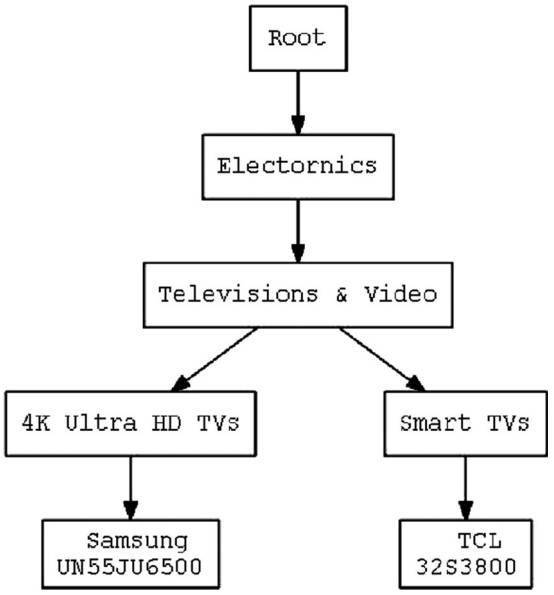


Fig. 2. A sample of purchase tree, built according to the product tree in Fig. 1.

define the sparsity of the lth level in φ_i, with respect to the product tree Ψ, as

$$\mathcal{S}^l_{\Psi}(\varphi_i) = \frac{|N^l(\varphi_i)|}{|N^l(\Psi)|}. \quad (1)$$

Given a set of n purchase trees $\Phi = \{\varphi_1, \ldots, \varphi_n\}$, and a product tree Ψ, the average sparsity of the lth level of Φ, denoted by $\overline{\mathcal{S}}^l_{\Psi}(\Phi)$, is defined as

$$\overline{\mathcal{S}}^l_{\Psi}(\Phi) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{S}^l_{\Psi}(\varphi_i). \quad (2)$$

3.5 Cover Tree

Cover tree, first invented by Beygelzimer et. al., is a leveled tree data structure for fast nearest neighbor operations in metric spaces [22]. Let CT be a cover tree on a data set S, as shown in Fig. 3. Each level in CT is indexed by an integer scale l which decreases from top to bottom. Let the cover set CS_l denote the set of objects in S associated with the nodes at level l. Each object in S appears at most once in each level, but may be associated with multiple nodes in the tree. CT obeys the following invariants for all levels:

- *Nesting*. $CS_l \in CS_{l-1}$. If an object $p \in S$ appears in CS_l then every lower level in the tree has a node associated with p.
- *Covering*. For each object $p \in CS_{l-1}$, there exists a $q \in CS_l$ such that $d(p,q) < 2^l$ and the node in level l associated with q is a parent of the node in level $l-1$ associated with p.
- *Separation*. For two different objects $r, q \in CS_l$, $d(r,q) > 2^l$.

where d is a metric defined on S. Let c be the *expansion constant* of S, which is a measure of the intrinsic dimensionality (as defined in [8]). CT has the following properties:

- The number of children of any node p is bounded by c^4;
- The maximum depth of any point p is $O(c^2, \log(n))$;

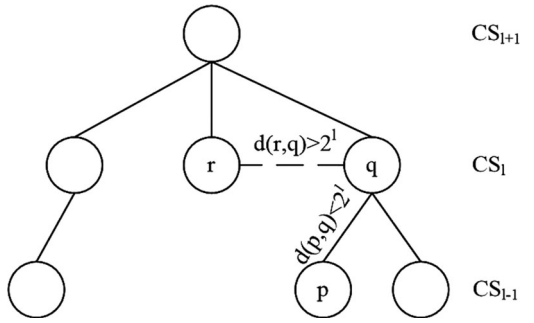


Fig. 3. An example of cover tree.

The basic operation to build a cover tree is the insertion operation, which is shown in Algorithm 1. Here, Q_l is a subset of the objects at level l which may contain the new object p as a descendant. If S consists of n objects, the computational complexity of inserting an object is $O(c^6 \log(n))$ [22]. A cover tree can be built on a data set S by sequentially applying Algorithm 1 to insert each object in S.

Algorithm 1. Insert (Object p, Cover Set Q_l, Level l)

Output: true if insertion succeeds, **false** otherwise.
1: Set $Q = \{C_q : q \in Q_l\}$.
2: Set $Q_{l-1} = \{q \in Q : d(p,q) \leq 2^l\}$.
3: **if** $Q_{l-1} = \emptyset$ **then**
4: return **false**.
5: **else**
6: **if** Insert$(p, Q_{l-1}, l-1)$ fails **then**
7: Pick $q = \arg\min_{q \in Q_l} d(p,q)$.
8: **if** $d(q,p) = 0$ **then**
9: exit.
10: **else**
11: insert p as a child of q.
12: return **true**.
13: **end if**
14: **else**
15: return **true**.
16: **end if**
17: **end if**

4 Distance Measure of Two Purchase Trees

Note that the commonly-used distances, such Euclidian distance, cannot be used to characterize the purchase tree data. Although the tree edit distance works for tree structure features, it can only capture the structure of a tree, the semantic meaning of the tree hierarchy will be ignored. To address this, we propose a new distance metric which can effectively compute the difference between two purchase trees.

Given a product tree Ψ and a set of n purchase trees $\Phi = \{\varphi_1, \ldots, \varphi_n\}$ where $\varphi_i \in \Psi$, let $H(\Phi)$ be the height of these purchase trees, $root(\varphi)$ be the empty root node of the purchase tree. Since a purchase tree φ is a subgraph of Ψ, we use the nodes in Ψ to index the nodes in a purchase tree φ. Let $C_v(\varphi_i)$ be children nodes of v in φ_i, and $N^l(\varphi_i)$ be all nodes in the lth level of φ_i. Given two purchase trees φ_i, φ_j and node $v \in \Psi$, we first compute the node distance defined on v as $\delta_v(\varphi_i, \varphi_j)$. If $v \notin \varphi_i$ and φ_j, the node distance $\delta_v(\varphi_i, \varphi_j) = 0$. If $v \in \varphi_i$ or φ_j, $\delta_v(\varphi_i, \varphi_j)$ is the Jaccard distance defined on the children $C_v(\varphi_i)$ and $C_v(\varphi_j)$:

$$\delta_v(\varphi_i,\varphi_j)=\frac{|C_v(\varphi_i)\Delta C_v(\varphi_j)|}{|C_v(\varphi_i)\bigcup C_v(\varphi_j)|}, \tag{3}$$

where $C_v(\varphi_i)\Delta C_v(\varphi_j)=C_v(\varphi_i)\bigcup C_v(\varphi_j)-C_v(\varphi_i)\bigcap C_v(\varphi_j)$ is the symmetric difference of two children nodes. $\delta_v(\varphi_i,\varphi_j)$ is bounded in $[0,1]$: $\delta_v(\varphi_i,\varphi_j)=0$ if $C_v(\varphi_i)\bigcap C_v(\varphi_j)=C_v(\varphi_i)\bigcup C_v(\varphi_j)$; $\delta_v(\varphi_i,\varphi_j)=1$ if $C_v(\varphi_i)\bigcap C_v(\varphi_j)=\emptyset$.

Definition 1. *The level tree distance between two purchase trees φ_i and φ_j on the lth level is defined as*

$$d^l(\varphi_i,\varphi_j)=\sum_{v\in N^l(\varphi_i)\cup N^l(\varphi_j)}\beta_v\delta_v(\varphi_i,\varphi_j), \tag{4}$$

where $1\le l\le H(\varphi)$ and β_v is the weight for node v. Since the nodes in Φ are organized in a tree structure, a natural way to define β_v as follows

$$\beta_v=\begin{cases}1 & \text{if } v=root(\Phi)\\ \frac{\beta_w}{|C_w(\varphi_i)\bigcup C_w(\varphi_j)|} & where\ v\in C_w(\varphi_i)\bigcup C_w(\varphi_j),\end{cases} \tag{5}$$

where w is the parent node of v and β_w has the definition as β_v.

Definition 2. *The PurTree distance between two purchase trees $\varphi_i,\varphi_j\in\Phi$ is computed as a weighted combination of the level tree distances in (4),*

$$d(\varphi_i,\varphi_j,\gamma)=\sum_{l=1}^{H(\Phi)}\omega_l d^l(\varphi_i,\varphi_j), \tag{6}$$

where ω_l is the lth level weight. $\{\omega_1,\ldots,\omega_{H(\Phi)}\}$ is a geometric sequence with common ratio γ ($\gamma>=0$) under constraints $\sum_{l=1}^{H(\Phi)}\omega_l=1$, defined as

$$\omega_l=\begin{cases}\frac{1-\gamma}{1-\gamma^{H(\Phi)}}\gamma^{l-1} & for\ \gamma>0\ and\ \gamma\neq 1\\ \frac{1}{H(\Phi)} & for\ \gamma=1\\ 1 & for\ \gamma=0\ and\ l=1\\ 0 & for\ \gamma=0\ and\ 1<l\le H(\Phi).\end{cases} \tag{7}$$

The distance metric $d(\varphi_i,\varphi_j,\gamma)$ can be adjusted by setting different γ as follows:

- If $\gamma=0$, $\omega_1=1$ and other weights are 0, thus only the top level L_1 contributes to the distance;
- If $0<\gamma<1$, $\omega_{l+1}=\gamma\omega_l<\omega_l$, thus a parent level contribute more to the distance than its child level;
- If $\gamma=1$, $\omega_l=\frac{1}{H(\Phi)}$, thus all levels contribute equally to the distance;
- If $\gamma>1$, $\omega_{l+1}=\gamma\omega_l>\omega_l$, thus a child level contribute more to the distance than its parent level;
- If $\gamma\to\infty$, $\omega_{H(\Phi)}=1$ and other weights are 0, thus only the bottom level $L_{H(\Phi)}$ contributes to the distance.

To set proper parameter γ, a simple way is to compute the pairwise distances between pairs of objects with different γ and choose proper γ according to the distribution of the pairwise distances (see Section 6.2 for example). $d(\varphi_i,\varphi_j,\gamma)$ can be simplified as $d(\varphi_i,\varphi_j)$ if we do not care about the value of γ.

Lemma 1. *Given n purchase trees $\Phi=\{\varphi_1,\ldots,\varphi_n\}$, for two purchase trees φ_i,φ_j in Φ and $1\le l\le H(\Phi)$, $d^l(\varphi_i,\varphi_j)\in[0,1]$.*

Proof. Since $\delta_v(\varphi_i,\varphi_j)$ lies in $[0,1]$, we can easily verify that $d^l(\varphi_i,\varphi_j)\ge 0$ and

$$\begin{aligned}d^l(\varphi_i,\varphi_j)&=\sum_{v\in N^l(\varphi_i)\cup N^l(\varphi_j)}\beta_v\delta_v(\varphi_i,\varphi_j)\\&\le\sum_{v\in N^l(\varphi_i)\cup N^l(\varphi_j)}\beta_v\\&=1.\end{aligned}$$

This completes the proof. □

Form the above lemma, we can easily prove the following theorem:

Theorem 1. *Given n purchase trees $\Phi=\{\varphi_1,\ldots,\varphi_n\}$, for two purchase trees φ_i and $\varphi_j\in\Phi$, $d(\varphi_i,\varphi_j,\gamma)\in[0,1]$.*

Lemma 2. *If $1\le l<H(\Phi)$, $d^{l+1}(\varphi_i,\varphi_j)\ge d^l(\varphi_i,\varphi_j)$.*

Proof. We denote $C_v(\varphi_i)\Delta C_v(\varphi_j)$ as $\triangle^v$ for node v, and $C_v(\varphi_i)\cap C_v(\varphi_j)$ as $\cap^v$, then $\delta_v(\varphi_i,\varphi_j)$ can be rewritten as

$$\delta_v(\varphi_i,\varphi_j)=\frac{|\triangle^v|}{|\triangle^v|+|\cap^v|}. \tag{8}$$

The sum of the distances on v's children nodes is

$$\begin{aligned}\sum_{w\in C_v(\varphi_i)\cup C_v(\varphi_j)}\delta_w(\varphi_i,\varphi_j)&=\sum_{w\in\triangle^v}\delta_w(\varphi_i,\varphi_j)\\&\quad+\sum_{w\in\cap^v}\delta_w(\varphi_i,\varphi_j).\end{aligned} \tag{9}$$

For $w\in\triangle^v$, we have $C_w(\varphi_i)\cap C_w(\varphi_j)=\emptyset$, thus $\delta_w(\varphi_i,\varphi_j)=1$. For $w\in\cap^v$, $\delta_w(\varphi_i,\varphi_j)\ge 0$. Then we give

$$\begin{aligned}\sum_{w\in C_v(\varphi_i)\cup C_v(\varphi_j)}\delta_w(\varphi_i,\varphi_j)&\ge|\triangle^v|\\&=(|\triangle^v|+|\cap^v|)\delta_v(\varphi_i,\varphi_j).\end{aligned} \tag{10}$$

Introducing the node weight β defined in (5) into (10) gives

$$\textstyle\sum_{w\in C_v(\varphi_i)\cup C_v(\varphi_j)}\beta_w\delta_w(\varphi_i,\varphi_j)\quad\ge\beta_v\delta_v(\varphi_i,\varphi_j). \tag{11}$$

Then we have

$$\begin{aligned}d^{l+1}(\varphi_i,\varphi_j)&=\sum_{v\in N^{l+1}(\varphi_i)\cup N^{l+1}(\varphi_j)}\beta_v\delta_v(\varphi_i,\varphi_j)\\&=\sum_{v\in N^l(\varphi_i)\cup N^l(\varphi_j)}\sum_{w\in C_v}\beta_w\delta_w(\varphi_i,\varphi_j)\\&\ge\sum_{v\in N^l(\varphi_i)\cup N^l(\varphi_j)}\beta_v\delta_v(\varphi_i,\varphi_j)\\&=d^l(\varphi_i,\varphi_j).\end{aligned} \tag{12}$$

□

With Lemma 2, we can easily prove the following theorem:

Theorem 2. *Given n purchase trees $\Phi=\{\varphi_1,\ldots,\varphi_n\}$, for two purchase trees φ_i and $\varphi_j\in\Phi$. If $1\le l\le t\le H(\Phi)$, $d^l(\varphi_i,\varphi_j)\le d^t(\varphi_i,\varphi_j)$.*

The above theorem ensures that the level PurTree distance increases with the increase of tree level, which satisfies our intuition.

Theorem 3. *$d(\varphi_i, \varphi_j)$ defined in (6) is a metric.*

Proof. Since the level tree distance $\delta_v(\varphi_i, \varphi_j)$ is Jaccard distance, which is a metric on the collection of all finite sets [23], we can easily verify that $d(\varphi_i, \varphi_j) = d(\varphi_j, \varphi_i)$ (Symmetry), $d(\varphi_i, \varphi_i) = 0$ (Reflexivity) and $d(\varphi_i, \varphi_j) \geq 0$ for all φ_i and φ_j in Φ (Positivity).

Given three Purchase trees φ_i, φ_j and φ_t,

$$\begin{aligned} d(\varphi_i, \varphi_j) + d(\varphi_j, \varphi_t) &= \sum_{l=1}^{H(\Phi)} \omega_l \sum_{v \in N^l(\varphi_i) \cup N^l(\varphi_j)} \beta_v \delta_v(\varphi_i, \varphi_j) \\ &\quad + \sum_{l=1}^{H(\Phi)} \omega_l \sum_{v \in N^l(\varphi_j) \cup N^l(\varphi_t)} \beta_v \delta_v(\varphi_j, \varphi_t) \\ &\geq \sum_{l=1}^{H(\Phi)} \omega_l \sum_{v \in N^l(\varphi_i) \cup N^l(\varphi_t)} \beta_v \delta_v(\varphi_i, \varphi_t) \\ &= d(\varphi_i, \varphi_t), \end{aligned}$$

which indicates that the distance satisfies triangle inequality. Therefore, it is a metric. □

Compared to the conventional distances such as Jaccard distance and tree edit distance, the new distance can consume both tree structure and semantic meaning of the tree hierarchy. Jaccard distance only works for flat data so it can not consume tree structure. Although tree edit distance explores tree structure, it ignores the differences among different tree levels. Moreover, the new distance is proved to be a metric. Therefore, many transaction data clustering algorithms can benefit from the new distance compared to the conventional distances such as Jaccard distance, tree edit distance and concept similarity.

5 A Purchase Tree Clustering Algorithm

Since a product tree often consists of tens of thousands of items, we need a fast clustering algorithm for purchase tree data. The commonly-used hierarchical clustering method is hard to use in real applications due to its high computational complexity. Moreover, the popular k-means algorithm cannot be applied for clustering the complex purchase tree data. Inspired by the recent work in [21], we propose a fast clustering algorithm for clustering of massive purchase tree data. In the new method, we propose a separate density to rank the purchase trees as candidate representative trees which are both dense and far from each other. Given a specified number of customer groups k, we select the top k customers as the representatives of k customer groups and perform clustering by assigning each customer to the nearest representative tree. We also propose a gap statistic based method for estimating the number of clusters from the purchase tree clustering results.

5.1 Ranking Candidate Cluster Centers via Separate Density

To select representative trees which are both dense and far from each other, we have to estimate the density of purchase tree data. The key task for density estimation is the k-nearest neighbor search, and the commonly-used k-d tree cannot handle complex purchase tree data. Note that the PurTree distance is proven to be a metric and cover tree works in metric spaces, we propose to estimate the density of purchase tree with cover tree.

The algorithm to build a cover tree CT from a set of n purchase trees $\Phi = \{\varphi_1, \ldots, \varphi_n\}$ is presented in Algorithm 2. We have shown in Theorem 1 that $d(\varphi_i, \varphi_j) \in [0, 1]$. From the separation invariant, we know that there is at most one purchase tree in level 0 and other trees lie behind level 0. Therefore, we insert each purchase tree into a cover tree by applying $Insert(\varphi_i, CS_0, 0)$ where CS_0 is the root node of CT. Since the computational complexity of Algorithm 1 is $O(c^6 \log(n))$, the overall computational complexity of Algorithm 2 is $O(c^6 n \log(n))$.

Algorithm 2. BuildCoverTree (Purchase Tree Data $\Phi = \{\varphi_1, \ldots, \varphi_n\}$)

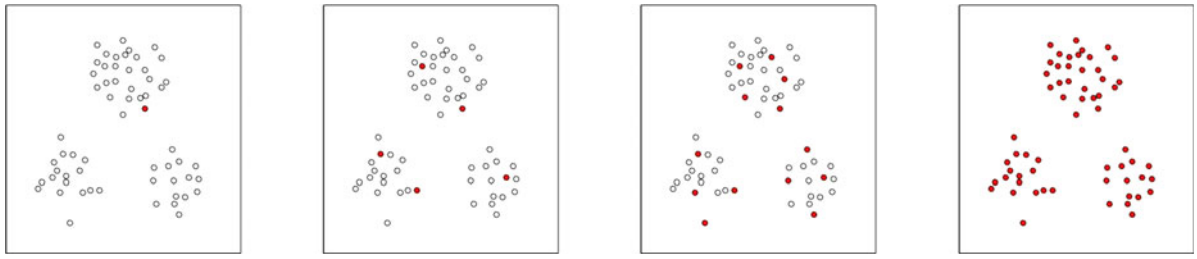
Output: A cover tree CT built on Φ.
1: Initialize an empty cover tree CT, in which CS_0 is the cover set at level 0.
2: **for** $i = 1; i <= n; i++$ **do**
3: $Insert(\varphi_i, CS_0, 0)$.
4: **end for**

With the cover tree CT, we want to find k purchase trees as k representative trees for clustering. To be able to separate these trees and form compact clusters, the k representative trees should be dense and far away from each other. If we simply compute the density for each object and select k densest objects as representative trees, they may be too close to form separate clusters. Recall the separation invariant, for two objects p and q in CS_l, $d(p, q) > 2^l$, where 2^l can be consider as the distance threshold for CS_l and it increases with the increase of the tree level l. Therefore, we can select the objects in level $l = \arg\max_l |CS_l| > k$ as candidate representative objects. Fig. 4 shows an example of four levels of cover sets, in which we can see that the initial cluster centers can be selected from five objects in cover set CS_{-1}. The next step is to estimate the density of the objects in CS_l. We define a cover tree based density as follows:

Definition 3. *Given a cover tree CT built on Φ with a metric d, the level density of an object $p \in CS_l$ is $den^l_{CT}(p) = |\{q \in \Phi \backslash \{p\} : d(p, q) \leq 2^l\}|$.*

The level density estimation algorithm is shown in Algorithm 3. Given an object $p \in CS_l$, to find all objects q in Φ such that $d(p, q) \geq 2^l$, we descend through the tree level by level, keeping track of a subset $Q_i \in CS_i$ of nodes that may contain q as a descendant. The algorithm iteratively constructs Q_{i-1} by expanding Q_i to its children in CS_{i-1} and throwing away any child q that $d(p, q) > 2^l$. For any object $q \in \Phi$, $des(q)$ can be computed during the building tree procedure thus no more time is required during the density estimation procedure.

Theorem 4. *Given a cover tree CT built on Φ with a metric d, for any object $p \in CS_l$, Algorithm 3 returns $den^l(p)$ defined in Definition 3.*



(a) Points in CS_0. (b) Points in CS_{-1}. (c) Points in CS_{-2}. (d) Points in CS_{-3}.

Fig. 4. Cover sets at different levels (red points are the objects in the cover sets and empty points are the left objects).

Proof. For any $q \in CS_i$, the distance between q and any descendant q' is bounded by $d(q,q') < \sum_{j=i}^{-\infty} 2^j = 2^{i+1}$. For any $q \in Q_i$, Algorithm 3 checks $d(p,q)$ and processes C_q as follows:

- If $d(p,q) \leq 2^l - 2^{i+1}$, the distance between p and $q' \in des(q)$ is bounded by $d(p,q') \leq d(p,q) + d(q,q') < 2^l$, i.e., $q \cup des(q) \in den^l_{CT}(p)$;
- If $d(p,q) \geq 2^l + 2^{i+1}$, the distance between p and $q' \in des(q)$ is bounded by $d(p,q') \geq d(p,q) - d(q,q') > 2^l$, then $q \cup des(q) \notin den^l_{CT}(p)$;
- If $d(p,q) \in (2^l - 2^{i+1}, 2^l + 2^{i+1})$, the distance between p and $q' \in C_q$ is bounded by $d(p,q') \in (2^l - 2^{i+2}, 2^l + 2^{i+2})$. Then C_q will be added into Q_{i-1}.

Therefore, Algorithm 3 can never throw out a grandparent $q' \in des(p)$. □

Algorithm 3. EstimateDensity (Cover Tree CT, Object $p \in CS_l$, Level l)

Output: $den^l_{CT}(p)$.
1: Set $den^l_{CT}(p) = 0$ and $Q_l = CS_l$.
2: **for** $i = l; i < -\infty; i--$ **do**
3: $Q = \{C_q : q \in Q_i\}$.
4: Set $\Lambda_{i-1} = \{q \in Q : d(p,q) \leq 2^l - 2^{i+1}\}$.
5: **for** each $q \in \Lambda_{i-1}$ **do**
6: $den^l_{CT}(p) += |des(q)|$.
7: **end for**
8: $Q_{i-1} = \{q \in Q : d(p,q) \in (2^l - 2^{i+1}, 2^l + 2^{i+1})\}$.
9: $den^l_{CT}(p) += |\{q \in Q - Q_{i-1} : d(p,q) \leq 2^l\}|$.
10: **end for**
11: $den^l_{CT}(p) += |\{q \in Q_{-\infty} : d(p,q) \leq 2^l\}|$.
13: **return** $den^l_{CT}(p)$.

In Step 3, the number of children encountered is at most $O(c^4|Q_i|)$. Step 3-3 takes time at most $O(c^4|Q_i|)$, and Step 3 requires $O(\max_i|Q_i|)$ time. Consequently, the running time is bounded by $O(c^4 \sum_{i=l}^{-\infty} |Q_i|)$. If in Step 4, $Q_{l-2} = \emptyset$, then Algorithm 3 takes time $O(1)$. In the worst case, if $Q_{-\infty} = \Phi \backslash \{p\}$, Algorithm 3 takes time at most $O(n \log(n))$.

Although the objects with high level density are supposed to be far from each other, if we select representative trees only according to level density, the selected representative trees may be concentrated on some small dense areas and dense objects in sparse area will be ignored. To further ensure that the selected dense objects are far away from each other, we define the separate distance as follows:

Definition 4: *Given a cover tree CT built on Φ with a metric d, the separate distance $sdis^l_{CT}(p)$ of an object $p \in CS_l$ is defined as the minimal distance between p and other objects in CS_l which has higher level density than p, i.e., $sdis^l_{CT}(p) = \min_{q \in CS_l, den^l_{CT}(q) > den^l_{CT}(p)} dis(p,q)$. If p has the highest level density, its separate distance is defined as $sdis^l_{CT}(p) = \max_{q \in CS_l} d(p,q)$.*

In a dense area, the densest object will be assigned with a high separate distance because the denser object is supposed to be far away from this area, while the other objects will be assigned with a small separate distance because the denser object is supposed to be near them. Finally, to select objects which are dense and far away from each other, we combine the level density and separate distance to form the separate density as follows:

Definition 5. *Given a cover tree CT built on Φ with a metric d, the separate density $sden^l_{CT}(p)$ of an object $p \in CS_l$ is defined as $sden^l_{CT}(p) = sdis^l_{CT}(p) * den^l_{CT}(p)$.*

According to Definition 5, the objects with high separate densities should be both dense and far from each other. The separate density can be easily computed with the estimated level density. Given a specified number of customer groups k, we select the top k customers with the highest separate densities as the representatives of k customer groups. Fig. 5 show the level densities and separate distances of five objects in CS_{-1} of Fig. 4b. From this figure, we can see that objects 1, 3 and 5 have both higher level densities and separate distances than other objects. Therefore, they will be selected as cluster centers. The next step is to perform clustering with the selected representative trees, which will be presented in the next section.

5.2 The PurTreeClust Algorithm

In this section, we propose a partitional clustering algorithm PurTreeClust, in which the separate density defined in the previous section is used for selecting the representative trees.

Given a cover tree CT built from n purchase trees, we want to cluster the n trees into k clusters. The basic method is to find k purchase trees as representative trees, and then assign each customer to the nearest representative tree. We first select a proper level l where $|CS_l| > k$, estimate their densities with Algorithm 3 and compute their separate densities, then the k representative trees can be selected as the k trees with the highest separate densities. If the cover tree is well distributed, i.e., the number of objects in cover set increases evenly with the decrease of the tree level, we can select a proper level l such that $|CS_l|$ is not too much bigger than k. However, in the worst case, $|CS_l|$ may be close to n.

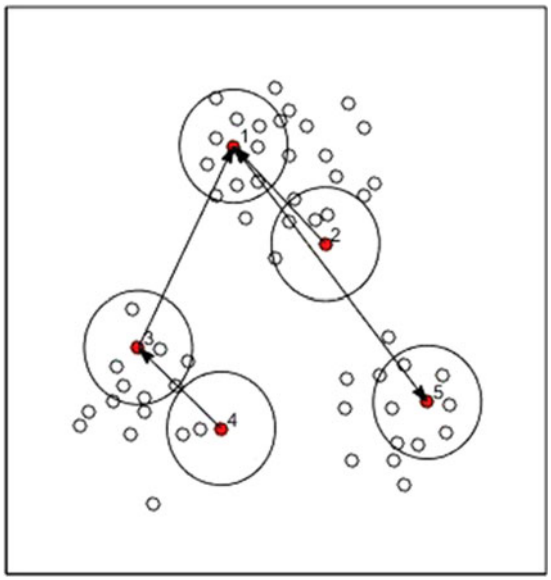


Fig. 5. The level densities and separate distances of five objects in CS_{-1} of Fig. 4b, in which the number of objects in the circle around a red point is its level density and the arrow starting from a red point is its separate distance.

In such case, we can set Q as $2k$ objects in CS_l with the biggest value of descendants. The details of the procedure is shown in Algorithm 4.

Algorithm 4. PurTreeClust (Cover Tree CT, no. of Clusters k)

Output. the clustering result.
1: // Select k representative trees.
2: $Q_0 = CS_0$, where CS_0 is the root level of CT.
3: **for** $l = 0$; $|Q_l| < 2k$; $l--$ **do**
4: $Q_{l-1} = \{C_q : q \in Q_l\}$.
5: **end for**
6: Set $Q = \{q \in Q_l : \forall q' \notin Q, |des(q)| > |des(q')|\}$ such that $|Q| = 2k$.
7: **for** each object $q \in Q$ **do**
8: Compute $den^l(q)$ via EstimateDensity(CT, q, l).
9: $sdis^l_{CT}(q) = \min_{p \in CS_l, den^l_{CT}(p) > den^l_{CT}(q)} dis(p, q)$.
10: $sden^l_{CT}(q) = sdis^l_{CT}(q) * den^l_{CT}(q)$.
11: **end for**
12: Set $U = \{q \in Q : \forall q' \notin U, sden^l(q) > sden^l(q')\}$ such that $|U| = k$.
13: // Clustering.
14: **for** $i = 1; i <= n; i++$ **do**
15: $P_i = \arg_{j, \varphi_j \in U} \min d(\varphi_j, \varphi_i)$.
16: **end for**
17: 18: **return** $\mathcal{P}$.

Given a set of n purchase trees, a cover tree CT can be built with Algorithm 2, which takes time at most $O(c^6 n \log(n))$. Then Algorithm 4 will return an expected clustering result with given k, which takes time at most $O(kn \log(n))$, so the overall procedure takes time at most $O(c^6 n \log(n) + kn \log(n))$, implying a bound of $O((c^6 + k) n \log(n))$. Compared with the method in [21], the new method eliminates the difficult in selecting proper cutoff distance d_c and reduces the computational complexity from $O(n^2)$ to $O(n, \log n)$. Moreover, the new method can quickly respond to changing business requirements. Once a cover tree is built, we can quickly cluster the data into different numbers of clusters. A new customer can be directly inserted into the built cover tree without rebuilding the whole tree.

5.3 Choosing the Number of Clusters

Given a transaction data, we can select a set of k (number of clusters) to run the PurTreeClust clustering algorithm to produce multiple clustering results. To select the best number of clusters, we extend the gap statistic to evaluate the number of clusters.

Gap statistic is a popular method for choosing the number of clusters [24]. It computes the change in within-cluster dispersion with that expected under an appropriate reference null distribution. Given a purchase tree data set Φ, let $\log(W(k))$ be the within-cluster dispersion with k clusters computed as

$$\log(W(k)) = \log\left\{\sum_{l=1}^{k} \frac{1}{2|P_l|} \sum_{i,j \in P_l} d(\varphi_i, \varphi_j)\right\}, \tag{13}$$

where $\{P_1, \ldots, P_k\}$ is the clustering result of k partitions.

Let $\Upsilon = \{\Phi'_1, \ldots, \Phi'_B\}$ be a set of B sample data sets generated from a null reference distribution, which is assumed to contain a single component. We can cluster these data sets and compute $\{\log(W_1(k), \ldots, \log(W_B(k))\}$ from the clustering results. The gap statistic is computed as the difference between the expectation of $\log(W(k))$ and $\log(W(k))$

$$Gap(k) = \frac{1}{B} \sum_{j=1}^{B} \log(W_j(k)) - \log(W(k)). \tag{14}$$

If the data consists of k^* well-separated clusters, we expect $\log(W(k))$ to decrease slowly when $k << k^*$ or $k >> k^*$; while decreasing slightly faster when k is close to k^*. In real applications, we can predefine a set of k, compute a set of $Gap(k)$ and draw them in a figure. Then the optimal number of clusters can be selected as the value of k for which $\log(W(k))$ falls the farthest below this reference curve, i.e., the biggest value of $Gap(k)$.

Given a set of n purchase trees $\Phi = \{\varphi_1, \ldots, \varphi_n\}$ and a set of numbers of clusters $K = \{k_1, \ldots, k_t\}$, the algorithm to compute t gap statistic values is shown in Algorithm 5. We first cluster Φ with different numbers of clusters and compute $\{\log(W(1)), \ldots, \log(W(t))\}$. To generate B reference data sets, we first compute the average sparsity of n purchase trees $\overline{\mathcal{S}}_\Psi(\Phi)$ and sample B purchase tree data sets $\{ST_1, \ldots, ST_B\}$. Each purchase tree in each reference data set consists of m sample purchase trees, whose nodes are uniformly sampled from Ψ, starting from level 2 down to level $H(\Psi)$+1. In the lth level, $|N^l(\Psi)| * \overline{\mathcal{S}}^l_\Psi(\Phi)$ nodes are uniformly sampled from $\{v \in N^l(\Psi) : P_v \in N^{l-1}(\Psi)\}$.

6 Experiments Results and Analysis

In this section, we present experimental results on real-life customer transaction data sets to investigate the properties of the PurTreeClust algorithm, and compare its effectiveness and scalability with related methods.

TABLE 1
Characteristics and Average Sparsity of Ten Customer Transaction Data Sets

Data	#Customers	#Products	$\mathcal{S}_\Psi^1$	$\mathcal{S}_\Psi^2$	$\mathcal{S}_\Psi^3$	$\mathcal{S}_\Psi^4$
D_1	795	1,264	89.895%	40.229%	0.832%	
D_2	208	9,760	15.087%	4.472%	2.684%	0.599%
D_3	416	14,274	16.293%	5.013%	3.065%	0.559%
D_4	832	18,137	15.130%	4.191%	2.582%	0.403%
D_5	1,665	22,822	15.456%	4.018%	2.411%	0.318%
D_6	3,330	28,065	15.641%	3.848%	2.314%	0.270%
D_7	2,433	93,731	23.401%	0.448%	0.386%	
D_8	4,867	96,538	23.215%	0.437%	0.374%	
D_9	9,735	99,907	23.347%	0.426%	0.362%	
D_{10}	19,471	103,337	23.187%	0.414%	0.0.348%	

Algorithm 5. Gap (Product Tree Ψ, n Purchase Trees $\Phi = \{\varphi_1, \ldots, \varphi_n\}$, a set of t Numbers of Clusters $K = \{k_1, \ldots, k_t\}$, Number of Copies B)

Output: A set of t gap statistic results.

$CT = BuildCoverTree(\Phi)$.
for $i = 1; i <= t; i++$ **do**
  $P_i = PurTreeClust(CT, k_i)$.
  Compute $\log(W(k_i))$ from P_i.
end for
Compute the average sparsity of n purchase trees $\overline{\mathcal{S}}_\Psi(\Phi) = \{\overline{\mathcal{S}}_\Psi^1(\Phi), \ldots, \overline{\mathcal{S}}_\Psi^{H(\Psi)}(\Phi)\}$.
for $j = 1; j <= B; j++$ **do**
  for $i = 1; i <= m; i++$ **do**
    $N^1(ST_{ij}) = \{\text{“}Root\text{”}\}$
    for $l = 2; j <= H(\Psi), l++$ **do**
      Uniformly sample $|N^l(\Psi)| * \overline{\mathcal{S}}_\Psi^l(\Phi)$ nodes from $\{v \in N^l(\Psi) : P_v \in N^{l-1}(\Psi)\}$ to form $N^l(ST_{ij})$.
    end for
  end for
  $SCT_j = BuildCoverTree(\{ST_{1j}, \ldots, ST_{mj}\})$.
end for
for $i = 1; i <= t; i++$ **do**
  for $j = 1; j <= B; j++$ **do**
    $P_i = PurTreeClust(SCT_j, k_i)$.
    Compute $\log(W_j(k_i))$ from P_i.
  end for
  $Gap(k_i) = (1/B)\sum_{j=1}^{B} \log(W_j(k_i)) - \log(W(k_i))$
end for
24: **return**$\{Gap(k_1), \ldots, Gap(k_t)\}$.

6.1 Transaction Data

In this section, we used ten real-life transaction data sets to investigate the effectiveness and scalability of the proposed algorithm. D_1 was built from a superstore's transactional data set,[1] which consists of 8,399 transaction records from 795 customers. $D_2 \sim D_6$ were built from five subsets of a super market's transactional data, which contains up to 25 million newest transaction records from 2013 to 2015. $D_7 \sim D_{10}$ were built from four subsets of a year of purchase history transaction data from the kaggle competition,[2] which consists of more than 349 million transaction records from more than 300 thousands customers. We computed the average sparsity for the ten data sets and their characteristics are shown in Table 1. We can see that the sparsity decrease rapidly from top level to bottom level, which indicates that most customers only bought a very small proportion of the total products.

1. https://community.tableau.com/docs/DOC-1236
2. http://www.kaggle.com/c/acquire-valued-shoppers-challenge/data

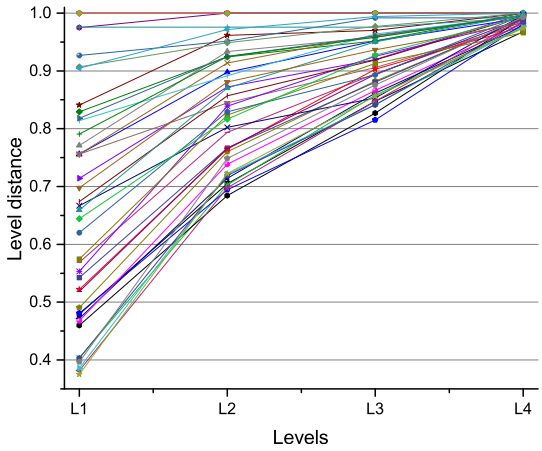
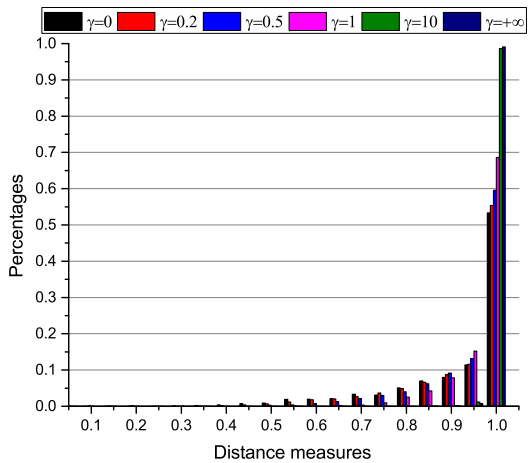


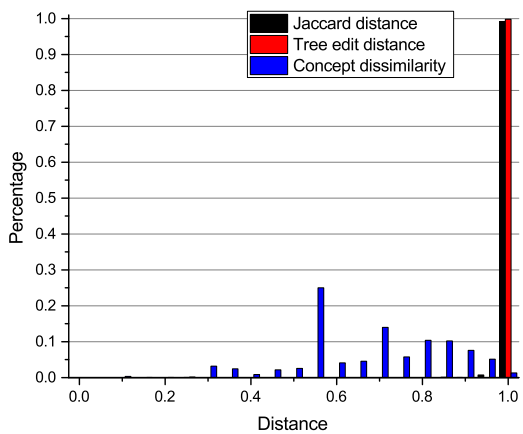
Fig. 6. Samples of the level PurTree distances on D_6.

6.2 The Properties of the PurTree Distance

We have shown in Theorem 2 that the level PurTree distance increases with the increase of tree level. To verify this



(a) Distance distributions by the PurTree distance.



(b) Distance distributions by the Concept dissimilarity, Jaccard distance and Tree edit distance.

Fig. 7. Comparison of four distance distributions on D_6.

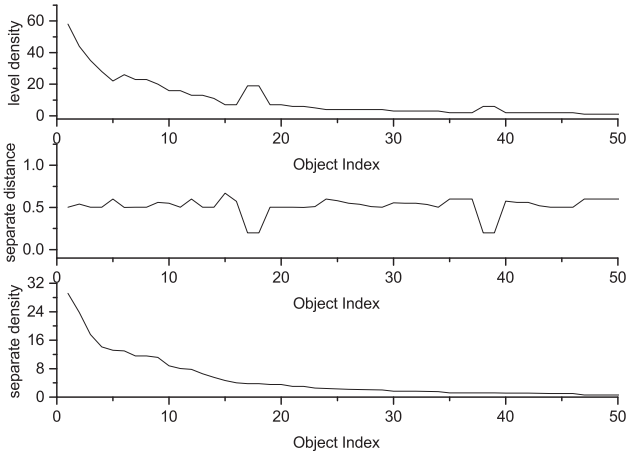


Fig. 8. Separate density on D_6.

theorem, we computed the level tree distance of pair objects on D_6 and drew some samples in Fig. 6. We can see that the level tree distances between all pairs of objects strictly increased with the increase of tree level. Distances on L_1 mainly fell in the range of $(0.35, 1]$, and distances on L_4 mainly fell in the range of $[0.95, 1]$.

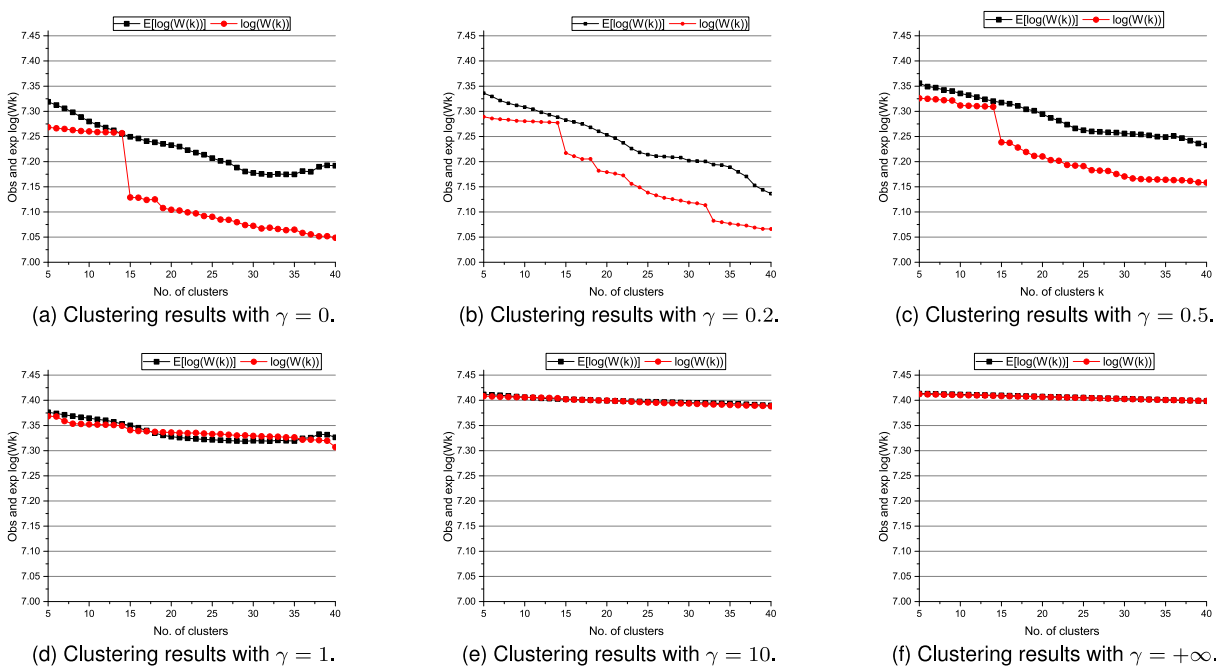
In the PurTree distance defined in (6), a positive distribution parameter γ is introduced to adjust the contributions of the purchase tree's different levels to the final distance. To investigate the effect of γ on the PurTree distance, we set $\gamma = \{0, 0.2, 0.5, 1, 10, +\infty\}$ and computed pairwise PurTree distances between pairs of objects on D_4. The distribution of the pairwise distances is drawn in Fig. 7a. From this figure, we can see that as γ increased, the distance between two purchase trees became closer to 1, thus the objects being far away from each other. As γ decreased, the distances fell more evenly in $[0, 1]$. If γ is too big, e.g., 10, the distances between objects will be too big to find compact clusters. But if γ is too small, e.g., 0, the distances between objects may be too small thus they are too close to be separated. In real applications, we can choose proper γ according to the distribution of pairwise distances between pairs of objects.

We also computed the Jaccard distance (only compute distances on leaf nodes of the purchase tree), tree edit distance [7] and concept dissimilarity [6] on pairs of objects in D_6. The distributions of the distances are drawn in Fig. 7b. From this figure, we can see that more than 99 percent of the distances by the Jaccard distance and tree edit distance fell in [0.9,1.0], from which it is difficult to recover cluster structure. Although the concept dissimilarity produced more evenly distributed distances compared with the Jaccard distance and tree edit distance, it is difficult to set three parameters properly. Compared with the results in Fig. 7b, the results produced by the PurTree distance are more explicable. Moreover, we have proven that the PurTree distance is a metric. Therefore, many transaction data clustering algorithms can benefit from this distance.

6.3 The Properties of the PurTreeClust Algorithm

We have defined a separate density to select the representative trees which are dense and far away from each other. The result on D_6 is shown in Fig. 8, in which the objects are sorted in descending order according to the separate density. From this result, we can see that most objects are in the same order as the order with the level density. We notice that some objects with relatively high level density have relatively small separate density. This is because that these objects have small separate distance, which means that they are close to denser objects. Therefore, separate density will assign the dense objects which are not far away from other dense objects with low value, thus avoid selecting them as candidate representative trees.

We selected 36 integers of k from 5 to 40 to run the PurTreeClust algorithm and computed the gap statistic values with Algorithm 5. The results are drawn in Fig. 9. We can see that both $E[\log(W(k))]$ and $\log(W(k))$ dropped slowly with the increase of k on all cover trees. With small γ,



(a) Clustering results with $\gamma = 0$. (b) Clustering results with $\gamma = 0.2$. (c) Clustering results with $\gamma = 0.5$.

(d) Clustering results with $\gamma = 1$. (e) Clustering results with $\gamma = 10$. (f) Clustering results with $\gamma = +\infty$.

Fig. 9. $E[\log(W(k))]$ and $\log(W(k))$ of clustering results by PurTreeClust on D_6.

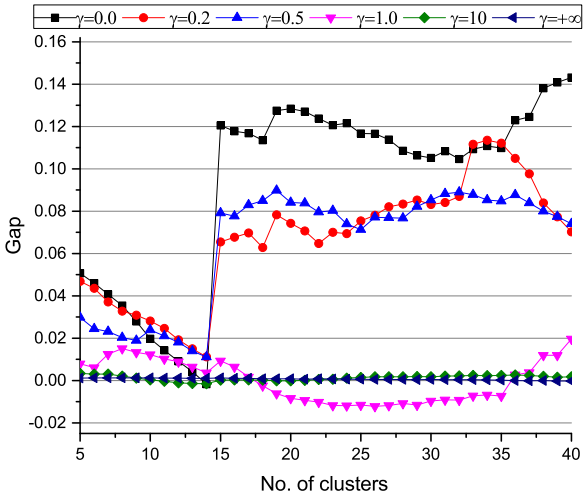
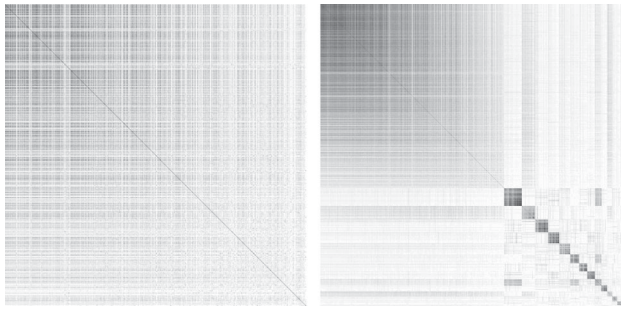


Fig. 10. Gap results of six clustering results by PurTreeClust on D_6.

$\log(W(k))$ decreased faster than $E[\log(W(k))]$ with small k and then the difference between them became stable. With big γ, $\log(W(k))$ got close to $E[\log(W(k))]$ for almost all k. The gap results are drawn in Fig. 10. We can see that the gap values were close to 0 when $\gamma = \{1, 10, +\infty\}$, which indicates that no cluster structure can be found with these three parameters. Recall the results in Fig. 7a, the distance between two purchase trees was too big when $\gamma = \{1, 10, +\infty\}$. Therefore, the customers were too far away from each other to uncover compact cluster structure. Note that $Gap(k)$ has a sudden increase when $k = 15$ for $\gamma = \{0, 0.2, 0.5\}$, we selected $\gamma = 0.2$ and $k = 15$ for the following results because the differences in all levels were considered when $\gamma = 0.2$.

To visually investigate the effectiveness of the clustering result by the PurTreeClust algorithm, we first computed the paired distances between trees in D_6 and drew the distance matrix in Fig. 11a, in which each row and column represent a purchase tree in the same order. The darker color represents lower distance and lighter color represents higher distance. From this figure, we cannot see clear cluster structure. To check whether clear cluster structure can be found from the clustering result by the PurTreeClust algorithm, we arranged the order of purchase trees in D_6 such that the trees in the same cluster were together, according to the clustering result with $\gamma = 0.2$ and $k = 15$. The result was drawn in Fig. 11b. From this figure, we can see 15 clear clusters.



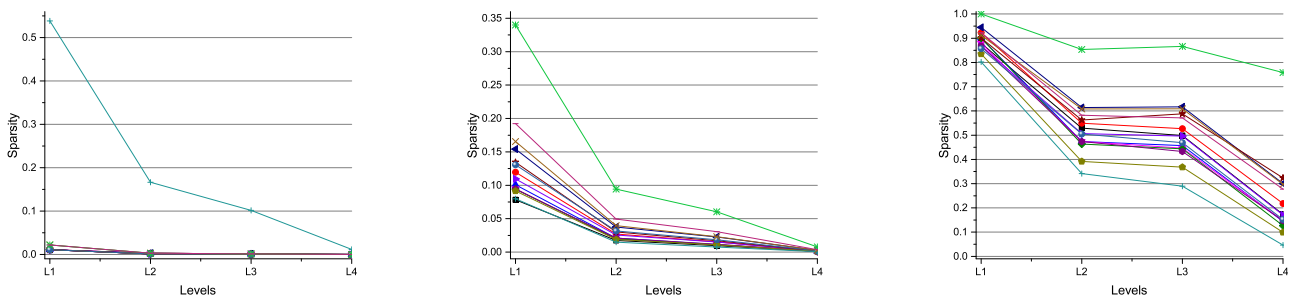
(a) Original PurTree distance matrix with $\gamma = 0.2$ on D_6.

(b) Clustering results by the PurTreeClust algorithm with $\gamma = 0.2$ and $k = 15$ on D_6.

Fig. 11. Distance matrix on D_6. In the two figures, the darker color represents lower distance and lighter color represents higher distance.

To further investigate the clustering result by the PurTreeClust algorithm, we computed the sparsity of 15 cluster centers from the clustering result by the PurTreeClust algorithm with $\gamma = 0.2$ and $k = 15$. The results are drawn in Fig. 12a. From this figure, we can see that the sparsity of a cluster are significantly higher than those of other clusters. Besides the big cluster, other clusters were small clusters with low sparsity. We also computed the average sparsity of the purchase trees in 15 clusters and drew the results in Fig. 12b. Compared with the results in Fig. 12a, we can see that the average sparsity of the big cluster is smaller than the sparsity of its cluster center, but the average sparsity of other clusters were close to or even bigger than the sparsity of corresponding cluster centers. Finally, we merged all purchase trees in a cluster into a purchase tree, called "cluster tree", and drew the average sparsity of the 15 cluster trees in Fig. 12c. We can see that the sparsity of the big cluster is still higher than those of other clusters. Interestingly, though the results of other 14 clusters were similar in Figs. 12a and 12b, their results were very different in Fig. 12c. The above results revealed the diversity of the uncovered cluster structures, thus verified the effectiveness of the PurTreeClust algorithm.

We reported the time costs of the PurTreeClust algorithm in the last experiment and the results are drawn in Fig. 13, in which the time costs with $k = 0$ are the time of building corresponding cover tree and the time costs with $k > 0$ are the time of building a cover tree plus the time of clustering. We can see that the time cost of building a cover tree increased with the increase of γ when $\gamma < 1$, and then decreased with the increase of $\gamma > 1$. This can be explained



(a) Sparsity of 14 cluster centers.

(b) Average sparsity of trees in each of 14 clusters.

(c) Sparsity of 14 cluster trees.

Fig. 12. Sparsity of clustering results on D_6.

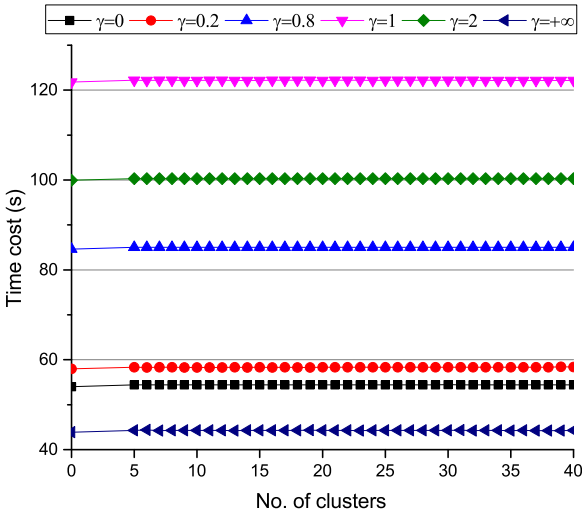
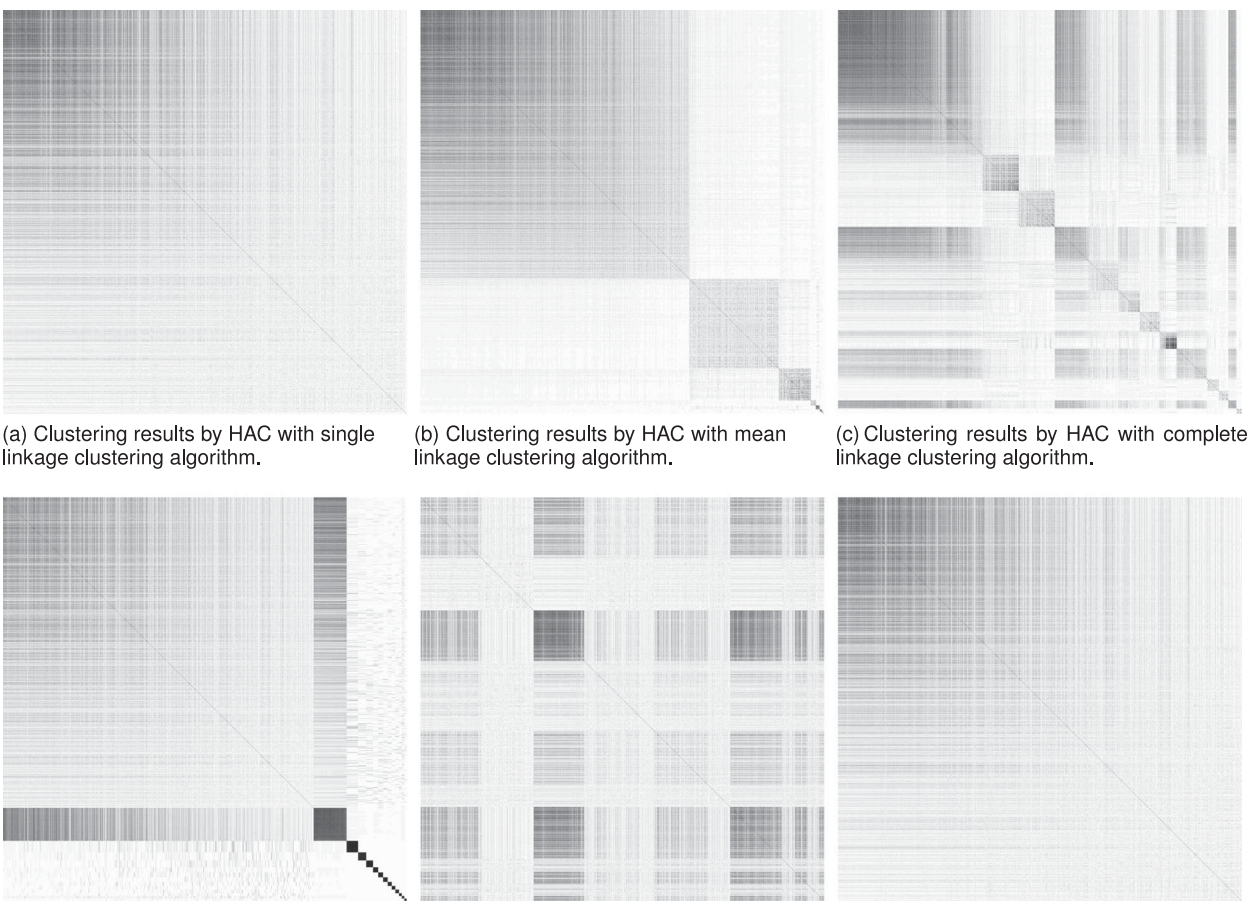


Fig. 13. The execution time of the PurTreeClust algorithm on D_6.

from the covering invariant of cover tree, since the distance between a parent node in the lth and its child node must be less than 2^l. We know from the results in Fig. 7 that as γ increased, the distance between two purchase trees became bigger, and then more objects will locate at lower levels and more distance comparison are involved during building a cover tree. If γ is too big, e.g., $\gamma = +\infty$, the distance between two purchase trees became close to 1 and almost all objects will be located in the level 0 without creating new levels and further distance comparison. We also notice that the time cost of the PurTreeClust algorithm kept nearly stable with the increase of the number of clusters. This indicates that we can very quickly cluster the data into different numbers of clusters, once a cover tree is built.

6.4 Comparison Results and Analysis

In this experiment, we compared the effectiveness of the PurTreeClust (PTC) algorithm with six clustering methods on the D_6, including DBSCAN (DAN) [20], two spectral clustering methods ratio cut RCut [25] and normalized cut NCut [26] and Hierarchical agglomerative clustering (HAC) with three linkages, i.e., HAC-SINGLE (HAC-S), HAC-MEAN (HAC-M) and HAC-COMPLETE (HAC-C). The PurTree distance was used for all seven clustering algorithms, in which the parameter γ was set as 0.2. The number of clusters k was set as 15. The clustering results by the six clustering algorithms excluding PurTreeClust were drawn in Fig. 14, which are organized in a similar way as Fig. 11. From these figures, we can see that HAC with single linkage and RCut uncovered no clusters. HAC with mean linkage and DBSCAN uncovered a very big cluster and many very small clusters. Although HAC with complete linkage and NCut uncovered relatively balanced clusters, but objects in different clusters have small distances. Compared with the results by the PurTreeClust algorithm in Fig. 11, we can see that PurTreeClust uncovered more balanced and clear clusters.



(a) Clustering results by HAC with single linkage clustering algorithm.

(b) Clustering results by HAC with mean linkage clustering algorithm.

(c) Clustering results by HAC with complete linkage clustering algorithm.

(d) Clustering results by DBSCAN.

(e) Clustering results by NCut.

(f) Clustering results by RCut.

Fig. 14. Clustering results by six clustering algorithms with the PurTree distance on D_6 ($\gamma = 0.2$ and $k = 15$).

TABLE 2
Comparisons of the Average $log(W_k)$ by Seven Clustering Algorithms on Ten Data Sets, in Which the Best Results are Highlighted in Bold

Data	DAN	HAC-S	HAC-M	HAC-C	NCut	RCut	PTC
D_1	4.44	3.93	3.83	3.82	4.70	4.72	**3.76**
D_2	4.55	4.39	4.31	4.27	4.43	4.55	**4.23**
D_3	5.23	5.16	5.06	5.04	5.14	5.23	**5.02**
D_4	5.93	5.90	5.79	**5.75**	5.86	5.91	5.76
D_5	6.62	6.61	6.50	6.45	6.56	6.60	**6.43**
D_6	7.32	7.31	7.19	**7.15**	7.26	7.32	7.21
D_7	5.47	5.44	5.42	5.40	5.42	5.47	**5.38**
D_8	6.17	6.17	6.14	6.12	6.10	6.12	**6.10**
D_9	6.86	6.85	6.85	6.84	6.86	6.88	**6.82**
D_{10}	7.63	7.65	7.68	7.62	7.66	7.70	**7.61**

We used all ten data sets to compare the effectiveness of the PurTreeClust algorithm with those of six clustering algorithms used in the previous experiment. In this experiment, the PurTree distance was used for all seven clustering algorithms, in which the parameter γ was set as $\{0, 0.2, 0.5, 1, 10, +\infty\}$. The same 36 integers from 5 to 40 were selected for k to run the six clustering algorithms excluding DBSCAN. For each clustering result, $\log(W(k))$ was computed for evaluation. The results are shown in Table 2. From these results, we can see that PurTreeClust produced the highest $log(W_k)$ results on six data sets. Besides PurTreeClust, HAC-COMPLETE produced better clustering results than other methods.

6.5 Scalability

We used all ten data sets in Table 1 to compare the scalability of the PurTreeClust algorithm against SPEC, HAC and DBSCAN. In this experiment, we selected a set of γ and k to run the three algorithms and computed the average time costs by three clustering algorithms. The results are reported in Table 3, in which SPEC, HAC and DBSCAN cannot finish clustering in accepted time on the last data set. In summary, the time costs by the PurTreeClust algorithm were much smaller than those by other three algorithms. This indicates that PurTreeClust scales better to large transaction data than the other six clustering algorithms.

To test the time cost of the PurTreeClust with the increase of the number of products, we selected D_6 and sampled 10 data sets with different number of products. The sample ratios of the 10 data sets were set as $\{0.1, 0.2, \cdots, 1\}$. The time costs of seven clustering algorithms on the 10 data sets were are reported in Table 4. From these results, we can see that the PurTreeClust algorithm has the smallest time cost. With the increase of the products, the times costs of all clustering algorithms were nearly not changed, which indicates that the PurTree metric distance scales well to the number of products.

TABLE 3
Time Costsc (s) versus the Number of Objects

#Samples	DAN	HAC-S	HAC-M	HAC-C	NCut	RCut	PTC
208	0.4	0.3	0.3	0.4	1.0	0.7	0.4
416	2	2	2	2	4	4	2
832	7	7	8	8	19	19	6
1,664	30	32	34	35	118	127	27
3,328	132	147	158	161	881	968	115
6,656	540	650	672	693	6,375	7,202	493
13,312	2,350	2,543	2,790	2,965	-	-	1,819

TABLE 4
Time Costs (s) Versus the Sample Ratio

Ratio	DAN	HAC-S	HAC-M	HAC-C	NCut	RCut	PTC
10%	99	130	102	101	216	214	91
20%	102	134	106	104	222	221	90
30%	102	132	105	103	222	220	94
40%	101	132	104	103	221	219	91
50%	100	133	104	102	219	217	90
60%	101	132	104	103	221	219	94
70%	99	129	102	100	215	214	91
80%	98	128	101	100	215	213	90
90%	98	127	101	100	212	211	90
100%	94	123	98	97	206	205	86

7 Conclusions

We have presented PurTreeClust for massive customer transaction data. In this method, "purchase tree" is built for each customer from the customer transaction data. A new distance metric is defined to effectively compute the distance from two purchase trees. The experimental results in Section 6.2 show that we can choose proper γ according to the distribution of pairwise distances between pairs of objects. To cluster these purchase trees, we first rank the purchase trees as candidate representative trees with a novel separate density, and then select the top k customers as the representatives of k customer groups. Finally, the clustering results can be obtained by assigning each customer to the nearest representative tree. We also propose a gap statistic based method to evaluate the number of clusters. A series of experiments were conducted, and the experimental results have shown that the new distance metric is effective and the PurTreeClust algorithm is more effective and scalable than six clustering algorithms.

However, in PurTreeClust, the quantities and amount spent are not considered. In the future, we will extend our method to incorporate more features into the purchase tree, such as monetary expense, amount, etc. Furthermore, we will study clustering of mixture data, e.g., purchase tree and general variables.

Acknowledgments

This research was supported by NSFC under Grants 61305059, 61773268 and 61473194.

References

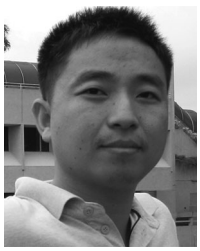
[1] Y. Yang, X. Guan, and J. You, "CLOPE: A fast and effective clustering algorithm for transactional data," in *Proc. 8th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2002, pp. 682–687.
[2] R. G. Drozdenko and P. D. Drake, *Optimal Database Marketing: Strategy, Development, and Data Mining*. Newbury Park, CA, USA: Sage, 2002.
[3] C.-Y. Tsai and C.-C. Chiu, "A purchase-based market segmentation methodology," *Expert Syst. Appl.*, vol. 27, no. 2, pp. 265–276, 2004.
[4] T.-C. Lu and K.-Y. Wu, "A transaction pattern analysis system based on neural network," *Expert Syst. Appl.*, vol. 36, no. 3, pp. 6091–6099, 2009.
[5] V. L. Miguéis, A. S. Camanho, and J. F. E. Cunha, "Customer data mining for lifestyle segmentation," *Expert Syst. Appl.*, vol. 39, no. 10, pp. 9359–9366, 2012.

[6] F.-M. Hsu, L.-P. Lu, and C.-M. Lin, "Segmenting customers by transaction data with concept hierarchy," *Expert Syst. Appl.*, vol. 39, no. 6, pp. 6221–6228, 2012.
[7] M. Pawlik and N. Augsten, "RTED: A robust algorithm for the tree edit distance," *Proc. VLDB Endowment*, vol. 5, no. 4, pp. 334–345, 2011.
[8] D. R. Karger and M. Ruhl, "Finding nearest neighbors in growth-restricted metrics," in *Proc. 34th Annu. ACM Symp. Theory Comput.*, 2002, pp. 741–750.
[9] R. Kuo, L. Ho, and C. M. Hu, "Integration of self-organizing feature map and k-means algorithm for market segmentation," *Comput. Operations Res.*, vol. 29, no. 11, pp. 1475–1493, 2002.
[10] T. Xiong, S. Wang, A. Mayers, and E. Monga, "DHCC: Divisive hierarchical clustering of categorical data," *Data Mining Knowl. Discovery*, vol. 24, no. 1, pp. 103–135, 2012.
[11] F. Giannotti, C. Gozzi, and G. Manco, "Clustering transactional data," in *Principles of Data Mining and Knowledge Discovery*. Berlin, Germany: Springer-Verlag, 2002, pp. 175–187.
[12] R. Xu and D. Wunsch, "Survey of clustering algorithms," *IEEE Trans. Neural Netw.*, vol. 16, no. 3, pp. 645–678, May 2005.
[13] X. Chen, Y. Ye, X. Xu, and J. Z. Huang, "A feature group weighting method for subspace clustering of high-dimensional data," *Pattern Recog.*, vol. 45, no. 1, pp. 434–446, 2012.
[14] X. Chen, J. Z. Huang, Q. Wu, and M. Yang, "Subspace weighting co-clustering of gene expression data," *IEEE/ACM Trans. Comput. Biol. Bioinf.*, to be published. doi: 10.1109/TCBB.2017.2705686.
[15] X. Chen, X. Xu, Y. Ye, and J. Z. Huang, "TW-k-means: Automated two-level variable weighting clustering algorithm for multi-view data," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 4, pp. 932–944, Apr. 2013.
[16] H.-P. Kriegel, P. Kröger, J. Sander, and A. Zimek, "Density-based clustering," *Wiley Interdisciplinary Rev.: Data Mining Knowl. Discovery*, vol. 1, no. 3, pp. 231–240, 2011.
[17] U. Von Luxburg , "A tutorial on spectral clustering," *Statist. comput.*, vol. 17, no. 4, pp. 395–416, 2007.
[18] T. Zhang, R. Ramakrishnan, and M. Livny, "Birch: An efficient data clustering method for very large databases," in *Proc. ACM SIGMOD Int. Conf. Manag. Data Rec.*, 1996, vol. 25, no. 2, pp. 103–114.
[19] X. Chen, F. Nie, J. Z. Huang, and M. Yang, "Scalable normalized cut with improved spectral rotation," in *Proc. 26th Int. Joint Conf. Artif. Intell.*, 2017, pp. 1518–1524.
[20] J. Sander, M. Ester, H.-P. Kriegel, and X. Xu, "Density-based clustering in spatial databases: The algorithm GDBSCAN and its applications," *Data Mining Knowl. Discovery*, vol. 2, no. 2, pp. 169–194, 1998.
[21] A. L. A. Rodriguez, "Clustering by fast search and find of density peaks," *Sci.*, vol. 344, no. 6191, pp. 1492–1496, Jun. 2014.
[22] A. Beygelzimer, S. Kakade, and J. Langford, "Cover trees for nearest neighbor," in *Proc. 23rd Int. Conf. Mach. Learn.*, 2006, pp. 97–104.
[23] A. H. Lipkus, "A proof of the triangle inequality for the tanimoto distance," *J. Math. Chemistry*, vol. 26, no. 1–3, pp. 263–265, 1999.
[24] R. Tibshirani, G. Walther, and T. Hastie, "Estimating the number of clusters in a data set via the gap statistic," *J. Roy. Statist. Soc.: Series B Statist. Methodol.*, vol. 63, no. 2, pp. 411–423, 2001.
[25] L. Hagen and A. B. Kahng, "New spectral methods for ratio cut partitioning and clustering," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 11, no. 9, pp. 1074–1085, Sep. 1992.
[26] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 8, pp. 888–905, Aug. 2000.
[27] X. Chen, M. Yang, J. Z. Huang, and Z. Ming, "TWCC: Automated two-way subspace weighting partitional co-clustering," *Pattern Recogniti.*, 2017, doi: https://doi.org/10.1016/j.patcog.2017.10.026.

Xiaojun Chen received the PhD degree from the Harbin Institute of Technology, in 2011. He is now an assistant professor in the College of Computer Science and Software, Shenzhen University. His research interests include clustering, feature selection, and massive data mining. He is a member of the IEEE.

Yixiang Fang received the BEng and the MS degrees from the Harbin Engineering University, and ShenZhen Graduate School, Harbin Institute of Technology, in 2010 and 2013, respectively. He is currently working toward the PhD degree in the Department of Computer Science, University of Hong Kong (HKU) under the supervision of Dr. Reynold Cheng. His research interests mainly focus on big data analytics on spatial-temporal databases, graph databases, uncertain databases, and web data mining.

Min Yang received the PhD degree from the University of Hong Kong, in February 2017. She is currently an assistant professor in the Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences. Her current research interests include machine learning and natural language processing.



Feiping Nie received the PhD degree in computer science from Tsinghua University, China in 2009. His research interests include the machine learning and its applications, such as pattern recognition, data mining, computer vision, image processing and information retrieval. He has published more than 100 papers in the following top journals and conferences: the *IEEE Transactions on Pattern Analysis and Machine Intelligence*, the *International Journal of Computer Vision*, the *IEEE Transactions on Image Processing*, the *IEEE Transactions on Neural Networks and Learning Systems*/*IEEE Transactions on Neural Networks*, the *IEEE Transactions on Knowledge and Data Engineering*, the *ACM Transactions on Knowledge Discovery from Data*, *Bioinformatics*, ICML, NIPS, KDD, IJCAI, AAAI, ICCV, CVPR ACM MM. He is now serving as associate editor or PC member for several prestigious journals and conferences in the related fields.

Zhou Zhao received the BS and the PhD degrees in computer science from the Hong Kong University of Science and Technology (HKUST), in 2010 and 2015, respectively. He is currently an associate professor with the College of Computer Science, Zhejiang University. His research interests include machine learning, data mining, and information retrieval.

Joshua Zhexue Huang received the PhD degree from the Royal Institute of Technology, Sweden. He is now a professor in the College of Computer Science and Software, Shenzhen University, and professor and chief scientist in the Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, and honorary professor in the Department of Mathematics, The University of Hong Kong. His research interests include data mining, machine learning, and clustering algorithms.