SEC: More Accurate Clustering Algorithm via Structural Entropy

Junyu Huang[1, 2], **Qilong Feng**[1, 2, *], **Jiahui Wang**[1, 2], **Ziyun Huang**[3], **Jinhui Xu**[4], **Jianxin Wang**[1, 2, 5, *]

[1] School of Computer Science and Engineering, Central South University, Changsha 410083, China
[2] Xiangjiang Laboratory, Changsha 410205, China
[3] Department of Computer Science and Software Engineering, Penn State Erie, The Behrend College
[4] Department of Computer Science and Engineering, State University of New York at Buffalo, NY, USA
[5] The Hunan Provincial Key Lab of Bioinformatics, Central South University, Changsha 410083, China
junyuhuangcsu@foxmail.com, csufeng@mail.csu.edu.cn, 224711099@csu.edu.cn, zxh201@psu.edu, jinhui@buffalo.edu, jxwang@mail.csu.edu.cn

Abstract

As one of the most popular machine learning tools in the field of unsupervised learning, clustering has been widely used in various practical applications. While numerous methods have been proposed for clustering, a commonly encountered issue is that the existing clustering methods rely heavily on local neighborhood information during the optimization process, which leads to suboptimal performance on real-world datasets. Besides, most existing clustering methods use Euclidean distances or densities to measure the similarity between data points. This could constrain the effectiveness of the algorithms for handling datasets with irregular patterns. Thus, a key challenge is how to effectively capture the global structural information in clustering instances to improve the clustering quality. In this paper, we propose a new clustering algorithm, called SEC. This algorithm uses the global structural information extracted from an encoding tree to guide the clustering optimization process. Based on the relation between data points in the instance, a sparse graph of the clustering instance can be constructed. By leveraging the sparse graph constructed, we propose an iterative encoding tree method, where hierarchical abstractions of the encoding tree are iteratively extracted as new clustering features to obtain better clustering results. To avoid the influence of easily misclustered data points located on the boundaries of the clustering partitions, which we call "fringe points", we propose an iterative pre-deletion and reassignment technique such that the algorithm can delete and reassign the "fringe points" to obtain more resilient and precise clustering results. Empirical experiments on both synthetic and real-world datasets demonstrate that our proposed algorithm outperforms state-of-the-art clustering methods and achieves better clustering performances. On average, the clustering accuracy (ACC) is increased by 1.7% and the normalized mutual information (NMI) by 7.9% compared with the current state-of-the-art (SOTA) algorithm on synthetic datasets. On real-world datasets, our method outperforms other clustering methods with an average increase of 12.3% in ACC and 5.2% in NMI, respectively.

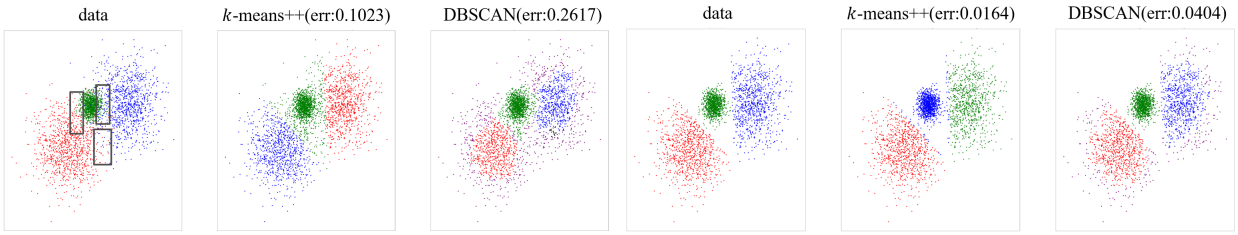
*Corresponding Authors

Introduction

As a widely studied unsupervised learning task, clustering involves partitioning data points into different clusters according to their similarity such that data points within the same cluster share high similarity as much as possible. By modeling clustering tasks as optimization problems, several classic clustering methods have been proposed, such as Lloyd-type methods, DBSCAN, hierarchical clustering and MST clustering. However, a recurring limitation of the existing methods is that they only use local neighborhood information during the optimization process. Furthermore, most existing clustering methods use Euclidean distances or clustering densities to measure the similarity between data points. These clustering methods might fail to capture the intrinsic global patterns of the given clustering instances, which could constrain the effectiveness of the algorithms for handling datasets with irregular patterns. Consequently, a critical challenge for achieving better clustering performance lies in effective integration of global structural information into the clustering optimization process.

As pointed out in (Du and Wang 2022), global structural information of the data points represents the high-level patterns and relationships across the entire dataset. Recently, a new metric based on the graph's structural information, known as structural entropy, was introduced to analyze the hierarchical structure of graphs. This metric utilizes an encoding tree approach, offering a novel perspective in graph analysis (Li and Pan 2016). By minimizing the structural entropy of the given graph, an encoding tree can be constructed where each node of the encoding tree is associated with a partition of the graph. In general, the encoding tree provides hierarchical abstractions of the graph, allowing the preservation of global structural information within its nodes.

For many datasets, the clustering performance is closely related to the distinctiveness between clusters. To enlarge the distinctiveness between clusters, we present an encoding-tree-based structural metric where global structural information of the given data points can be captured to guide the clustering process. Firstly, we design a new sparse graph embedding technique for clustering instances using k-Nearest Neighbor (k-NN) and thresholding strategies. k-NN and thresholding strategies help to identify the most rele-



(a) Comparisons of clustering performances on k-means++ and DBSCAN with the "fringe set".

(b) Comparisons of clustering performances on k-means++ and DBSCAN without the "fringe set".

Figure 1: Comparisons of clustering performances on k-means++ and DBSCAN with and without the "fringe set", where the black box represents the locations for the "fringe set". The error rate for clustering is given for k-means++ and DBSCAN in the figure.

vant neighbors while removing irrelevant neighbors for each node in the graph, which can provide potentially better local and global learnable structures for encoding tree. Secondly, in order to obtain complex and non-linear structures, we propose an iterative encoding tree method, where hierarchical abstractions of the encoding tree are iteratively extracted as new clustering features to better represent the intrinsic hidden relationships of the given datasets.

For the clustering problem, empirical observations suggest that most of the misclustered data points locate on the boundaries of the clustering partitions, which we call the "fringe set". Figure 1 gives illustrations to show the impact of the fringe set for clustering problem. It can be seen that the fringe set presents significant assignment challenges for classic clustering methods such as k-means++ and DBSCAN. The existence of the fringe set can significantly deteriorate the performance of classic clustering methods. In order to enhance the capability of the proposed algorithm to better identify and assign data points in the fringe set, we propose an iterative pre-deletion and reassignment method that can identify, delete, and reassign data points in the fringe set during the clustering optimization process. By using this method, interference caused by the fringe set can be alleviated to obtain a more resilient and precise clustering partition. The main contributions of this paper are summarized as follows:

- We propose a new sparse graph embedding method for clustering instances. The proposed graph embedding method can provide a better representation of the relationships between the given data points.
- Based on the sparse graph obtained, we propose an iterative encoding tree method to extract global structural information from the given clustering instances. The proposed feature extraction method provides iterative hierarchical abstractions of the encoding tree structure, enhancing the algorithm's capability to capture the hidden relationships within the entire dataset.
- We propose an iterative pre-deletion and reassignment technique such that the fringe set can be identified, deleted and reassigned during the clustering optimization process. With this technique, the interference of the fringe set can be alleviated to obtain better clustering results.
- Empirical experiments demonstrate that the proposed SEC algorithm outperforms other state-of-the-art clustering methods in terms of clustering quality. The average clustering performance shows an increase of 12.3% in ACC and 5.2% in NMI on real-world datasets. In particular, for a specific subset of these real-world datasets, the clustering ACC and NMI are increased by 20.6% and 9.8%, respectively.

Related Work

Lloyd-Type Clustering Methods

The main idea behind Lloyd-type clustering methods is to iteratively assign data points to the nearest clustering centroids and update the centroids to minimize the sum of the squared distances. However, as pointed out in (Blömer et al. 2016), Lloyd-type methods (Lloyd 1982) are highly sensitive to the initialization. Arthur and Vassilvitskii (Arthur and Vassilvitskii 2007) proposed the k-means++ seeding method, which uses D^2-sampling strategy to achieve an $O(\log k)$-approximation guarantee on clustering quality. Lattanzi and Sohler (Lattanzi and Sohler 2019) proposed a combination of local search and k-means++ method, which can yield a constant approximation in linear running time. A distinct line of work for obtaining better initialization is to integrate heuristic strategies into Lloyd-type methods (Li and Wu 2012; Mawati, Sumertajaya, and Afendi 2014; Zhou et al. 2017; Nainggolan et al. 2019; Yang et al. 2021; Huang et al. 2021). However, they still rely heavily on the local neighborhood information for clustering optimization.

MST Clustering Methods

MST clustering aims to create a minimum spanning tree (MST) through iterative connections of data points with minimum edge weights. Clustering partitions can be obtained by cutting inconsistent edges with larger weights. For MST clustering methods, Zahn (Zahn 1971) defined inconsistent edges as those edges with weights significantly larger than the average weight of neighboring edges. Chowdhury and Murthy (Chowdhury and Murthy 1997) proposed a new measurement of inconsistency based on the identification of

the boundaries separating distinct clusters. By integrating local density information, several density-based MST clustering methods were proposed, such as the LDP-MST method (Cheng et al. 2019), which avoids the influence of noisy points and reduces the running time.

Density-Based Clustering Methods

Density-Based clustering methods (DB methods) form clustering partitions according to the density of the data points. DBSCAN (Ester et al. 1996) is one of the most widely used methods that quantifies the density of data points using local neighborhood information. However, as pointed out in (Bhattacharjee and Mitra 2021), DBSCAN is highly sensitive to the search radius of local neighborhoods. In (Ram et al. 2010) and (Campello, Moulavi, and Sander 2013), density fluctuations within the same cluster were considered during the DBSCAN process to improve the algorithm's robustness. In (Ankerst et al. 1999) and (Hou, Gao, and Li 2016), parameter-adaptive and non-parametric versions of DBSCAN methods were proposed.

Hierarchical Clustering Methods

Hierarchical clustering mainly falls into two categories: divisive and agglomerative methods. The divisive approach starts with a single cluster that encompasses all data points and iteratively splits this cluster into smaller clusters until each cluster contains a single data point. The agglomerative approach iteratively merges the most similar clusters until a single cluster containing all the data points is formed. Steinbach, Karypis, and Kumar (Steinbach, Karypis, and Kumar 2000) used the k-means method to iteratively form smaller clusters. Gracia and Binefa (Gracia and Binefa 2011) modified the definition of the k-means objective function to achieve better performance. (Sneath 1973) used the Euclidean distance to define the similarity when performing merging operations. (Yang et al. 2023) used local density to improve the merging process for hierarchical clustering methods.

Preliminaries

Given an integer $n \in \mathcal{Z}^+$, denote $[n]$ as the set $\{1, 2, \ldots, n\}$. Given a set $X = \{x_i | i \in [n]\}$ of data points and a labeling partition $C = \{c_j | j \in [k]\}$, for any data point x_i, let $l_i = (l_{i1}, \ldots, l_{ik})$ be the binary vector denoting which cluster x_i is assigned to, where $l_{ij} = 1$ if x_i is in the j-th cluster, otherwise $l_{ij} = 0$. Given an undirected weighted graph $G = (V, E, W)$, where V is the vertex set, E is the edge set, and $W : E \to \mathcal{R}^+$ is the weight function of edges, let $VOL = \sum_{e \in E} W(e)$ be the sum of the weights of edges. For each $v \in V$, we denote the sum of the weights of its connected edges as $vol(v)$. For a point c_i with $l_i = \{l_{i1}, \ldots, l_{ik}\}$, the center c_j for the j-th cluster is defined as $c_j = \sum_{i \in [n]} l_{ij} x_i / \sum_{i \in [n]} l_{ij}$.

Clustering with Global Structural Feature

For clustering tasks, existing clustering methods commonly rely on the pairwise distances and densities to group the data points into clusters. Thus, a fundamental challenge is how to effectively incorporate global structural features into the clustering optimization process.

In this section, we present a new algorithm called the SEC algorithm (Algorithm 1), which integrates structural entropy to capture the global structural features of the clustering instances. The SEC algorithm mainly consists of three parts: (1) sparse graph embedding (lines 2-3 of Algorithm 1); (2) structural entropy extraction (lines 4-5 of Algorithm 1); (3) iterative pre-deletion and reassignment (line 7 of Algorithm 1). Figure 2 provides a graph illustration to show how the algorithm works.

The proposed sparse graph embedding method adapts the k-NN and thresholding strategies for graph sparsification. In the structural entropy extraction phase, an encoding tree is first constructed based on the obtained graph embedding. Each vertex of the encoding tree represents a graph partition determined by the structural entropy of the graph nodes. Then, we propose an iterative encoding tree method to obtain global structural features by iteratively augmenting the dimension of the raw dataset and adjusting the tree structure.

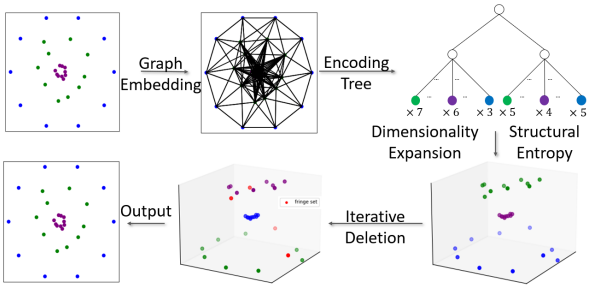


Figure 2: A graph illustration on how Algorithm 1 works

To avoid the influence of easily misclustered data points, we propose a new definition called the fringe set to find data points that are located in the regions consisting of multiple distinct clusters and share similar structural entropy. Data points in the fringe set are inherently difficult to separate. To overcome this challenge, in the iterative pre-deletion and reassignment phase, we propose a fringe set exclusion method, which can identify, delete, and reassign the fringe points during the clustering optimization process.

Graph Embedding

For the clustering problem, a natural way of graph embedding is to represent the given dataset as a complete graph, where the weight between any two point is their Euclidean distance. However, for the clustering problem, intra-cluster similarity should be much smaller than the inter-cluster similarity. Hence, a complete graph may mislead the relationship between data points and prevent the algorithm from obtaining good structural information. Thus, we present a new graph embedding method, as described in Algorithm 2, to construct a sparse graph for the clustering problem.

The formal graph embedding process is given in Algorithm 2. The algorithm iteratively finds the k-nearest neighbors (denoted as k-NN) of each data point (steps 2 to 4) to form the edges of the graph. However, the k-NN method

Algorithm 1: SEC

Input: Dataset X, a set $D = \{d_1, \ldots, d_t\}$ of tree depths, the number of clusters k, a set $N = \{nn_1, \ldots, nn_t\}$ of neighbor search parameters, a set $\epsilon = \{\epsilon_1, \ldots, \epsilon_t\}$ of thresholds, and parameters δ and t.
Output: Labels of data points.

for all $i = 1, 2, \cdots, t$ **do**
  $G(V, E, W) \leftarrow \text{BuildGraph}(X, nn_i, \epsilon_i)$;
  $VOL \leftarrow \sum_{e \in E} W(e)$;
  $ent \leftarrow \text{EncodingTree}(G, d_i)$;
  $X \leftarrow \text{concat}(X, ent)$;
end for
$l \leftarrow \text{IterativeDeletion}(X, k, \delta)$;
return l;

Algorithm 2: BuildGraph

Input: Dataset X, a number of edges N and a threshold ϵ.
Output: A sparse graph G.

Initialize G as an empty graph;
for all $i = 1, 2, \cdots, n$ **do**
  for all $j = 1, 2, \cdots, N$ **do**
    Let x be the j-furthest point from x_i;
    If $d(x, x_i) > \epsilon$, then break;
    Add edge(x, x_i) with weight $d(x, x_i)$;
  end for
end for
return $G(V, E, W)$;

may overlook the non-uniform patterns within the graph due to its strict requirement that each node must be connected to precisely k neighbors. Therefore, in step 5 of Algorithm 2, a thresholding step is used to remove irrelevant neighbors for each node, where data points with distances larger than a certain threshold should not be assigned to the same cluster since they are significantly distant from each other.

Encoding Tree Construction

The encoding tree is a hierarchical tree structure that can reflect the hierarchical abstractions of the intrinsic and nonlinear patterns among the data points. Based on this structure, features with global structural information can be extracted to help the clustering optimization process.

A standard encoding tree is a binary tree constructed from leaf nodes to the root node (Li and Pan 2016). Each node of the encoding tree is a partition of the graph. Given a graph G with vertex set V and edge set E, an encoding tree T of G should satisfy the following properties: 1) for each node $v_\tau \in T$, it contains a vertex subset $T_{v_\tau} \subseteq V$; 2) for the root node v_r, it contains the whole vertex set V; 3) for each node $v_\tau \in T$, it has a parent node v_τ^+, and a child node set V_τ^-; 4) for any two nodes n_a and n_b such that n_a and n_b share the same parent node v_τ, it should be guaranteed that $T_{n_a} \cap T_{n_b} = \emptyset$; 5) for each leaf node v, T_v is a singleton subset containing a single graph vertex, where the total number of leaf nodes is exactly $|V|$.

The one-dimensional structural entropy of a graph G (denoted as $H^1(G)$) is defined as $H^1(G) = -\sum_{v \in V} \frac{vol(v)}{VOL} \cdot \log \frac{vol(v)}{VOL}$. For each node $v_\tau \in T$ with $v_\tau \neq v_r$, the node entropy is defined as $H^T(G; v_\tau) = -\frac{g_{v_\tau}}{VOL} \log_2 \frac{vol(v_\tau)}{vol(v_\tau^+)}$, where g_{v_τ} is the sum of the weights of all the edges between nodes in T_{v_τ} and outside T_{v_τ}, and $vol(v_\tau)$ is $\sum_{v \in T_{v_\tau}} vol(v)$. The D-dimensional structural entropy of G can be defined as $H^D(G) = \min_T \sum_{v_\tau \in T, v_\tau \neq v_r} H^T(G; v_\tau)$, where T represents the set of encoding trees with heights at most D.

The encoding tree is constructed by iteratively merging the leaf nodes to minimize the structural entropy until a binary tree is formed. However, directly constructing an encoding tree may not yield appropriate clustering partitions. This may arise from the significant imbalance between the number of nodes in the encoding tree and the number of predefined clusters for partitioning. To achieve improved clustering partitions, we aim to limit the height of the encoding tree to a pre-defined tree height D, where a tree-height restriction approach is proposed. The tree-height restriction approach mainly consists of three operations: the COMBINE, the DROP, and the SINGUP. Intuitively, the COMBINE operation aims to minimize the structural entropy during tree construction, the DROP operation aims to reduce the tree height of the encoding tree to satisfy certain height constraint with minimum increase in structural entropy, and the SINGUP operation aims to adjust the encoding tree to the specified tree height without additional increase in structural entropy.

Definition 1 *Given two vertexes $v_c^1, v_c^2 \in V_r^-$, the operation* $\mathbf{COMBINE}(v_c^1, v_c^2)$ *is defined as creating a child node v_τ for v_r to serve as the parent node for v_c^1 and v_c^2, where $T_{v_\tau} = T_{v_c^1} + T_{v_c^2}$. The operation* $\mathbf{DROP}(v_\tau)$ *is defined as removing the node v_τ from T and connecting the children of v_τ to its parent, where $T_{v_\tau^+} = T_{v_\tau^+} \cup T_{v_\tau}$. Given a leaf node $v_i \in V$, the operation* $\mathbf{SINGUP}(v_i, D)$ *is defined as adding a node v_i' as the parent of v_i and the child of v_i^+ repeatedly until the depth of v_i is D, where $T_{v_i'} = T_{v_i}$.*

The specific encoding tree construction process is given in Algorithm 3. Based on the proposed tree operations, Algorithm 3 can be divided into three stages.

STAGE 1. In stage 1 (steps 2 to 3), the algorithm explores all possible COMBINE operations to determine the best one that yields the maximal reduction in structural entropy. Then, two children v_c^1 and v_c^2 are merged to form a new tree node. In Algorithm 3, COMBINE operations are repeated until $|V_r^-| < 3$. Finally, a binary encoding tree T can be obtained.

STAGE 2. In stage 2 (steps 5 to 6), all possible DROP operations are explored to reduce the height. We use $T_{\mathbf{DROP}(v_\tau)}$ to denote the encoding tree that is obtained after performing the operation $\mathbf{DROP}(v_\tau)$. Then, we repeat stage 2 until the height of the encoding tree is no larger than D. If there exists a node v not satisfying the height constraint, using $\mathbf{DROP}$ operations might reduce the height of the children of v. As a result, there might exist some leaf nodes with different heights. Thus, the remaining task in-

Algorithm 3: EncodingTree

Input: Graph $G = (V, E, W)$, tree depth $D > 1$.
Output: A vector of structural entropy for the data points.

Build an initial encoding tree T with a root node v_r, and each leaf node of v_r corresponds to a unique vertex in V as one of its children;
while $|V_r^-| > 2$ **do**
  $\text{COMBINE}(v_c^1, v_c^2) \leftarrow \arg\max_{(v_c^1, v_c^2)} \{\mathcal{H}^T(G) - \mathcal{H}^{T_{\text{COMBINE}(v_c^1, v_c^2)}}(G) \,|v_c^1, v_c^2 \in V_r^-\}$;
end while
while Height$(T) > D$ **do**
  $\text{DROP}(v_\tau) \leftarrow \arg\min_{v_\tau} \{\mathcal{H}^{T_{\text{DROP}(v_\tau)}}(G) - \mathcal{H}^T(G) \,|v_\tau \in T \& v_\tau \neq v_r \& v_\tau \notin V\}$;
end while
for all $v_i \in V$ **do**
  If Depth$(T, v_i) < D$, then Call $\mathbf{SINGUP}\,(v_i, D)$;
end for
for all $i \in \{1, 2, \ldots, n\}$ **do**
  Find a path $P = [v_i, \ldots, v_r]$ in encoding tree T from v_i to the root v_r;
  $\mathcal{H}^T(x_i) \leftarrow -\sum_{v_\tau \in P} \frac{g_{v_\tau}}{VOL} \log \frac{vol(v_\tau)}{vol(v_\tau^+)}$;
end for
return $[\mathcal{H}^T(x_1), \mathcal{H}^T(x_2), \ldots, \mathcal{H}^T(x_n)]^{\mathrm{T}}$;

volves adjusting the heights of the leaf nodes to ensure that all the leaf nodes share the same height.

STAGE 3. In this stage, we use the SINGUP operation to simultaneously modify all leaf nodes to restrict their height to the specified parameter D without increasing the structural entropy of the graph partitions. The time complexity for the SINGUP operation is $O(nD)$, where D is the specified height after adjustment, and n is the number of vertices in the graph.

To this end, we have introduced how to obtain structural entropy for both the graph G and the nodes in the encoding tree. For the clustering problem, we will next introduce the definition of structural entropy for a data point to achieve the hierarchical abstraction of the entire dataset.

Definition 2 *Given an encoding tree T, for each data point $x_i \in X$ corresponding a leaf node $v_i \in T$, there exists a unique path $P = [v_i, \ldots, v_r]$ from v_i to the root node v_r. The structural entropy of data point x_i is defined as the cumulative sum of the entropy associated with each node along the path P, where $\mathcal{H}^T(x_i) = -\sum_{v_\tau \in P} \frac{g_{v_\tau}}{VOL} \log \frac{vol(v_\tau)}{vol(v_\tau^+)}$.*

Based on the three tree operations defined and the structural entropy of data points, a modified encoding tree can be constructed using Algorithm 3, where a vector of global features is constructed for each data point in the dataset.

Iterative Pre-deletion and Reassignment

For the clustering problem, it is observed that most misclustered data points are located at the boundaries consisting of distinct clustering partitions, which we refer to as the fringe points. Fringe points can lead to significant deviation of the clustering boundaries. To avoid the impact of the fringe points and obtain better clustering partitions, we propose an iterative pre-deletion and reassignment method, which can iteratively delete and reassign the fringe points during the clustering process. The fringe points are defined using the projective distance, which is given in Definition 3.

Definition 3 *Given a point A and a vector $\overrightarrow{BC}$ from point B to point C, the projective distance Pd between point A and vector $\overrightarrow{BC}$ is defined as $Pd(A, \overrightarrow{BC}) = \overrightarrow{BC} \cdot \overrightarrow{BA}/d(B, C)$, where $\overrightarrow{BC} \cdot \overrightarrow{BA}$ is the dot product of the vectors and $d(B, C)$ is the Euclidean distance between points B and C.*

Based on projective distance, the fringe set is defined in Definition 4. In the clustering process, Algorithm 4 can identify and exclude the fringe set before each clustering center updating step. To avoid significant deviation from the ground truth clustering partitions, the algorithm finds the midpoints between the updated centers with and without the fringe set as the new centers. After updating the clustering centers, data points in the fringe set are randomly assigned to the obtained centers to achieve better clustering results.

Definition 4 *Consider a dataset $X \subseteq \mathcal{R}^d$ and a set $C \subseteq \mathcal{R}^d$ of centers, where $X = \{x_i | i \in [n]\}$ and $C = \{c_j | j \in [k]\}$, let c_{h_i} be the center to which x_i is assigned. The set of points, denoted by $\{x_i | Pd(x_i, \overrightarrow{c_{h_i} c_j}) > \frac{\delta}{2} d(c_{h_i}, c_j), i \in [n], j \in [k], j \neq h_i\}$, forms what is called a fringe set, and the points in the fringe set are called fringe points, where δ is a constant specified by the input of the algorithm.*

Algorithm 4: IterativeDeletion

Input: Data points $X = \{x_i | i = 1, 2, \ldots, n\}$, the number of clusters k, and the "fringe set" region parameter δ.
Output: Labels of the clustering partitions.

Let $C = \{c_1, \ldots, c_k\}$ and $L = (l_1, \ldots, l_n)$ be the set of the centers and labels returned by the k-means++ algorithm, respectively;
Initialize $C^{nF}, C^F, C' \leftarrow C$ and $L^{nF}, L^F, L', \leftarrow L$;
while True **do**
  for all $i = 1, 2, \ldots, n$ **do**
    Let c_{h_i} be the center that x_i is assigned to;
    for all $j = 1, 2, \ldots, k$ **do**
      If $Pd(x_i, \overrightarrow{c_{h_i} c_j}) > \frac{\delta}{2} d(c_{h_i}, c_j)$ and $j \neq h$, $l_{ih_i}^{nF} \leftarrow 0$ and $l_{ij}^F \leftarrow 1$;
    end for
  end for
  for all $j = 1, 2, \ldots, k$ **do**
    $c_j^{nF} \leftarrow \frac{\sum_{i \in [n]} l_{ij}^{nF} x_i}{\sum_{i \in [n]} l_{ij}^{nF}}$, $c_j^F \leftarrow \frac{\sum_{i \in [n]} l_{ij}^F x_i}{\sum_{i[n]} l_{ij}^F}$, $c_j' \leftarrow \frac{c_j^{nF} + c_j^F}{2}$;
  end for
  Update the labels $L' = (l_1', \ldots, l_n')$ as $l_i' = (l_{i1}', \ldots, l_{ik}')$, where $l_{ij}' = 1$ if $j = \arg\min_{j \in [k]} d(x_i, c_j')$, otherwise $l_{ij}' = 0$;
  If $L' \neq L$, $L^{nF}, L^F, L \leftarrow L'$. Otherwise, stop the while loop;
end while
return L;

Experiment

In this section, we conduct empirical experiments on different synthetic and real-world datasets to evaluate the performance of our proposed SEC algorithm. All the experiments are conducted on 72 Intel Xeon Gold 6230 CPUs with 500GB memory.

Algorithms. In our experiment, we consider six algorithms: our SEC algorithm, the k-means++ algorithm (denoted as KM) in (Arthur and Vassilvitskii 2007), the fuzzy k-means algorithm (denoted as FK) in (Ruspini 1969) (a variation of the k-means algorithm which assigns each point to the center with probability), the DBSCAN algorithm (denoted as DB) in (Ester et al. 1996) (a classic density-based method), the LDP-MST algorithm (denoted as LM) in (Cheng et al. 2019) (the state-of-the-art MST and density-based clustering method), and the HCDC algorithm (denoted as HC) in (Yang et al. 2023) (the state-of-the-art algorithm for hierarchical clustering methods).

Datasets. We evaluate the effectiveness of our algorithm on both synthetic and real-world datasets. Following the prior work (Arthur and Vassilvitskii 2007; Ester et al. 1996; Cheng et al. 2019; Yang et al. 2023), synthetic datasets are two-dimensional, and all the real-world datasets can be found in the UCI machine learning repository [1].

Experimental Setup. The distances between data points are set as Euclidean distances. Following the settings in (Arthur and Vassilvitskii 2007; Ester et al. 1996; Cheng et al. 2019; Yang et al. 2023), we run all the algorithms on each dataset for five times and report the average results. For our algorithm, we also study the influence of structural entropy and design an ablation experiment to show the impact of iterative construction of the encoding tree and iterative pre-deletion of the fringe set (see the full version).

Evaluation. Following the settings in (Arthur and Vassilvitskii 2007; Ester et al. 1996; Cheng et al. 2019; Yang et al. 2023), we use two external evaluation criteria: accuracy (ACC) and normalized mutual information (NMI). With ACC, we can evaluate the percentage of correctly assigned data points. For NMI, it measures the similarity between the clustering results and the true partitioning of the datasets.

Experiment on Synthetic Datasets. The synthetic datasets are summarized in Table 1. For each dataset X, the fringe set ratio is measured as the number of fringe points relative to the data size, denoted as $FR = |FP|/|X|$ in the table.

Datasets	Instances	Clusters	FR
square5	1000	4	0.133
compound	399	6	0.318
ds4c2sc8	462	8	0.050
threenorm	1000	2	0.153
aggre	788	7	0.039
2d-3c-no123	715	3	0.010
2d-20c-no0	1517	20	0.010
s4	5000	15	0.202
sizes5	1000	4	0.018

Table 1: Synthetic datasets

[1] https://archive.ics.uci.edu

Data	KM	FK	DB	HC	LM	Ours
square5	0.865	0.865	0.733	0.278	0.556	**0.868**
compound	0.654	0.659	0.699	**0.995**	0.807	0.972
ds4c2sc8	0.760	0.903	0.656	0.589	0.762	**0.961**
threenorm	0.672	0.646	0.823	0.514	0.623	**0.896**
aggre	0.784	0.792	0.990	0.996	0.997	**1.0**
2d3cno123	0.867	0.794	0.964	0.994	0.917	**0.997**
2d20cno0	0.941	0.860	0.978	0.994	0.962	**0.995**
s4	0.795	0.8	0.5328	0.288	0.662	**0.806**
sizes5	0.978	0.689	0.965	0.991	0.991	**0.997**

Table 2: Results of ACC scores on synthetic datasets

Tables 2 and 3 show the comparison results on 9 synthetic datasets. Our SEC algorithm outperforms others in clustering ACC on 8 datasets. On average, SEC improves ACC by 17.8%, 22.9%, 19.3%, 58.7%, and 20.2% over k-means++, fuzzy k-means, DBSCAN, HCDC, and LDP-MST, respectively. By fixing the state-of-the-art result as a reference, the average ACC is increased by 1.7%. Similarly, for NMI, the average clustering NMI of our SEC algorithm is increased by 61.5%, 98.0%, 26.5%, 311.7%, and 49.1%, respectively. By fixing the state-of-the-art result as a reference, the average NMI is increased by 7.9%.

Data	KM	FK	DB	HC	LM	Ours
square5	0.642	0.641	0.461	0.073	0.406	**0.649**
compound	0.718	0.711	0.641	**0.987**	0.852	0.935
ds4c2sc8	0.748	0.846	0.690	0.726	0.815	**0.921**
threenorm	0.097	0.066	0.328	0.026	0.122	**0.520**
aggre	0.879	0.849	0.979	0.988	0.992	**1.0**
2d3cno123	0.720	0.611	0.850	0.965	0.806	**0.984**
2d20cno0	0.963	0.936	0.980	0.991	0.981	**0.993**
s4	0.720	0.721	0.589	0.386	0.688	**0.731**
sizes5	0.884	0.566	0.837	0.943	0.941	**0.977**

Table 3: Results of NMI scores on synthetic datasets

Real-World Datasets. Table 4 summarizes the real-world datasets used in our experiments. Table 5 and Table 6 show the comparison results on the real-world datasets.

Datasets	Instances	Dimensions	Clusters	FR
iris	150	4	3	0.073
wine	178	13	3	0.275
glass	214	9	6	0.519
PenDigits	10992	16	10	0.185
Yeast	1484	8	10	0.499
segment	2310	19	7	0.277
control	600	60	6	0.148

Table 4: Real-world datasets

It can be seen from Tables 5 and 6 that our SEC algorithm outperforms the existing clustering algorithms on all of the real-world datasets used in our experiments. By calculating the average values over all datasets, on average, SEC improves ACC by 21.9%, 28.8%, 25.1%, 43.2%, and 29.3% over k-means++, fuzzy k-means, DBSCAN, HCDC, and LDP-MST, respectively. The NMI improvements are 19.3%,

47.4%, 34.7%, 52.2%, and 18.9%, respectively. In particular, for more than half of the datasets used in the experiments, the clustering ACC and NMI are increased by 20.6% and 9.8%, respectively. In the full version, we present the experimental results of 92 synthetic and real-world datasets used in other clustering methods in the related work. On average, our SEC algorithm improves the ACC and NMI by 3.09% and 18.82%, respectively.

Data	KM	FK	DB	HC	LM	Ours
iris	0.893	0.893	0.893	0.907	**0.973**	**0.973**
wine	0.944	0.697	0.668	0.562	0.983	**0.989**
glass	0.542	0.495	0.491	0.407	0.449	**0.626**
PenDigits	0.667	0.684	0.741	0.532	0.744	**0.812**
Yeast	0.377	0.327	0.389	0.396	0.416	**0.428**
segment	0.501	0.493	0.530	0.452	0.552	**0.770**
control	0.645	0.737	0.687	0.648	0.408	**0.872**

Table 5: Results of ACC scores on real-world datasets

Data	KM	FK	DB	HC	LM	Ours
iris	0.758	0.893	0.734	0.806	**0.901**	**0.901**
wine	0.816	0.421	0.416	0.299	0.928	**0.947**
glass	0.418	0.355	0.436	0.307	0.284	**0.534**
PenDigits	0.682	0.638	0.729	0.651	0.770	**0.779**
Yeast	0.271	0.176	0.2	0.246	0.256	**0.295**
segment	0.510	0.477	0.602	0.564	0.686	**0.687**
control	0.726	0.679	0.821	0.816	0.671	**0.868**

Table 6: Results of NMI scores on real-world datasets

Discussion on Experiments. To better show the results of the SEC algorithm, we mainly divide the datasets into four categories: (1) Type-1: datasets with stripe-like shape and without fringe sets; (2) Type-2: datasets with fringe sets and without stripe-like shape; (3) Type-3: datasets with fringe sets and stripe-like shape; (4) Type-4: datasets without stripe-like shape and fringe sets. Figure 3 provides an illustration of typical examples for each type of the dataset.

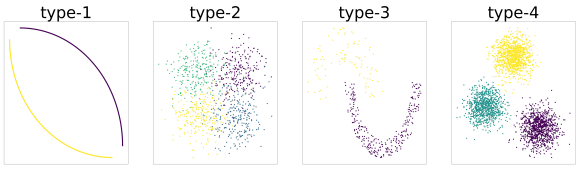


Figure 3: Typical examples for each type of the datasets

Table 7 summarizes the statistical analysis for different types of datasets used in the experiments. We report the number of each type of dataset, the ACC and NMI improvements using the SEC algorithm, and the number of datasets where the SEC algorithm finds optimal clustering partitions

	numbers	ACC boost	NMI boost	count of opt
type-1	5	-	-	5
type-2	20	2.720%	5.169%	5
type-3	34	2.391%	47.260%	21
type-4	47	5.060%	1.240%	21

Table 7: Statistical analysis of the experimental results

Figure 4 demonstrates the impact of structural entropy on clustering quality, where the visualization for four of the two-dimensional datasets with stripe-like shapes and without fringe sets is provided. We give plots of the raw datasets and the datasets after incorporating structural entropy. The k-means algorithm achieves ACCs of 0.25, 0.702, 0.706, and 0.7406, respectively; and our algorithm achieves 1, 1, 1, and 1, respectively. It can be seen that the four datasets do not have distinct Voronoi structures. As pointed out in (Reddy, Jana, and Member 2012), it is hard for k-means algorithms to get high-quality results on these datasets. The new datasets, obtained after dimensionality expansion using structural entropy, exhibit distinct Voronoi structures, which is the main reason that high-quality results can be obtained.

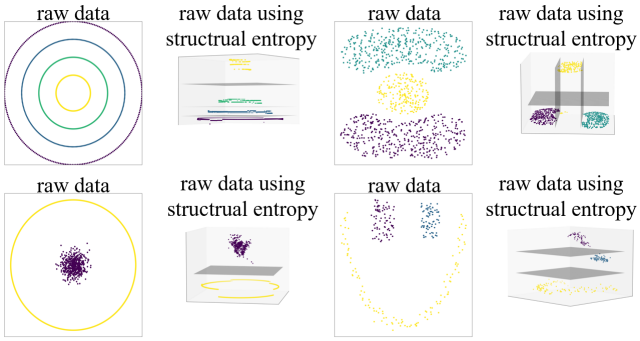


Figure 4: Comparisons of datasets with and without global structural features

Conclusion

In this work, we propose a new clustering method called SEC, which uses structural entropy to obtain global structural features to guide the clustering process. Moreover, a pre-deletion and reassignment method is used in SEC to handle datasets with fringe sets to obtain better clustering performance. Experimental results show that our proposed SEC method outperforms other state-of-the-art clustering algorithms in terms of clustering quality.

Acknowledgments

This work was supported by National Natural Science Foundation of China (62172446, 62350004, 62332020), Open Project of Xiangjiang Laboratory (22XJ02002, 22XJ03005), and Central South University Research Programme of Advanced Interdisciplinary Studies (2023QYJC023). This work was also carried out in part using computing resources at the High Performance Computing Center of Central South University.

References

Ankerst, M.; Breunig, M. M.; Kriegel, H.-P.; and Sander, J. 1999. OPTICS: Ordering points to identify the clustering structure. *ACM Sigmod Record*, 28(2): 49–60.

Arthur, D.; and Vassilvitskii, S. 2007. k-means++: the advantages of careful seeding. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1027–1035.

Bhattacharjee, P.; and Mitra, P. 2021. A survey of density based clustering algorithms. *Frontiers of Computer Science*, 15: 1–27.

Blömer, J.; Lammersen, C.; Schmidt, M.; and Sohler, C. 2016. Theoretical analysis of the k-means algorithm - a survey. In *Algorithm Engineering: Selected Results and Surveys*, volume 9220, 81–116. Springer.

Campello, R. J. G. B.; Moulavi, D.; and Sander, J. 2013. Density-based clustering based on hierarchical density estimates. In *Proceedings of the 18th Pacific-Asia Knowledge Discovery and Data Mining*, volume 7819, 160–172.

Cheng, D.; Zhu, Q.; Huang, J.; Wu, Q.; and Yang, L. 2019. Clustering with local density peaks-based minimum spanning tree. *IEEE Transactions on Knowledge and Data Engineering*, 33(2): 374–387.

Chowdhury, N.; and Murthy, C. 1997. Minimal spanning tree based clustering technique: relationship with bayes classifier. *Pattern Recognition*, 30(11): 1919–1929.

Du, H.-Y.; and Wang, W.-J. 2022. A clustering ensemble framework with integration of data characteristics and structure information: a graph neural networks approach. *Mathematics*, 10(11): 1834.

Ester, M.; Kriegel, H.; Sander, J.; and Xu, X. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, 226–231.

Gracia, C.; and Binefa, X. 2011. On hierarchical clustering for speech phonetic segmentation. In *Proceedings of the 19th European Signal Processing Conference*, 2128–2132.

Hou, J.; Gao, H.; and Li, X. 2016. DSets-DBSCAN: A parameter-free clustering algorithm. *IEEE Transactions on Image Processing*, 25(7): 3182–3193.

Huang, W.; Peng, Y.; Ge, Y.; and Kong, W. 2021. A new Kmeans clustering model and its generalization achieved by joint spectral embedding and rotation. *PeerJ Computer Science*, 7: e450.

Lattanzi, S.; and Sohler, C. 2019. A better k-means++ algorithm via local search. In *Proceedings of the 33rd International Conference on Machine Learning*, volume 97, 3662–3671.

Li, A.; and Pan, Y. 2016. Structural information and dynamical complexity of networks. *IEEE Transactions on Information Theory*, 62(6): 3290–3339.

Li, Y.; and Wu, H. 2012. A clustering method based on K-means algorithm. *Physics Procedia*, 25: 1104–1109.

Lloyd, S. P. 1982. Least squares quantization in PCM. *IEEE Transactions on information Theory*, 28(2): 129–136.

Mawati, R.; Sumertajaya, I. M.; and Afendi, F. M. 2014. Modified centroid selection method of K-means clustering. *IOSR Journal of Mathematics*, 10(2): 49–53.

Nainggolan, R.; Perangin-angin, R.; Simarmata, E.; and Tarigan, A. F. 2019. Improved the performance of the K-Means cluster using the sum of squared error (SSE) optimized by using the elbow method. *Journal of Physics: Conference Series*, 1361(1): 012015.

Ram, A.; Sunita, J.; Jalal, A.; and Manoj, K. 2010. A density based algorithm for discovering density varied clusters in large spatial databases. *International Journal of Computer Applications*, 3: 1–4.

Reddy, D.; Jana, P. K.; and Member, I. S. 2012. Initialization for K-means clustering using voronoi diagram. *Procedia Technology*, 4: 395–400.

Ruspini, E. H. 1969. A new approach to clustering. *Information and Control*, 15(1): 22–32.

Sneath, P. H. A. 1973. The principles and practice of numerical classification. *Numerical Taxonomy*, 573.

Steinbach, M.; Karypis, G.; and Kumar, V. 2000. A comparison of document clustering techniques. *Technical Report*.

Yang, J.; Wang, Y.-K.; Yao, X.; and Lin, C.-T. 2021. Adaptive initialization method for K-means algorithm. *Frontiers in Artificial Intelligence*, 4: 740817.

Yang, Q.-F.; Gao, W.-Y.; Han, G.; Li, Z.-Y.; Tian, M.; Zhu, S.-H.; and Deng, Y.-h. 2023. HCDC: A novel hierarchical clustering algorithm based on density-distance cores for data sets with varying density. *Information Systems*, 114: 102159.

Zahn, C. T. 1971. Graph-theoretical methods for detecting and describing gestalt clusters. *IEEE Transactions on Computers*, 100(1): 68–86.

Zhou, X.; Gu, J.; Shen, S.; Ma, H.; Miao, F.; Zhang, H.; and Gong, H. 2017. An automatic k-means clustering algorithm of GPS data combining a novel niche genetic algorithm with noise and density. *ISPRS International Journal of Geo-Information*, 6(12): 392.