EURASIP Journal on Wireless Communications and Networking

RESEARCH

Open Access

SKDStream: a dynamic clustering algorithm on time-decaying data stream

Hui Liu[1], Aihua Wu[1*], Mingkang Wei[1] and Chin-Chen Chang[2]

*Correspondence: ahwu@shmtu.edu.cn

[1] Department of Engineering, Shanghai Maritime University, Shanghai, China
[2] Department of Information Engineering and Computer Science, Feng Chia University, Taichung, Taiwan

Abstract

Data stream is a type of data that continue to grow over time. For example, network security data stream will constantly be generated in the field of data security, and encrypted data stream will be generated in the privacy protection scenario. Clustering is a basic task in the analysis of data stream. In addition to the large amount of data and limited computer memory, there are the following challenges in time-decaying data stream clustering: (1) How to quickly process time-varying data stream and how to quickly save vaild data. (2) How to maintain and update clusters and track their evolution in real time. Based on the fact that the existing data stream algorithms do not provide a good strategy to the above problems, this paper proposes a dynamic clustering algorithm named SKDStream. The algorithm divides the entire data space into distinct minimal bound hypercubes, which are maintained and indexed by a newly defined structure, SKDTree, that aggregates and updates clusters in real time without requiring large primary storage. Clusters are composed of dense hypercubes. Experiments on synthetic datasets and real datasets show that the response time of the algorithm is similar to that of existing dataflow algorithms, but the quality of the generated clusters is relatively stable over time. Furthermore, the SKDStream algorithm is able to track the evolution of the number of clusters, centers, and density in real time, and compared to D-stream, SKDStream is efficient and effective in clustering.

Keywords: Data mining algorithm, Density-based clustering, Decaying stream, High density subspace

1 Introduction

Time-decaying data stream is a kind of stream with vitality relationship which decays over time. Each of these data points is alive and has a vitality attribute. It begins to fade out as soon as arrives and eventually becomes invalid. The dynamic clustering of time-decaying data stream focuses on the real time of data that are not attenuated, and pays more attention to the recent changes. For each of data point, the higher the priority level is in the short term, the lower the priority level is in the longer term. It is mainly used in scenarios such as communication data encrypting, network attack monitoring, public opinion monitoring, real-time analysis of stock trading and weather forecasting. Real-time clustering of the unattenuated part of the data stream has important practical

© The Author(s) 2022. **Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

significance, which can not only help people understand the current focus, but also provide a basis for further analysis of its evolution.

Scenario 1: Detection of hot and sensitive topics in public discussion. Comments on social media are typical time-decaying data stream, which contain socially sensitive topics [10], one of the focus of public opinion monitoring. In traditional data analysis, hot topics can be revealed by density-based clustering of a given set of comments. However, in practical applications, based on the dynamic, sensitive and frequently changing characteristics of comment sets, people are more inclined to pay attention to the latest topics. To find such topics, obsolete data should not be included in clustering. How to cluster the comments stream to find current sensitive topics is a typical dynamic clustering problem on time-decaying data stream.

Scenario 2: Analysis of eye-tracking information in recommendation system. A new technology in the field of e-commerce is to capture the users' interest or theme by tracking their eye-movements. The eye-tracking information is generated continuously while browsing the web, which is fast-changing, orderly, and potentially limitless.

Finding recommended objects can be accomplished by clustering user eye-tracking stream [26]. The dynamic changes in these recommended objects should reflect current interests and should not be swayed by outdated data. Obviously, this is also a typical dynamic clustering problem on time-decaying data stream.

Scenario 3: Identification of malicious network attacks. Network access [30] is another kind of time-decaying data stream. If new clusters are discovered through dynamic clustering, it indicates the start of a wave of malicious attacks. Capturing such changing clusters and tracking their evolutionary trends has important practical implications, helping to identify malicious attacks in real time and take appropriate preventive measures.

Unlike normal stream, each data in the time-decaying stream has life, with the highest vitality at birth as time goes by, so it is a challenge to dynamically cluster such data. Firstly, except for restriction on traditional stream clustering, such as computer, memory, and time constraints, source data are dynamic and require an appropriate storage strategy to store newly arrived data and eliminate outdated data in time. Secondly, a cluster at one moment may no longer meet the density requirement as time going, some data in it have no life and should be abandoned while the others need to be merged into new clusters. Thus, the clustering algorithm must have the ability to automatically detect the change in source data and dynamically adapt the clusters. The reorganization of clusters in turn requires the original data to be traceable, which is a huge challenge to limited memory. The key point for these challenges is an effective storage structure so that it can dynamically adapt to fast-arriving stream, and timely remove the influence of outdated data accurately, it can also ensure that newly arrived data can be classified into appropriate clusters in one scan and existing clusters can be merged and reorganized immediately, or a solution to extract information in one scan which can approximately represent and restore all valid data, so the dynamic clustering algorithm can match the scheme.

We propose SKDStream, a dynamic clustering algorithm based on time decaying-data streams, which treats clusters as a dense data space and uses three types of storage within it: a fragment of main storage for data before forming initial clusters and few live outliers, an index tree for clusters and a group of disk files for cluster details.

At the beginning, when one window data come, a clustering will be activated, once the first cluster formed, an index tree named SKDTree will be created, and all data in the cluster will be summarized as a leaf node of SKDTree, while the other data are still stored in the main memory. Later, when new window of data comes, for any one of its data, if it belongs to a cluster space, it will not be stored but be summarized in the cluster node, otherwise, it will be kept in the main memory. Once the main memory is full, outdated data will be discarded and a new clustering will be activated. SKDTree, an improved version of KD-tree [5], used in this paper to store cluster units, each node of which represents a downwardly contained data space. Particularly each leaf node of SKDTree represents a smallest cluster, a data space with a density exceeding given threshold. As new data rolling up, a subtree of cluster may appear where density of each non-leaf node is also larger than the threshold. The whole subtree will be fanned out as a disk file, only the root will remain in the SKDTree, i.e., the SKDTree will only keep cluster units and not any subsets of them. Thereby the tree height is limited and can be maintained in memory. With the dynamic changes in the data, the subtrees fanned out to the hard disk may be read back, which not only ensures the validity of the dynamic clustering results, but also reduces the data storage pressure of the algorithm.

Main contributions of this paper are threefold:

- A new tool for describing the available data space with redefined minimum bounding hypercube in stream clustering. Minimum bounding hypercube is a subspace of data points and a closed volume that contains all its descendant spaces. It can be described by density, effective value, boundary range, etc. Depending on the time span of stream, the density of the hypercube can be determined through the formula defined in this paper, thereby we can dynamically adjust the clustering results. By reducing memory and computing costs, the minimum bounding hypercube works as an effective technology for dynamic clustering.
- We propose a new cluster storage structure SKDTree. The SKDTree divides the data space from top to bottom until the cluster cells. Only by dynamically maintaining summary information such as the clusters' density and their ancestor nodes, the cluster space can be summarized upwards in a prompt and efficient manner. This is a simple and straightforward structure and needs far less memory. The nature of its KD-tree is conducive to efficiently query the space of adjacent clusters, and by further merging them, the final clustering results are obtained.
- A dynamic clustering algorithm for time-decaying data stream has been proposed. It can summarize high-density subspace online. Compared with existing algorithms, it supports obtaining evolving clustering results. The clusters are formed by the summarizing and merging of nodes in SKDTree. The number of clusters, cluster center, and cluster density are all dynamic, reflecting the clustering of current valid data.

Next, the second section is related work, the third section introduces the basic concepts, the fourth section introduces the structures and operation of the SKDTree, the fifth section of the SKDStream algorithm, the sixth section is the experimental part of the algorithm.

2 Related work

In recent years, many data stream clustering algorithms have been proposed.

The two-stage clustering algorithm [2, 4, 8, 9, 17, 25] is a classic solution to solve the data stream clustering problem. Algorithms extract summary statistics from valid data in the online phase. When entering a new data, the summary statistics will be updated. Then performing clustering in the offline phase to obtain clusters. This type of algorithms can obtain clusters on the data stream, but require frequent execution of the offline process, which is time-consuming and computationally expensive. It also cannot dynamically update clusters and track the evolution process. The article [4] proposed an improvement to it, choosing an appropriate time to run the offline process, but it did not really solve the high cost of the essential problem of the offline process.

In recent years, scholars have proposed algorithms for online processing of data stream clustering problems, such as [11, 13, 20, 21, 28, 29]. The algorithms perform clustering for each newly entered data to obtain the latest clustering results. Due to memory constraints, and usually the most recent data are more valuable. The proposed algorithms are designed based on the window model or decaying models. With the passage of time, the vitality of data continues to decay. Some define the data life weights according to their vitality. When the life weights is less than the threshold, the data will be deleted directly, and the deleted data will be permanently invalid, and will not be restored again, and affect any subsequent clustering results.

Existing online algorithms are mainly divided into two categories: partitioning-based and density-based algorithms. Partitioning-based data stream clustering algorithms group data into a predefined number of clusters in the light of the distance (or similarity) to cluster centroids. Thereby, the number of clusters should be set up in advance. Algorithms choose to insert existing clusters based on the distance between newly entered data and existing clusters or initialize a new cluster, such as [1, 15, 18, 22]. Partitioning-based algorithms are simple and feasible in general. However, they are highly dependent on distance, radius and other parameters, and only spherical clusters are generated.

The density-based data stream clustering algorithms [7–9, 16, 20, 24] find clusters according to the dense area. The algorithms abstract the summary information of the arrived data constantly in many existing cluster cells, which can be represented as feature vectors. These cluster cells are then merged into clusters according to their accessibility and connectivity. Density-based algorithms are able to monitor the change in the number of clusters directly. They have the capability to handle noise data with strong robustness too. The ref. [16] proposed a data stream clustering algorithm EDMStream that explores the evolution of the density mountain. The core idea is to use the density mountain to abstract the distribution of real-time data stream, and on this basis to track the changes in data distribution and the evolution of clusters, yet unfortunately the algorithm has fixed the central point at the beginning, and will not update them later. Consequently, the algorithm can only track the disappearance of clusters and the adjustment between clusters, and is unable to identify the emergence of a new cluster. CEDAS [19] is a fully online density-based time-decaying stream clustering algorithm. Specifically, CEDAS clusters and creates a graph structure. In the graph structure, the nodes are the micro-clusters and the edges are the connectivity between micro-clusters. But when the density of data space is not even, the clustering result will be low quality.

Another typical density algorithms, grid-based data stream clustering algorithm [6, 12, 14, 27], divides the data space where the stream data are grouped into multiple grid units, all data are mapped to the grid, and clusters are achieved by recording the density of the grid. In order to reduce the calculation and memory costs, grid cells in clustering algorithms are generally of fixed size, however an inflexible structure generally brings limitations. Furthermore, sparse grid cells will be directly discarded in the final clustering, which affects the accuracy of the final clustering results. MuDi-Stream [3] is a hybrid algorithm which is based on both density-based and grid-based approaches. It customizes the density threshold for each cluster and overcomes the problem multi density clusters. However, the calculation is high, at the cost of reducing quality of clusters.

The SKDStream algorithm proposed in this paper is a density-based clustering algorithm for data stream, of which the life weights is designed based on common decaying model, whereas obsolete data remain stored in a certain way and are permitted for subsequent clustering. Moreover, the online clustering updates the stream data clusters at any time, and tracks the evolution clustering results in real time. The center, number, and density of the evolutions clusters are constantly changing. Additionally, the relevant information of all subspaces is stored in the corresponding nodes of the SKDTree during the execution of the algorithm, which means no small density space will be discarded randomly but strictly according to the decaying function, therefore real-time and accurate clustering results are promised.

3 Methods

3.1 Basic conceptions and problem definition

Definition 1 Data Stream

A data stream $S = \{X_{t_1}, X_{t_2}, \ldots, X_{t_N}\}(N \to \infty)$ is a sequence of data with same characteristics that arrives in a period of time. Let t_1 be the start time, and t_N be the current time. $X_{t_i} = \{x_j = \langle x_j^1, x_j^2, \ldots, x_j^d \rangle\}(j > 0)$ is the set of data that arrived at the time t_j, where each data x_j has d attributes.

Definition 2 Life Weights

Given a data X_i that arrives at time t_i in stream data S, of which the life weights at time t is defined as:

$$f_{x_i}(t_i, t) = 2^{-\lambda(t-t_i)} \tag{1}$$

λ is the decay parameter ($\lambda \geq 1$). The larger the λ is, the faster the data decay and thus the weaker the life weights turns. Obviously, the life weights of X_i constantly decreases over time until drops below the threshold, and consequent that point will be considered invalid. This is a wisely used decay function in many stream clustering algorithms. In this paper, we choose such that fit is in the range (0,1].

Definition 3 Time-Decaying Data Stream

Given a data stream $S = \{X_{t_1}, X_{t_2}, \ldots, X_{t_N}\}(N \to \infty)$, a life weights f and a threshold τ, Time-Decaying Data Stream DS at current moment t_N is a subset of S, that is, $DS = \{X_{t_j} = x_j = \langle x_j^1, x_j^2, , x_j^d \rangle | (f^{x_j}(t_j, t) > \tau) \wedge X_{t_j} \in \mathrm{S}\ \}$.

The time-decaying data stream includes the data whose life weights is larger than the threshold in the original data stream S. As shown in Fig. 1, S stores all the current data from the beginning to t_N, and the DS only remains the current data whose life weights greater than τ, while data below the life weights threshold τ are regarded stale data.

Definition 4 Minimum Bounding Hypercube (MBH)

Given a set of data $X = \{x_n\}(n \in \mathbb{Z})$ where each data $x_n = (x_n^1, x_n^2, \ldots, x_n^k)(k \in Z)$ has k-dimensional features, the corresponding MBH can be represented by M^k:

$$M^k = \langle m_1, m_2, \ldots, m_k \rangle$$

$m_k = \langle min(Project_k(X)), max(Project_k(X)) \rangle$ represents the binary combination of the minimum and maximum of the projection of the set X on the kth dimension.

Project() is a projection function, and $Project_k(X) = \{x_n^k\}(n \in \mathbb{Z})$ is the projection of the x_n on kth dimension. It represents the vertical distance to the plane in the kth dimension.

When $k = 2$, the minimum bounding rectangle MBR is a rectangular space.

Given two MBHs $M^k = \langle m_1, m_2, \ldots, m_k \rangle, N^k = \langle n_1, n_2, \ldots, n_k \rangle$:

- If $m_i \cap n_i \neq \varnothing$ for any $i (1 \leq i \leq k)$, then M^k intersects with N^k;
- If there exists at least an $i (1 \leq i \leq k)$ such that $max(m_i) = min(n_i)$, and $m_j \cap n_j \neq \varnothing$ for any $j (1 \leq j \leq k, j \neq i)$, then M^k is adjacent to N^k;
- Otherwise, M^k is separated from N^k.

Any MBH can be divided into multiple adjacent MBHs.

Definition 5 Reachable Space

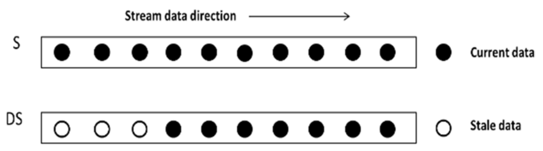


Fig. 1 Stream data at t_N.In the figure 1, the black solid circle represents the current data, and the black hollow circle represents the stale data

Given a set of MBHs $\mathcal{S} = \{M_1^k, M_3^k, , M_n^k\}$ in a k-dimensional space. $\mathcal{S}$ is a reachable space if for any of its MBH $M_i^k (1 \leq i \leq n)$, there is at least another hypercube $M_j^k (1 \leq j \leq n, i \neq j)$ in $\mathcal{S}$ that M_i^k is adjacent to M_j^k.

If there exist no superset $\mathcal{S}'$ of $\mathcal{S}$ so that $\mathcal{S}'$ is also a reachable space, then $\mathcal{S}$ is a **maximum reachable space** of the k-dimensional space.

Definition 6 Density(ρ_t)

Given a multidimensional space D that contains multiple data. The density of D at time t_i is defined as

$$\rho_{t_i} = \frac{|D|}{v(D)} \tag{2}$$

where $v(D)$ represents hyperplane volume of D, $|D|$ represents the number of data inside D at t_i. Also, t_i could be defined as a period of time.

$\rho_{t_{i+1}}$, the density of D at the next time t_{i+1} is defined as:

$$\rho_{t_{i+1}} = \rho_i * f(t_i, t_{i+1}) + \rho_{t_i \sim t_{i+1}}$$

where $f(t_i, t_{i+1})$ represents the life weights of data from t_i to t_{i+1}, and $\rho_{t_i \sim t_{i+1}}$ represents density of new data which arrive between t_i and t_{i+1}.

Definition 7 Cluster Cell

Given a data set S and a MBH M^k in DS. The density of M^k has been defined as $\rho_t(M^k)$. If $\rho_t(M^k) > \Theta$, then MBH M^k is a cluster cell at time t.

Definition 8 Dynamic Cluster

Given a data set S and a maximum reachable space in $\mathcal{RS}$. The density of $\mathcal{RS}$ has been defined as $\rho_t(\mathcal{RS})$. If $\rho_t(\mathcal{RS}) > \Theta$,then maximum reachable space $\mathcal{RS}$ is a dynamic cluster at time t in S.

At same time, points that do not belong to any dynamic cluster in data set are defined as Outliers.

We aim to calculate the dynamic cluster on time-decaying stream at a certain moment. That is, for a given time t and current data stream DS, our aim is to find the set $C_t = \{c_1^t, c_2^t, \ldots, c_m^t, c_o^t\}$, where $c_i^t (1 \leq i \leq m)$ is a dynamic cluster over DS and c_o^t is the set of outliers, that satisfies the following equations:

$$\begin{cases} c_1^t \cup c_2^t \cup \cdots \cup c_m^t \cup c_o^t = C_t \\ c_i^t \cap c_j^t = \varnothing (i \neq j) \\ c_i^t \cap c_o^t = \varnothing \\ C_t = argmin(\sum_{i=1}^{m} MBH(c_i^t)) \end{cases}$$

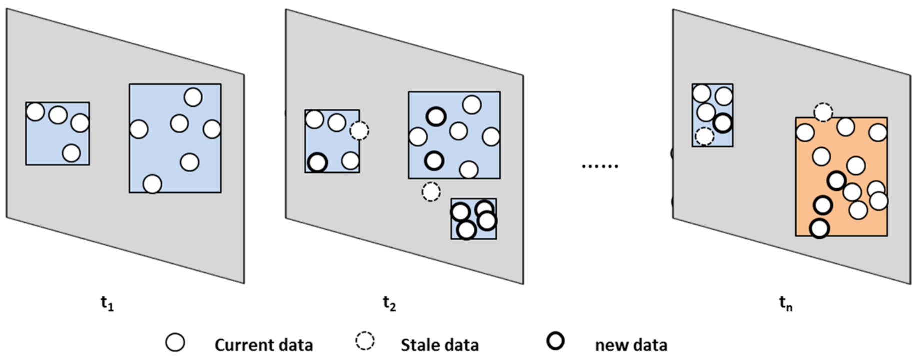


Fig. 2 Dynamic clustering on time-decaying data stream. In the figure2 , each gray area represents data clustering result at differentinstants of time; the blue rectangle represents the cluster cell and the orangerectangle represents the dynamic cluster. And the dotted hollow, the solid hollow and the thickened solid hollow circle represent the stale, curent and newdata respectively.

Fig. 2 depicts the process of dynamic clustering on a data stream. As shown, the first column includes two cluster cells (the blue rectangles) at time t_1, and then at t_2 new data arrive but some of the current data turn into the stale data, the number of cluster cells and the location of existing cluster cells are changed. The last column reflects the clustering result at t_n. It demonstrates the changing of cluster cells as one of the previous cluster cells has developed into a dynamic cluster (an orange rectangle).

3.2 SKDTree, summary files and operations

3.2.1 SKDTree definition

Definition 9 Splitting Key

In a given d-dimensional data space $\mathcal{D}$, let $S = \{M_1^d, M_2^d, \ldots, M_n^d\}$ be the set of all cluster cells of $\mathcal{D}$ at time t, and $k(1 \le k \le d)$ be a specified dimension, then the splitting key on dimension k is

$$key = argmin\{Count(O_j^k > key) - Count(O_j^k < key)\} \tag{3}$$

where O_j^k is the attribute value of the center point O_j of the j-th cluster MBH on dimension k within space D, $Count(O_j^k < key)$ are the MBHs that meet the condition $O_j^k < key$ in S while $Count(O_j^k > key)$ are the MBHs that meet the condition $O_j^k > key$.

All points whose kth attribute equals the key construct a splitting hyperplane F_k which is an axis, a plane and a volume for 2D, 3D and 4D space, respectively, and in $d(d > 4)$ dimension is a $d - 1$ dimensional hyperplane.

Any space L can be divided by a splitting hyperplane into two subspaces: one called $range_{left}$ in which points with values less than key in the dimension $(x_i^k < key)$ and the other called $range_{right}$ in which points with values larger than key in the dimension $(x_i^k > key)$. Obviously F_{key} is the upper limit of $range_{left}$ and the lower limit of $range_{right}$.

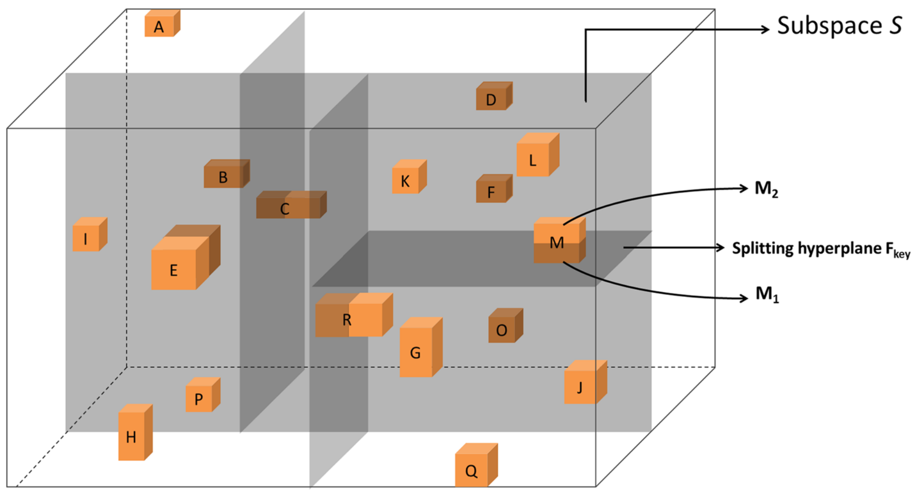


Fig. 3 Dividing instances in 3D space. In the figure3, the largest cube represents space D, and each orange cuberepresents a cluster cell. The gray planes represent splitting hyperplanes be-tween different subspaces

Table 1 Attributes of various nodes in the SKDTree

Attribute	Meaning
M^k	Custer cell
ρ_M	Leaf node density: cluster cell density
range	The space represented by node indicating MBH
dim	Splitting dimension
key	Splitting key (Eq. 3)
f	Life weights (Eq. 1)
ρ	Node density: the spatial density represented by nodes (Eq. 2)
t_{last}	The last update density time
ptrs	Pointers to the child node(non-leaf node: two pointers, sub-leaf node: 1–2d pointers)

If a cluster cell M intersects with F_{key}, M is cut by F_{key} into two cluster cells in the space of $range_{left}$ and $range_{right}$, respectively.

Fig. 3 shows dividing instances in three-dimensional space. The largest cube represents space D, which contains all cluster cells in the current time-decaying data stream. After several divisions based on the splitting key of different dimensions, it is subdivided into several subspaces. The gray planes represent splitting hyperplanes between different subspaces. A clear demonstration of space division is subspace $\mathcal{S}$ on the near right which is divided by F_{key} perpendicular to z-axis into two subspaces $\mathcal{S}1(range_{left})$ and $\mathcal{S}2(range_{right})$. Besides, after the dividing, cluster cell M splits into two cluster cells M_1 and M_2, that separately belongs to $\mathcal{S}1$ and $\mathcal{S}2$.

Definition 10 SKDTree

Given a d-dimensional data space $\mathcal{S}$, the SKDTree is a summary tree of $\mathcal{S}$ with following characteristics:

- There are three types of nodes:leaf nodes, sub-leaf nodes and non-leaf nodes, that are defined using attributes listed in Table 1, namely, $M^k, f, t_{last}, ptrs$ for leaf nodes, $range, f, \rho, t_{last}, ptrs$ for sub-leaf nodes and $range, dim, key, f, \rho, t_{last}, ptrs$ for non-leaf nodes.
- Each leaf node is a cluster cell. Sub-leaf nodes can represent the smallest space containing leaf nodes.They are the father nodes of leaf nodes and contains 1–2d pointers.
- A root node is a special non-leaf node which indicates the complete data space. Apart from leaf nodes and sub-leaf nodes are non-leaf nodes. A non-leaf node in the i-th layer(the root node if $i = 1$) has two child nodes, which represents that a parent space is split into two child spaces $range_{left}$ and $range_{right}$ by hyperplane in the (i mod d) dimension.

KD-tree is actually no stranger to common tree structures. It is essentially a multi-dimensional binary tree, which is a data structure that divides k-dimensional space. The binary query tree (ISBST) which we familiar with is essentially a KD-tree in one-dimensional space. The original KD-tree uses the currently available data to divide the k-dimensional space, and stores the instance points in the process of continuous division space. The segmentation feature enables fast retrieval of data points in the space. Also, the feature determines that such kind of tree structure can quickly query neighbors in a sample space. Each node of SKDTree splits the space cut by the corresponding parent node again, and further forms different subspace. When searching for the location of a point, it will be transformed into the subspace where the search point is located. Similarly in SKDTree, the summary information of the data space is stored hierarchically where the leaf layer is a cluster cell, but the other node is a $range_{left}$ or $range_{right}$ which covers all its children. From a spatial perspective, there is a blank space between the sub-leaf node and the leaf node, indicating that the sub-leaf node is not equal to the spatial union of its child nodes. Moreover, among other nodes, there is no blank space between the parent and the child, thus the density of the parent node is equal to the quotient of the total number of points of all the children divided by the representative space volume of the parent node. The number of points of the child node may be calculated in the other direction. Because the density of children varies as the data change, the density of its ancestor nodes must likewise be updated synchronously. We simply keep the node density of the tree until querying the clustering results to increase usability.

Whether the space of a node in SKDTree described above is equal to the sum of child nodes, is the fundamental reason for introducing sub-leaf nodes. As shown in Fig. 3, the space $\mathcal{D}$ can be described as the root node of the SKDTree. It will be split into two subspaces, $range_{left}$ and $range_{right}$, which can be represented by two child nodes of the root, and each subspace can be split into two sub-subspaces$\cdots$ In such way, the whole space can be split according to its dimensions one by one, each split produce a new level of two child nodes until the sub-leaf node, whose child represents a data space of a cluster cell.

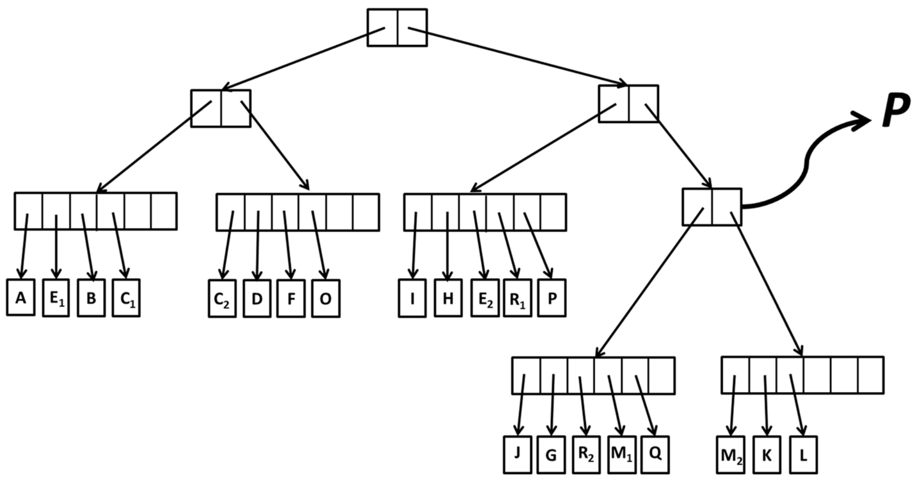


Fig. 4 SKDTree corresponding to data space

Fig. 4 is the SKDTree corresponding to the three(X, Y, Z)-dimensional data space in Fig. 3, of which the root node represents the entire data space $\mathcal{D}$. The two child nodes of root represent the two subspaces of $\mathcal{D}$ divided by the X dimension. Similarly, the four subspaces in the third layer can be obtained, respectively, with the two nodes above divided according to splitting key in Y-dimension, and they may split too. Similarly, these four nodes can also be further divided. Take node P as an example, its left and right sub-leaf nodes, separated by dimension Z, contain cluster cells J, G, Q, M_1, R_2 in subspace $\mathcal{S}_1$ and K, L, M_2 in subspace $\mathcal{S}_2$, respectively.

In the actual data space, there is no one-to-one correspondence between the cluster cells and leaf nodes. One cluster cell may be partitioned into different subspaces and spread into different leaf nodes as a result of space splitting. However, due to their irregular shapes, dynamic cluster cells in real space are more likely to have numerous leaf nodes belonging to various sub-leaf nodes, thus the union of them all are needed to restore the actual cluster cells. At a given moment, the set of all reachable cluster cells in SKDTree is queried, the dynamic clustering result of the time-decaying data stream is obtained.

However, due to cluster cells only exist in the leaf layer of the SKDTree, when the data change dynamically, the leaf layer may: (1) Form new cluster cells; (2) Increase or reduce the density of existing leaf nodes. The density of sub-leaf nodes will change as a result of these adjustments, and the level of sub-leaf layer in the tree will be affected as well (level changes only occur in sub-leaf nodes). When density of the space representedby the sub-leaf node overflows, that space must be re-divided, either by raising its level or by forming a new sub-leaf node. Due to the density change may affect a sub-leaf node or even an ancestor node, the subtree rooted at this node should be summarized to a new cluster cell, replacing the subtree in situ. Considering that the new cluster cell may not match the density requirements in the future due to density changes, it must be reverted to the subtree that are not supposed to be discarded. In this paper, this is implemented with summary files, which are stored in external storage and each records subtree separately. When data changes cause density underflow, if a cluster cell does not have a corresponding summary file, in addition to deleting the leaf node from the SKDTree, remaining data size of the cluster cell must be calculated according to the density and volume, and all of them will be evenly distributed in the original cluster cell space and put back into the data stream. Otherwise the leaf node recorded in the summary file will be reloaded into the SKDTree. Of course, the density of the leaf nodes of the repaired subtree may not meet the requirements of the cluster cell, and it is still necessary to delete the leaf nodes and put valid part of data back in the data stream.

Ancestral relationships may exist between subtrees that are summarized one after another. For example, after a certain subtree is summarized, the density of the upper subtree increases, therefore the summary operation continues upwards. If the root of a subtree in the first summary process is the leaf node of a subtree in the second summary process, they should be spliced together into the same subtree and placed in the same summary file. Given two subtrees A and B, we say that B can directly splice A when there is a leaf node in subtree A which has the same root of subtree B. When subtree C can be spliced directly to A and subtree B can be spliced directly to C, then subtree B can be spliced indirectly to A. All subtrees that can be spliced directly or indirectly should be spliced before written into the summary file.

3.2.2 SKDTree operations

As previously stated, the SKDTree changes dynamically as data arrive. To maintain and query the SKDTree's correspondence with the actual data space, some maintenance operations are necessary, including building a SKDTree, adding leaf nodes, deleting leaf nodes, summarizing the subtree, subtree re-insertion, and dynamic cluster querying.

Building a SKDTree

Given a d-dimensional data space and the set of cluster cells inside, the algorithm Build() divides the range enclosing all M^k at once along dimensions, and recursively generates the SKDTree in a top-down way. The main idea is that:

(1) When there is only one input cluster cell, the established SKDTree is a tree with only a root node, sub-leaf node, and leaf node remain the same range.
(2) When the number of input cluster cell is greater than 1 but less than or equal to 2d, the established SKDTree has only one root node, one sub-leaf node and multiple leaf nodes.
(3) When the number of input cluster cells is more than 2d, the root is created, the dimension and splitting hyperplane are generated by recursive computing. When the number of cluster cells in non-leaf nodes space is fewer than 2d, creating sub-leaf nodes and leaf nodes in their own space. The original cluster cell will be separated into two cluster cells if it crosses the splitting hyperplane during the space division process.

Fig. 5 shows the structure of the SKDTree initially built under different numbers of initial cluster cells.

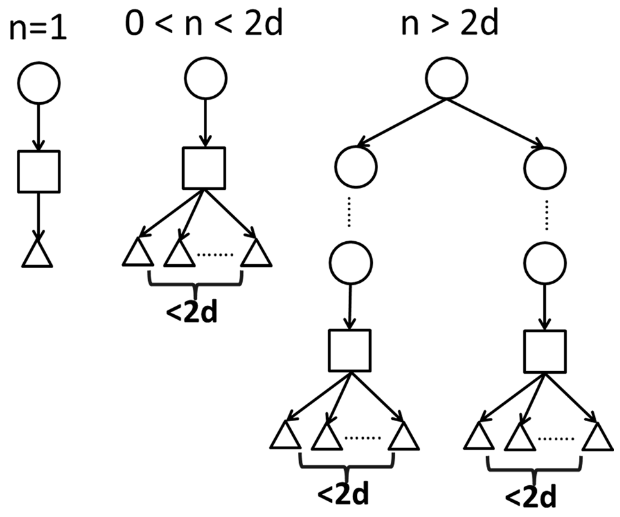


Fig. 5 Build initial SKDTree. In the figure 5, the circles represent non-leaf nodes, the squares representsub-leaf nodes and the triangles represent leaf nodes in SKDTree

Algorithm 1 Build

```
Require: $\{M^k\}$, root, $dim$, $d$
Ensure: SKDTree
 1: if count($\{M^k\}$) ¿ $2d$ then
 2:     update $dim$, calculate $key$, $range_{left}$, $range_{right}$
 3:     create non-leaf node $Q_1, Q_2$, $root.ptrs \to Q_1, Q_2$
 4:     for each $M^k$ do
 5:         if min($Project_k(M^k) < key$)   &&   max($Project_k(M^k) > key$) then
 6:             $M^k \to M_1^k, M_2^k$
 7:         end if
 8:     end for
 9:     Build($\{M^k\} \in range_{left}$, $Q_1$, $Q_1.dim$, $d$)
10:     Build($\{M^k\} \in range_{right}$, $Q_2$, $Q_2.dim$, $d$)
11: else
12:     root.ptrs $\to$ create sub-leaf node $Q$
13:     if count($\{M^k\}$) $\in (1,\ 2d]$ then
14:         create leaf node $p_i$ for each $M^k$
15:         $Q.ptrs \to p_i$
16:     else
17:         create a leaf node $p$
18:         $Q.ptrs \to p$, $p.range \leftarrow Q.range$
19:     end if
20: end if
```

Adding leaf nodes The SKDTree and the current cluster cell M^k which waiting to be added may have the relationship of intersection, separation, and inclusion.

According to the specification of SKDTree and the framework of SKDStream algorithm, as illustrated in Fig. 6a, if there exist a node in SKDTree whose range totally covers M^k, they will exist: (1) The node in question is not a leaf node. (2) There must be a node A with the lowest range that all of the other nodes must be descendants of A. We say that the cluster cell M^k and the SKDTree are included in node A.

As shown in Fig. 6b and Fig. 6c, if there are nodes in the SKDTree or its subtrees that cross the range of M^k, they will exist: (1) The nodes in question are not leaf nodes. (2) After these nodes are grouped according to their respective paths, each group must have a node with the largest range. Let them be $B_1, B_2, \ldots, B_n$. Then, we say that cluster cell M^k and the SKDTree intersect at nodes $B_1, B_2, \ldots, B_n$.

If the relationship of M^k and SKDTree (or its subtrees) neither intersection nor inclusion, then we say that M^k and SKDTree are separated, as shown in Fig. 6d. And apparently when M^k is separated from the root of the SKDTree, it is definitely separated from all other subtrees.

A new cluster cell is always inserted into the SKDTree's leaf layer, but the meaning of the corresponding insertion process is different, as is the relationship between the cluster cell and the tree. Sometimes the insertion causes the tree to grow taller, or it is required to change the range of ancestor nodes, and other times it is necessary to divide the cluster cell. Here are some fundamental operations.

Operation 1 Add a new cluster cell M^k to the sub-leaf node A.

Prerequisite: M^k and SKDTrees are included in node A.

Steps: The algorithm checks if A is a sub-leaf node with less than full children. If so, insert M^k as a child node of A. Otherwise, two sub-leaf nodes are generated, and the nodes in hyperspace become their children, as the children of A and M^k together form a hyperspace to be divided. The splitting key is calculated according to Eq. 4. Afterward, update the density of all nodes on the path using Eq. 3.

Operation 2 Split the new cluster cell M^k.

Prerequisite: M^k and SKDTree or its subtrees intersect at nodes $B_1, B_2, \ldots, B_n$

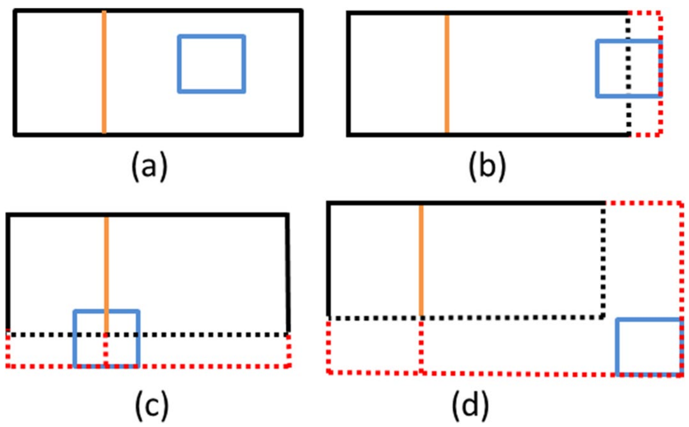


Fig. 6 The instance of updation after inserting new MBH in 2D space. **A** Upadation when the new MBH is included in only one node in SKDTree, **B** Upadation when the new MBH intersects only one node in SKDTree, **C** Upadation when the new MBH intersects more than one node in SKDTree, **D** Upadation when the new MBH separates from all the nodes in SKDTree. Note: The black rectangle represents the range of SKDTree root. The blue rectangle represents the cluster cell to be inserted. The orange solid line represents the original split hyperplane. The black dashed line represents the updated range. The red dashed line represents the new range

Steps: M^k splits into $M^{k_1},\ldots,M^{k_n},M^{k_{n+1}}$, left $M^{k_1}\in B_1,\ldots,M^{k_n}\in B_n$, and $M^{k_{n+1}}$ is the remaining part. If the remaining part (if not empty) is not the MBH, it will be split into several smallest cluster cells. Then calculate the density of cluster cell after splitting according to Eq. 3.

Example: As shown in Fig. 7a, where there is only one vertex intersection between the existing cluster cell A and the cluster cell B to be inserted, B is divided into three cluster cells B_1,B_2 and B_3 by the extended boundary line. Similarly in Fig. 7b, there are two vertices that intersect the existing cluster cell C and cluster cell D to be inserted. Then the extended intersecting boundary line divides D into four sub cluster cells D_1 ,D_2,D_3 and D_4. The overlapping space belongs to the existing cluster cell, i.e., B_3 and D_2 belong to A and C, respectively.

Operation 3:Extend the range of the SKDTree or its sub tree nodes.

Prerequisite: M^k is separated from the SKDTree.

Steps: Modify the range and density of the root of SKDTree or its subtrees so that it can absorb M^k. Repeat the following steps until the child of the sub-leaf node is discovered in the children of the changed node, then find the closest node to M^k but separate from M^k. Modify the range and density of the node so that it can absorb M^k.

Thus, the algorithm steps to insert cluster cell M^k into the SKDTree is:

Examining the relationship between M^k and root A, and:

(1) If M^k is separated from A, perform operation 3 then operation 1, thereupon insert M^k into the modified sub-leaf node.

(2) If M^k and SKDTree included in A, and A is a non-leaf node, execute operation 2 then split M^k into a left child $M^{k_1}\in A$ and a right child of $M^{k_2}\in A$, each of which will be inserted recursively into the left and right subtrees of A. If A is a sub-leaf node, perform operation 1 and just insert M^k.

(3) If M^k and SKDTree or its subtrees intersect at nodes $B_1,B_2,\ldots,B_n$, perform operation 2, and divide M^k into $M^{k_1}\in B_1,\ldots,M^{k_n}\in B_n$ and the remainder $M^{k_{n+1}},M^{k_{n+2}},\&,M^{k_{n+n}}$. For each M^{k_i}, insert it recursively into the subtree with B_i as the root node. For each $M^{k_{n+j}}$, first execute operation 3 and then operation 1, and insert $M^{k_{n+j}}$ into the modified sub-leaf node.

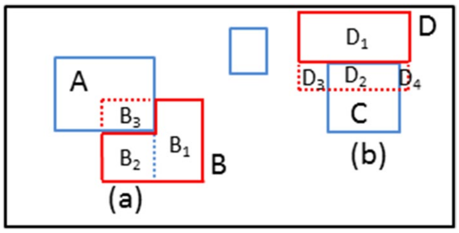


Fig. 7 The instance of intersection and division after new inserting MBH in 2D space. **A** Division when nodes intersect on one dimension, **B** Division when nodes intersect on multi dimensions. Note: The blue rectangle indicates an existing cluster cell, and the red rectangle indicates a new cluster cell

```
Algorithm 2 AddLeafNode
Require: $M^k$, root, $d$
Ensure: SKDTree
 1: if $M^k \nsubseteq root.range$ then
 2:     update root and other related nodes' range
 3: end if
 4: for each node $n$ in $root.ptrs$ do
 5:     if $M^k \subseteq n.range$ then
 6:         InsertInSubLeaf($M^k$, root, $d$)
 7:     else if $M^k \cap n.range \neq \varnothing$ then
 8:         $M^k \rightarrow M^{k_1}, ..., M^{k_n}, M^{k_{n+1}}$
 9:         InsertInSubLeaf($M^{k_i}$, root, $d$)
10:     else
11:         n $\rightarrow$ n.ptrs
12:     end if
13: end for
```

```
Algorithm 3 InsertInSubLeaf
Require: $M^k$, sub-leaf $n$, $d$
Ensure: SKDTree
 1: if $n.ptrs < 2d$ then
 2:     $n.ptrs \rightarrow M^k$
 3: else
 4:     Build($M^k$, $n$, $n.dim$, $d$)
 5: end if
```

Deleting leaf nodes When the density of a leaf node in the SKDTree is less than the threshold Θ and there is no summary subtree to connect, the node should be removed, and the fresh data of the node should be restored to the original stream for subsequent re-clustering.

The algorithm DelLeafNode() removes leaf nodes with densities less than Θ. If all of its leaf nodes are deleted, the sub-leaf node is also deleted, and the space represented by the sub-leaf node becomes invalidated. The space represented by the node's parent is re-divided. Fig. 8 depicts the SKDTree corresponding to 2D space. Assume that after decaying, the density of the two leaf nodes of sub-leaf node A drop below Θ and the two leaf nodes must be removed directly, space A, as shown on the right of Fig. 8, become invalid and Build() algorithm will be called to rebuild the space represented by node C(the gray shaded part). Similarly, deleting all of the leaf nodes of sub-leaf

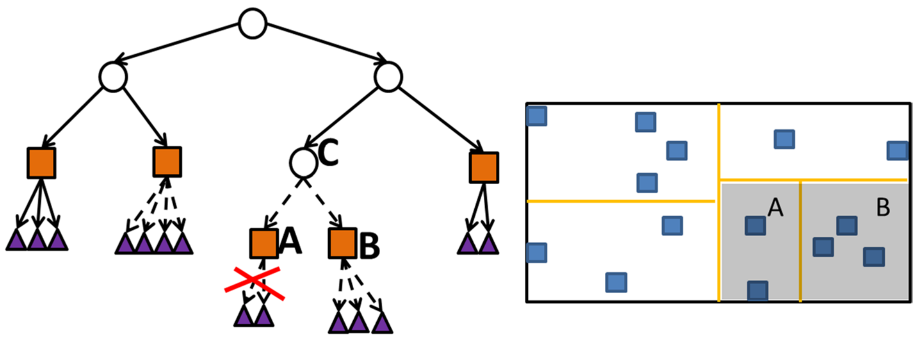


Fig. 8 The instance of deleting leaf nodes in 2D SKDTree. On the left side of figure 8 is an SKDTree corresponding to the data spaceon the right. In the SKDTree, non-leaf nodes are represented as hollow circles,sub-leaf nodes as orange squares, and leaf nodes as purple triangles. In the rightside, the gray shaded part are in line with non-leaf node C

nodes A and B results in the deletion of the parent node C. The corresponding space turns invalidated. A, B and C are removed. And the Build() is invoked to reconstruct the space represented by the parent node of C.

Algorithm 4 delLeafNode

```
Require: $M^k$, $\Theta$
Ensure: SKDTree
1: if $\rho_M < \Theta$ then
2:     update $\rho$ of $M^k$' parent node $S$
3:     if $S.ptrs \rightarrow \varnothing$ then
4:         Build($M^k$, $S.parent$, $S.parent.dim$, $d$)
5:     end if
6: end if
```

Summarizing the subtree From the beginning of SKDTree building, all leaf nodes are cluster cells that have met the density threshold Θ. Therefore, the process for summarizing the subtree looks for nodes other than leaf nodes that exceed the density threshold, summarizes the subtrees rooted at those nodes into a new cluster cell, and save it as external files, pruning the SKDTree into a simpler structure during building process in a certain extent. The Summarize() process recursively summarizes the nodes upwards until the density of a node is less than Θ, at which the fan-out operation is executed. As shown in Fig. 9, the densities of node N and children nodes A and B are all above the threshold Θ. As a result, the algorithm stores the subtree represented by N as a summary file into external storage, and summarizes N as a new cluster cell, that is, the new leaf node N in the SKDTree. The summarizing process also creates sub-leaf node N' and leaf node N'' for node N. The range of nodes N' and N'' is equal to the size of the cluster cell represented by node N.

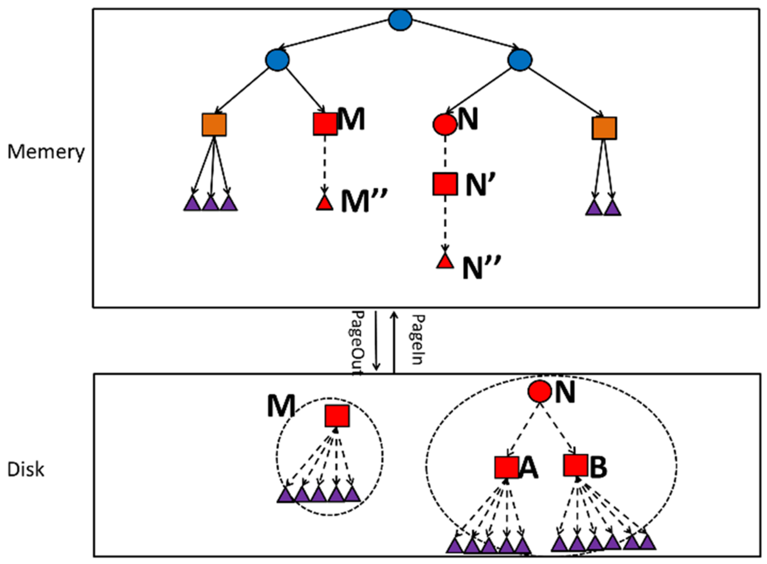


Fig. 9 The instance of pageIn and pageOut. It is a whole SKDTree in the memery, and there are multiple summarizedsubstrees in the disk

Algorithm 5 Summarize

```
Require: sub-leaf node $P$
Ensure: summary file, SKDTree
 1: if $P.\rho > \Theta$ then
 2:     $S \leftarrow$ the parent of $P$
 3:     if $S.\rho > \Theta$ then
 4:         $P \leftarrow S$
 5:         Summarize($P$)
 6:     else
 7:         PageOut the Subtree rooted at $S.ptrs$
 8:         if $S$ is sub-leaf node then
 9:             create a leaf node $S''$
10:             $S.ptrs \rightarrow S''$
11:         else if $S$ is non-leaf node then
12:             create a sub-leaf $S'$ and a leaf node $S''$
13:             $S.ptrs \rightarrow S'$, $S' \rightarrow S''$
14:         end if
15:     end if
16: end if
```

Re-inserting the substree Given a node P that has been summarized. When $P's$ density is below the threshold as data decay, it indicates that P is not a cluster cell at the moment. We need read back the corresponding summary subtree and update all nodes' density in SKDTree. Such as read the file that stores the subtree corresponding to node P in external memory and re-insert it into the SKDTree.

When the density of a leaf node in the SKDTree is less than the threshold Θ and there is a summary subtree that can be spliced, the summary subtree needs to be read back to the memory and spliced into the SKDTree. ReInsert() returns the summary subtree corresponding to the leaf node *P* to its parent node, that is, the sub-leaf node *P*′. At the same time, the density of each node in the newly inserted subtree must be updated and calculated. DelLeafNode() must be run if the density of leaf nodes in the subtree does not fulfill the density requirements. As shown in Fig. 9, when the density of node *M* in the SKDTree is less than Θ, read the corresponding *M* subtree from the external memory and put it back into the subtree of the sub-leaf node *M*′ in the SKDTree. After updating the density of the subtree node, there is a leaf node that fail to match the cluster cell density constraint, and it is destroyed directly, as depicted inside the dashed ellipse.

Algorithm 6 ReInsert

```
Require: $P$, root
Ensure: SKDTree
1: PageIn the subtree rooted at $P \leftarrow file_p$
2: the sub-leaf node $P' \leftarrow P.parent$
3: $P'.ptrs \rightarrow$ the subtree root $P$
4: update the $\rho$ of the descendant node of $P$
```

Query dynamic clusters When the user proposes to query the current clusters, QueryCluster() is called to identify all the reachable space with the summary leaf node *R* and obtain the data stream dynamic cluster. The QueryCluster() algorithm searches the reachable space of the summary cluster cells for adjacent cluster cells and labels them with the same cluster label. In the SKDTree, finding the maximum reachable space of a summary cluster cell requires backtracking from the leaf node to the $2d + 1$ level. With each layer backtracked, all leaf nodes in other child nodes are checked for nodes adjacent to the summary cluster. The space range represented by the summary node in d-dimensional space is adjacent to at most $2d$ hyperplanes, hence it is only necessary to search for reachable cluster cells in the range that contains and is adjacent to the cluster resolution space. And, since nodes in the SKDTree contains a relationship from top to bottom, there is no need to traverse the entire SKDTree, only $2d$ nodes need to be traced upwards from the leaf nodes of the summarization. In Sec.4.2.4, the summarizing process creates one sub-leaf node layer for each summary non-leaf node. To obtain the maximum reachable space of a solution node, the QueryCluster() algorithm needs to return to 2d+1 levels totally. In Fig. 10, it is the most special example of summarized sub-leaf node querying dynamic clusters in 2D space. The summarized cluster is represented by the rectangle denoted by the red solid line. To obtain a complete dynamic cluster, it is necessary to scan the reachable space in this case, going back up $2d$ times according to the marked serial number. Similarly, non-leaf nodes query dynamic clusters need to backtrack $2d + 1$ times, therefore this algorithm determines that query dynamic clusters backtrack $2d + 1$ levels.

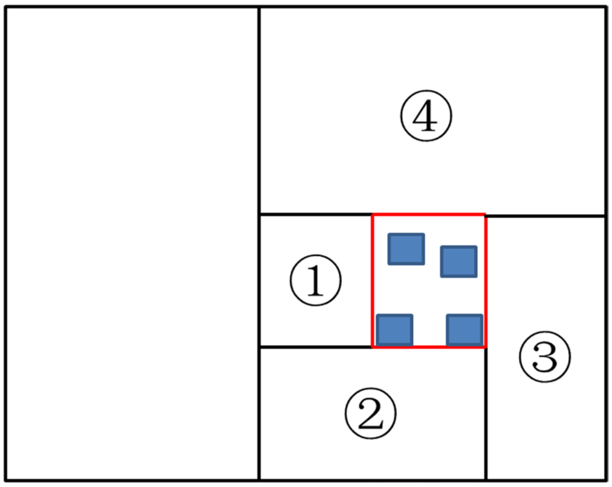


Fig. 10 The instance of query dynamic clusters. In the figure 10, the rectangle outlined in red represents a example of sum-marized clustering result

Algorithm 7 QueryCluster

```
Require: root, d, $\Theta$
Ensure: dynamic cluster $c_R$
 1: for each $R.\rho > \Theta$ in SKDTree do
 2:     InitStack(S), i=0, $r$ = root
 3:     while $r \neq R$ —— $i < 2d$ —— IsEmpty(S) do
 4:         if $r$ then
 5:             Push(S,$r$)
 6:             r = $p \leftarrow$ the node from $r$ to $R$
 7:         else
 8:             Pop(S, $r$), i++
 9:             for each node $q$ in the other child of $r$ do
10:                 if $R$ and $q$ satisfy Def.5 then
11:                     mark them as the $c_R$
12:                 end if
13:             end for
14:         end if
15:     end while
16: end for
```

3.3 SKDStream algorithms

The difference between SKDStream and other time-decaying data stream clustering algorithms is that it can automatically obtain the existing clustering clusters at the current time, adjust the clustering results in time and track the evolution of clustering.

Storage structures: SKDTree

SKDTree is a data structure for abstracting the data stream sample space. All nodes in the tree do not store the real data, but only the summary information of the data space. The nodes of each layer contain the subspace in turn according to the way of dimension alternation. Each leaf node is a cluster cell meeting the density requirements rather than a single point. The basic structure and related operations of SKDTree are described in Sec.4.

Initialization

Initially, the initial clusters that absorb the newly arrived points according to the OPTICS algorithm. Once the size of cached initial clusters exceeds the predefined value, the cluster cells satisfy the density are obtained. Through executing the Build() mentioned in Sec.4.2.1 to initialize the SKDTree. The tree is saved in memory to facilitate the subsequent update of the structure.

Key steps

(1) New data assignment. A new data from time-decaying stream is assigned to an existing cluster cell, so as to place straight into SKDTree leaf node, and update the density of relevant nodes. Otherwise the new data will be cached in memory.
(2) Cluster cell emergence (SKDTree AddLeafNode(), SKDTree Summarize()). Cluster cells can be generated in two ways: one is the cluster cells selected by preliminary clustering after a certain time of cached data in step (1). The other is according the step (3), there should be a larger cluster cell through merging the existing smaller cluster cell. The cluster cells satisfying the density threshold obtained from the time-decaying must be stored in the SKDTree leaf node.
(3) Cluster cell mergence (SKDTree Summarize(), SKDTree DelLeafNode()). With the entry of new data, timely-density of some existing cluster cells increases continuously, and so does the density of the subspace contain cluster cells recursively. When the density of a subspace reaches the density threshold, two or more existing small cluster cells in the subspace are combined into a larger cluster cell. In our algorithm, the merge of cluster cells is a recursive process. And the summary information of the previous small cluster cells is not explicitly stored in SKDTree.
(4) Cluster cell splitting (SKDTree ReInsert()). The timely-density of some cluster cells decays as the life weights of data is fading. When the density of cluster cell is lower than threshold, deleting invalid data in it and recover and retain the previous small cluster cells if still meet the density requirement. The above process completes the splitting of cluster cells. Otherwise, we will delete the cluster cells directly.
(5) Tracking the changes of clusters (SKDTree QueryCluster()). The steps (3) and (4) lead to the cluster evolution. We can track the maximum reachable space to obtain the timely clusters in time-decaying data stream.

Figure 11 shows the SKDStream overview in which the storage structures SKDTree are designed and the key steps to perform the algorithm are annotated.

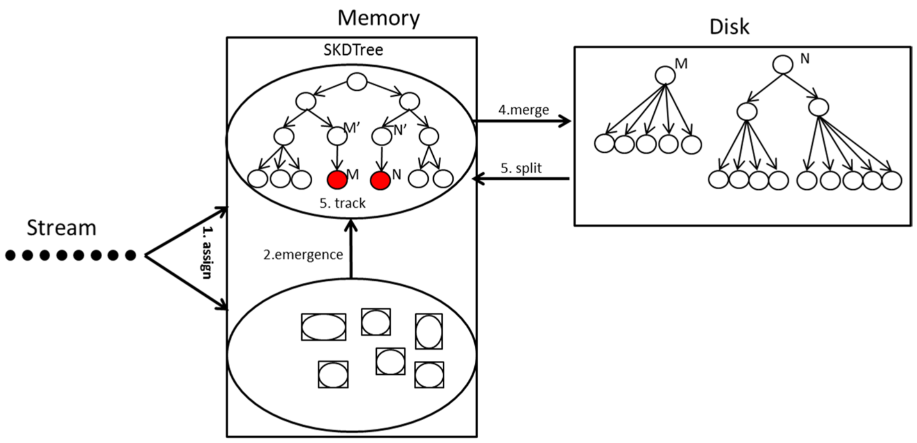


Fig. 11 SKDStream overview. The left represents the time-decaying data stream, the middle part is theSKDTree stored in the memery, and the right part are the summarized subtreesin the disk

Algorithm 8 SKDStream

```
Require: SKDTree, DS
Ensure: dynamic clusters
 1: for t do
 2:     add point $x_i$ in $X_{t_j}$ and $X_{t_j+t}$ to Buffer[]
 3:     for $x_i$ in Buffer[] do
 4:         if $x_i$ in $root.range$ then
 5:             insert $x_i$ in SKDTree, delete $x_i$ from Buffer[]
 6:         end if
 7:     end for
 8:     cluster cell $M_i^k(i > 0) \leftarrow$ OPTICS(Buffer[])
 9:     update the $f$ in Buffer[] and SKDTree
10:     delLeafNode(root, $\Theta$)
11:     Summarize(root,$\Theta$)
12:     ReInsert($P$, root, $file_p$)
13:     for $M^k$ in $M_i^k(i > 0)$ do
14:         addLeafNodes($M^k$,root)
15:         delete point in $M^k$ from Buffer[]
16:     end for
17:     Summarize(root,$\Theta$)
18: end for
19: QueryCluster(root, $d$, $\Theta$)
```

4 Results and discussion

We compare the SKDStream algorithm with EDMtream and D-Stream algorithm experimentally. All experimental results are done on a computer with 3.4GHz Intel Core i5CPU and 8GB RAM. The development environment is based on the Windows operating system and the programming language is Java.

Table 2 Parameter settings

Parameter name	Speed(v)	Life weights(λ)	Neighborhood radius(r)
Parameter value	1000pt/s	0.002	≤ 60

4.1 Preparations

Database The experimental datasets include a synthetic dataset and two real datasets CoverType and KDDCUP99. The synthetic data set is a collection of two-dimensional data generated by the article [21], named Syn data set by us. It contains 50,000 stream data. It is clear to describe the clusters and show the evolution by them. Both the public datasets are from UCI Machine Learning Repository. The CoverType is a 54-dimensional data set, containing 581012 data, and KDDCUP99 is a 34-dimensional data set, containing 494021 data.

Parameter setting In the experiment, unless in a specific experimental situation, the SKDStream algorithm sets the arrival rate to 1000pt/s. At the same time, in order to have the same decaying effect with other comparison algorithms, SKDStream set the decaying function the same to article [14], which requires the Def.2 life weights satisfies $2 - \lambda = 0.998$, so in the following we set λ to 0.002. Secondly, we use OPTICS algorithm to obtain the MBH in the second layer. There are two params including neighborhood radius r and the minimum number of core points neighborhoods *pointNum* in the OPTICS algorithm. Therefore, the two parameters are also become the dynamic parameters in SKDStream. But the effect of *pointNum* is minimal for the specific implementation process of the SKDStream. All experiments in the following article, the value of the radius is generally set to $r \leqslant 60$. The Sec.6.5 will discuss in detail the effect of r on the quality of clustering. Finally, the adjustment of dynamically density threshold Θ will also be discussed in subsequent chapters. Table 2 shows values of other parameters except the density threshold in SKDStream.

4.2 Tracking cluster evolution of data stream

The biggest difference between stream data and other algorithms is the ability to track the evolution of clustering. We use a two-dimensional synthetic data set to test the ability of tracking evolution process for SKDStream algorithm. The Syn data set contains 50,000 data. We set the arrival rate of stream data to 1000pt/s. It can be calculated that it takes 50s from the beginning to the end of the stream data. Fig. 12 is a snapshot of the real-time evolution of data stream clusters. Different data point color represent different cluster. The figure shows the evolution process of the number and location of the clusters. In order to describe the changing of clusters more clearly, Fig. 13 shows the evolution of clusters in concrete form. The horizontal axis represents the clusters life cycle of data stream and the vertical axis indicates the cluster name. The different line colors indicate the clusters that exist in the data stream at the moment. The color of the line represents the existence of the cluster corresponding to the same color of the data in Fig. 12. In addition, it has marked the process of merging, separating and disappearing clusters in the figure. For example, in the 32s, cluster 3 gradually disappears, after that, cluster 6 is separated into two clusters around 33s, and two new clusters named cluster 8 and cluster 9 appear.

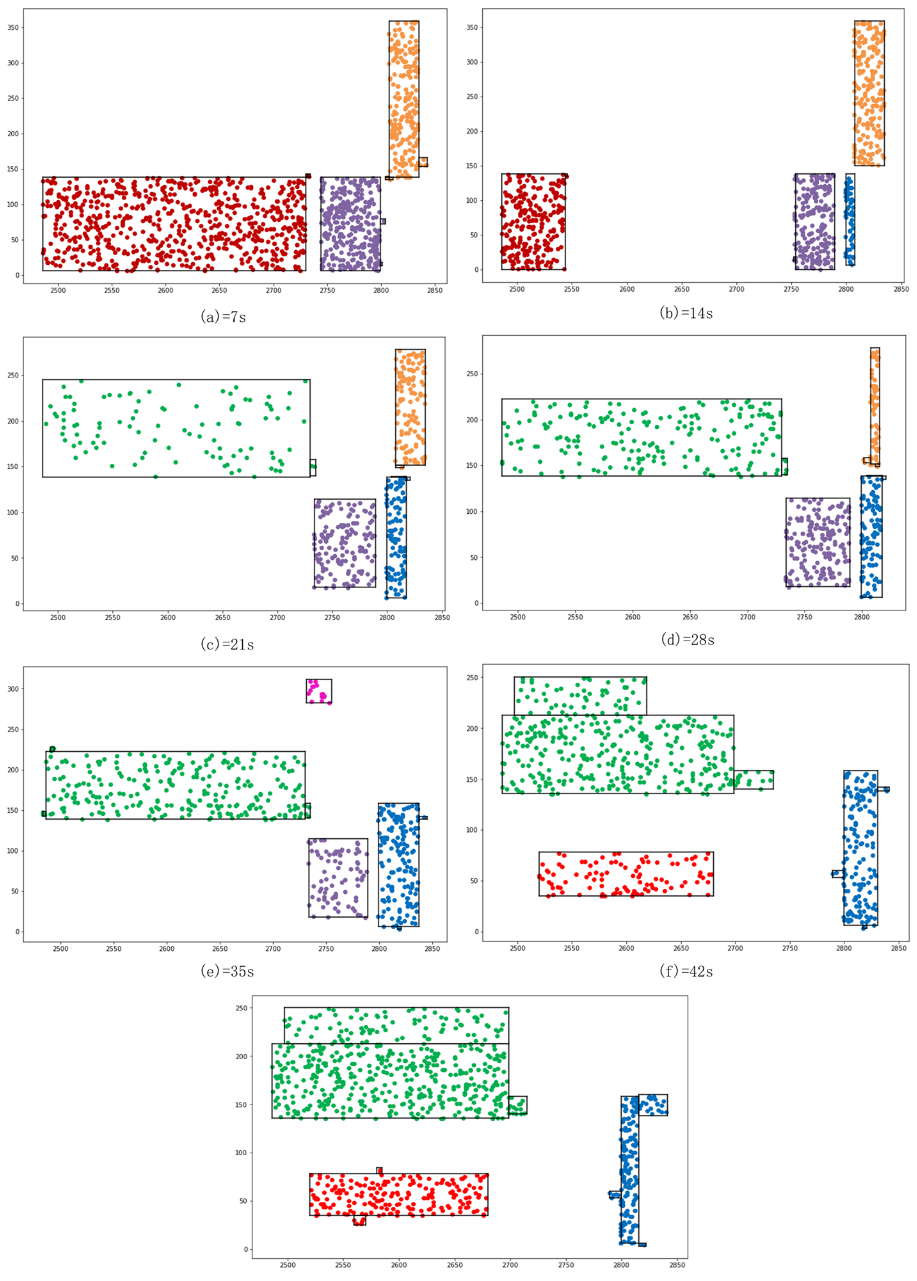


Fig. 12 The snapshot of data distribution clustering at different times. In the figure 12, each of picture represents a snapshot of clustering resultat a certain time. Data points of dierent color belong to dierent clusters

4.3 Response time comparison

We compare the response time of SKDStream algorithm with EDMStream and D-Stream. The three algorithms are tested and compared on two public data sets. The algorithm runs with a fixed rate on data stream. We record the response time every 25s. As shown in Figs. 14 and 15, the D-Stream cannot fully obtain the clustering results on the CoverType. The response time of SKDStream and EDMStream are shorter than

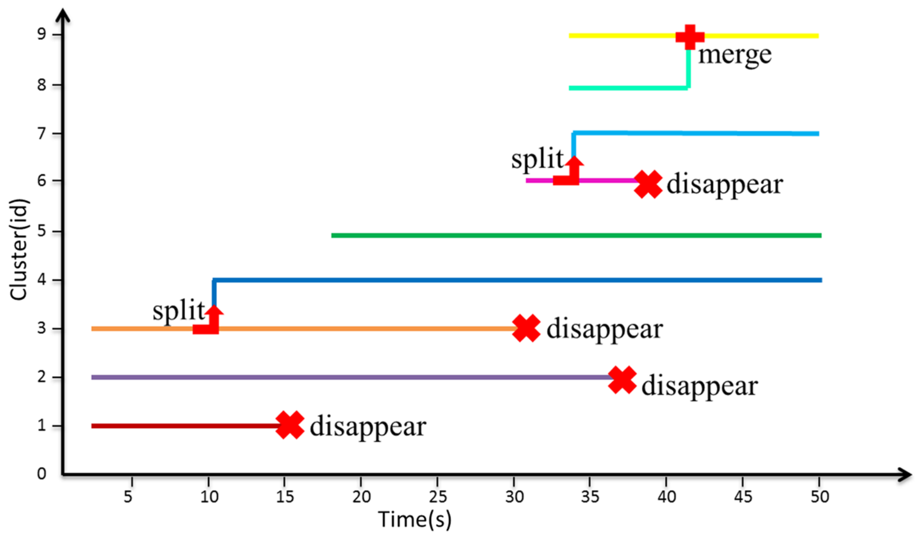


Fig. 13 The evolutionary process of clusters. In the figure 13, the horizontal axis represents the clusters life cycle andthe vertical axis indicates the cluster name. The different line colors indicatethe clusters that exist in the time-decaying data stream

D-Stream. Although the later response time of EDMStream is lower than SKDStream a little, however, the SKDStream is significantly better than the EDMStream in the initial response time. It shows the high effciency of the EDMStream algorithm at the initialization stage. In the early stage of EDMStream algorithm implementation, compared with other algorithms to calculate the distance between data, the outline structure of the MBH takes full advantage which can save time and cost. In the later stage, it takes more time in later time due to the re-insert of the subtree and the increase of the tree structure.

4.4 Cluster quality comparison

The focus of our experiment is to evaluate the clustering quality of the SKDStream algorithm on two real data sets. The quality comparison experiment puts up EDMStream and D-Stream algorithm as reference models, respectively. We calculate the validity index value of Jaccard coeffcient, FM index and Rand index related to test the clustering quality. These clustering index values are all distributed in the interval [0, 1]. The larger the value is, the better the cluster quality is. When the value is equal to 1, it means that the model to be tested is highly similar to reference model.

These clustering indicators [23] are calculated as follows:

$a = |SS|, SS = \{(x_i, x_j)|\lambda_i = \lambda_j, \lambda_i^* = \lambda_j^*, i < j\}$,

$b = |SD|, SD = \{(x_i, x_j)|\lambda_i = \lambda_j, \lambda_i^* \neq \lambda_j^*, i < j\}$,

$c = |DS|, DS = \{(x_i, x_j)|\lambda_i \neq \lambda_j, \lambda_i^* = \lambda_j^*, i < j\}$,

$d = |DD|, DD = \{(x_i, x_j)|\lambda_i \neq \lambda_j, \lambda_i^* \neq \lambda_j^*, i < j\}$,

That is, C is the result cluster division of current algorithm, C^* is the cluster division given by the reference model. λ and λ^* represent the cluster label vectors corresponding to C and C^*.

Jaccard Coefficient: $JC = \frac{a}{a+b+c}$

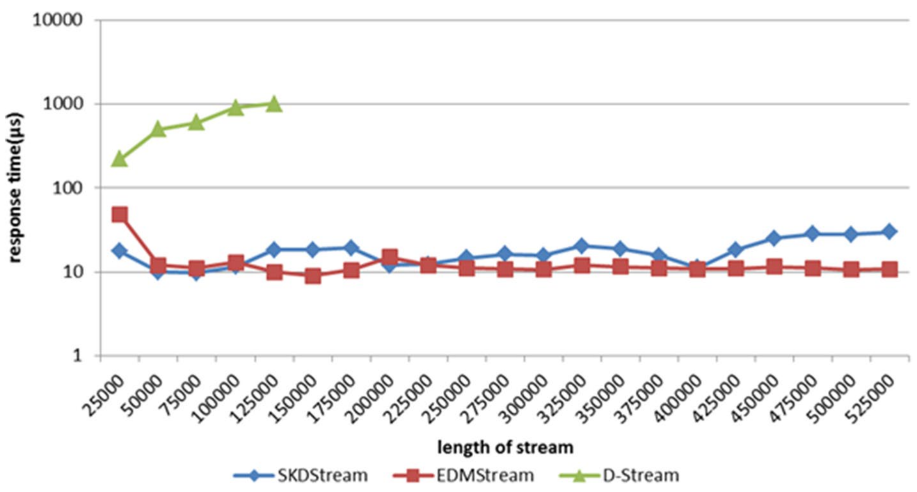


Fig. 14 Comparison of response time(CoverType). In the figure 14, the blue, red and green fold describe the response time ofSKDStream, EDMStream and D-Stream respectively in the CoverType dataset

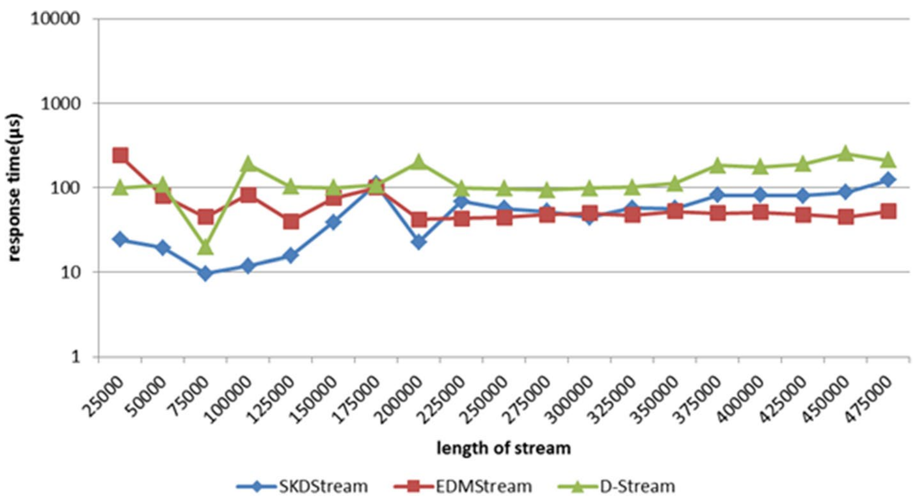


Fig. 15 Comparison of response time (KDDCUP99). In the figure 15, the blue, red and green fold described the response timeof SKDStream, EDMStream and D-Stream respectively in the KDDCUP99dataset

FM Index: $\sqrt{\frac{a}{a+b} * \frac{a}{a+c}}$

Rand Index: $RI = \frac{2(a+d)}{m(m-1)}$

As shown in Fig. 16, the index values are almost all around 0.9 when the EDMStream as a reference model. And with the passage of time, the cluster quality of clusters is relatively stable with little fluctuation. The reference model in Fig. 17 is D-Stream, effectiveness index values on the two data sets are slightly lower than EDMStream as reference model. In addtion, the indicator value fluctuates greatly. It can be seen that the clustering results of the SKDStream algorithm are more similar to EDMStream. In the article of EDMStream algorithm has been tested to show that the clustering quality of EDMStream is better than D-Stream. Therefore, the clustering quality of SKDStream is better than that of the D-Stream algorithm. The clustering quality of SKDStream is almost equivalent to EDMStream. It shows the effectiveness of the SKDStream algorithm.

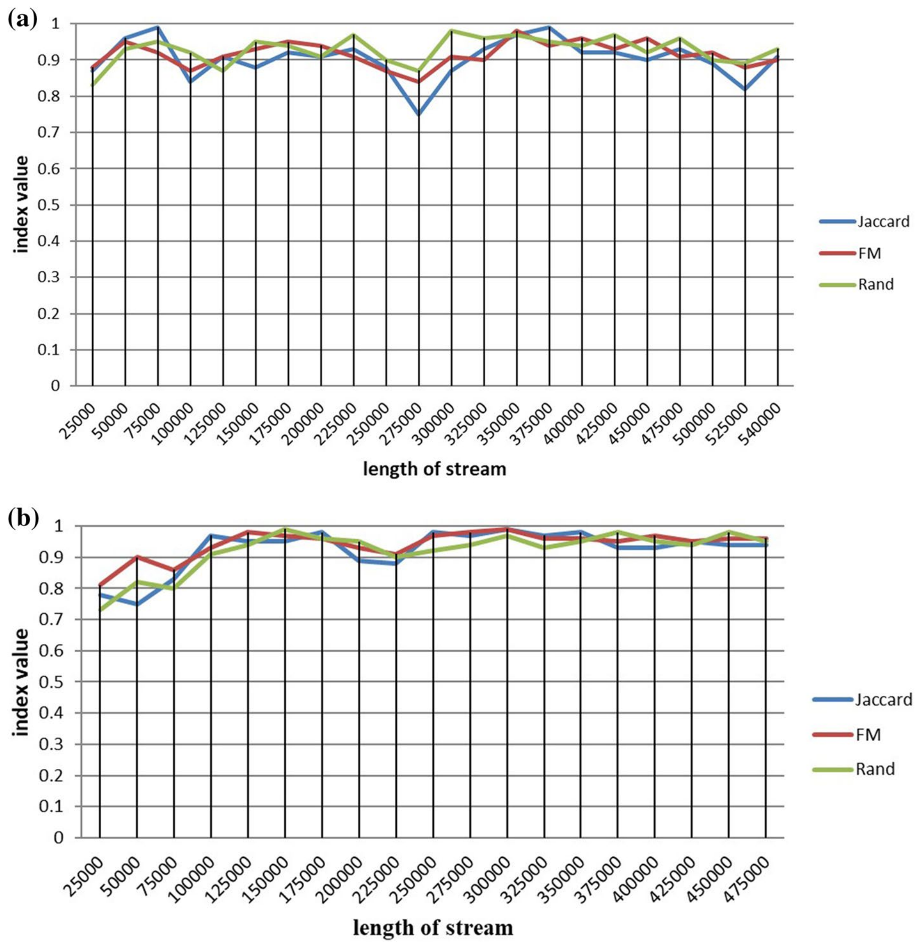


Fig. 16 Reference model: EDMStream. In the figure 16, the blue, red and green fold were respectively used to record Jaccard, FM and Rand indicators when reference model is EDMStream. **A** shows cluster quality in CoverType and **B** shows cluster quality in KDDCUP99

4.5 Parameters discussion and influence

The discussion about all the parameters in the SKDStream is based on the CoverType. The clustering effect is evaluated by the purity of clusters. Due to the arrival rate of data stream has a certain impact on the quality of clustering. We record the purity to evaluate cluster quality by changing the arrival rate. As shown in Fig. 18, we vary the stream data arrival rate (100pt/s, 1000pt/s, 5000pt/s, 10000pt/s, respectively). It shows that the clustering quality is relatively stable. In addition, the clustering quality in the case of a small rate is generally better than a high rate. In order to further verify the influence of the arrival rate for stream data on the SKDStream clustering result. We calculate the clustering quality with different rates on Covertype. As shown in Fig. 19, the arrival rate of data has little effect on clustering quality. Even when the arrival rate is large (> 5000t/s), the average purity can still reach 0.85.

The setting of the neighborhood radius r in the second layer of algorithm has a greater impact on the quality of the cluster. Fig. 20 shows the effect of neighborhood radius on the clustering purity in two-dimensional and three-dimensional space. It can be seen that the neighborhood radius cannot be too large. When radius is too

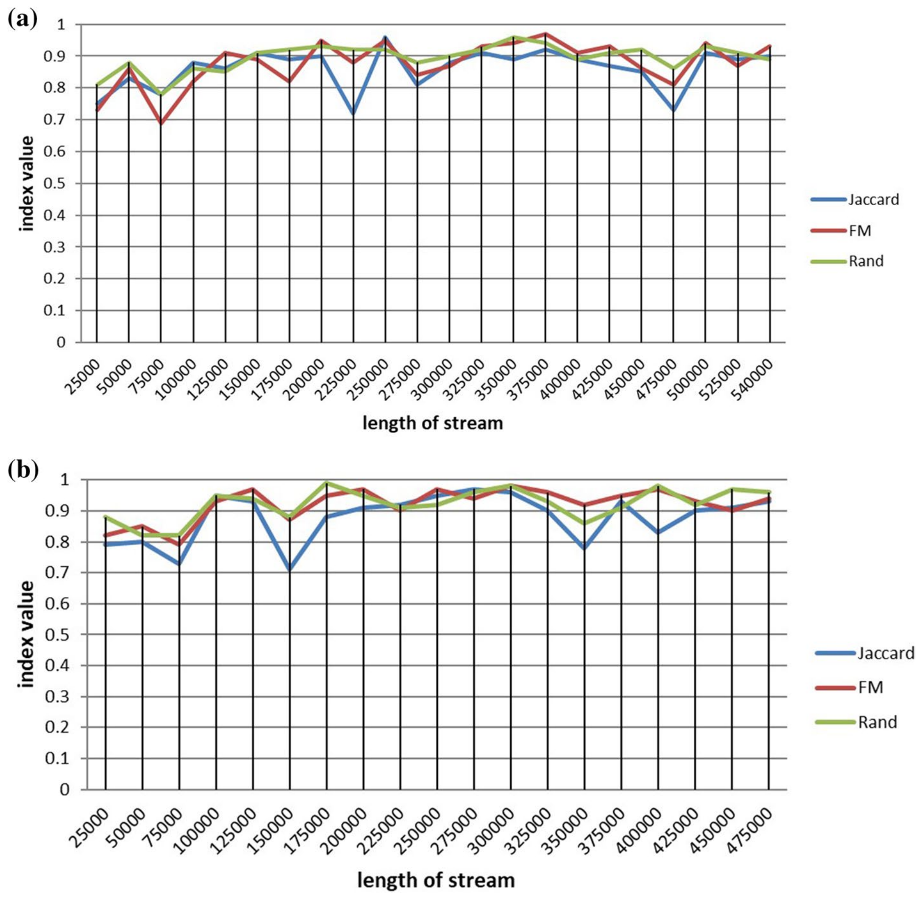


Fig. 17 Reference model: D-Stream. In the figure 17, the blue, red and green fold were respectively used to record Jaccard, FM and Rand indicators when reference model is D-Stream. **A** shows cluster quality in CoverType and **B** shows cluster quality in KDDCUP99

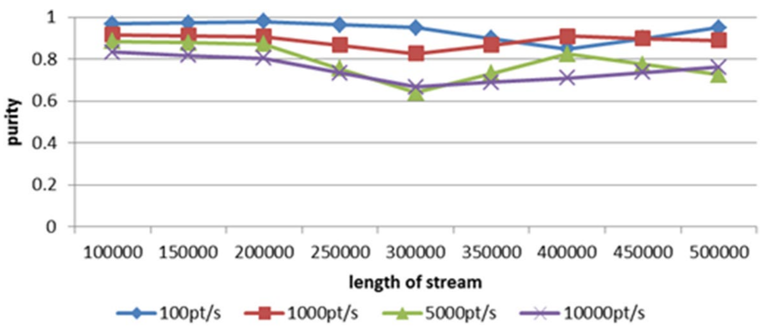


Fig. 18 Effect of stream data rate on purity. In the figure 18, different colors are used to indicate the influence of fourdifferent rates on the clustering purity

large, it will directly affect the quality of clustering. The neighborhood radius is about 15 and 50 in the two-dimensional space. The average purity is the highest at this moment. In the same way, the neighborhood radius in the three-dimensional space is about 20 and 35, If the parameter r is too large or too small, it will affect the quality of stream data clustering. The result can provide guidance for the setting of the neighborhood radius parameter in the experiment of this article.

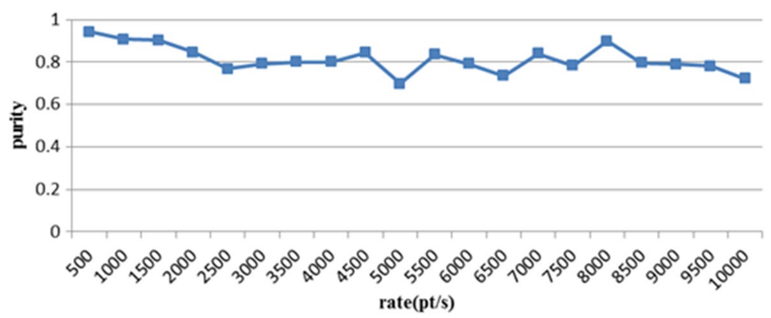
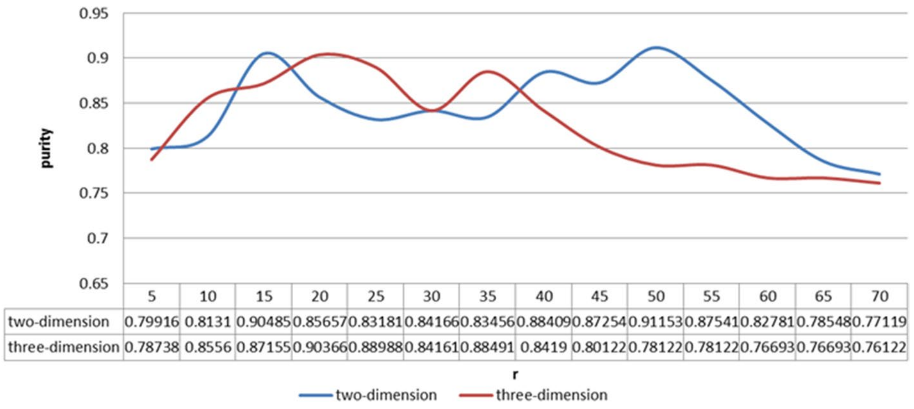


Fig. 19 Effect of stream data rate on purity over time. In the figure 19, the blue fold describes purity of clustering result at multipledifferent rates over time



	5	10	15	20	25	30	35	40	45	50	55	60	65	70
two-dimension	0.79916	0.8131	0.90485	0.85657	0.83181	0.84166	0.83456	0.88409	0.87254	0.91153	0.87541	0.82781	0.78548	0.77119
three-dimension	0.78738	0.8556	0.87155	0.90366	0.88988	0.84161	0.88491	0.8419	0.80122	0.78122	0.78122	0.76693	0.76693	0.76122

Fig. 20 Effect of r on clustering purity. The blue curve depicts the effect of radius changes on purity in two-dimensional space, while the red curve depicts the effect of radius changeson purity in three-dimensional space

In SKDStream algorithm, the density threshold Θ is a key parameter to capture clustering results. In real-life scenarios, the arrival rate of streaming data may be constantly changing. The dynamic adjustment of the density threshold is an indispensable part of the data stream clustering algorithm. The SKDStream has the ability of adjusting the density threshold to adapt to the evolution of data distribution. In order to better demonstrate the effectiveness of the dynamic density threshold, we use the synthesized data set Syn to compare the dynamic adaptive and static fixed density thresholds. The following table counts the number of nodes in the SKDTree tree. In order to compare the memory consumption cost at different density values, we adjust threshold value in a timely manner. It can ensure the number of nodes in the tree keeping relatively stable. It is obvious that the dynamic adjustment of the density threshold usually consumes lower memory cost than the static fixed value (Table 3).

5 Conclusion

We propose the SKDStream algorithm, a dynamic clustering algorithm on time-decaying data stream, which employs a hierarchical strategy to rapidly process dynamically decaying data. SKDStream defines MBH to abstract subspaces, thereby effectively representing summary information, and maintains clusters in real-time through the storage

Table 3 The numbers of node in SKDTree

Time	5	10	15	20	25	30	35	40	45	50
Static	18	23	46	58	81	68	74	56	50	49
Dynamic	18	29	35	41	52	44	63	48	53	46

structure SKDTree. We compare SKDStream with EDMStream and D-Stream in terms of response time and clustering quality, the final experimental results demonstrate the efficient and effective of clustering, as well as the ability of tracking the evolution of clusters, including number, center and density. In future, we hope that SKDStream algorithm can be improved to adapt to complex stream data, such as multiple stream data with mapping relationship and multi-view stream data.

Abbreviations

MBH	Minimum bounding hypercube

Author contributions
All authors reviewed and edited the manuscript. All authors read and approved the final manuscript.

Funding
This work is supported by the National Natural Science Foundation of China(61202022). Innovative Research Group Project of the National Natural Science Foundation of China.

Availability of data and materials
The experimental datasets generated and analyzed during the current study are from UCI Machine Learning Repository.

Declarations

Competing interests
The authors declare that they have no competing interests.

Received: 11 May 2022 Accepted: 15 August 2022
Published online: 15 October 2022

References
1. M.R. Ackermann, M Märtens, C. Raupach, K. Swierkot, C. Lammersen, C. Sohler, Streamkm++ a clustering algorithm for data streams. J. Exp. Algorithmics **17**, 1–2 (2012)
2. C. Aggarwal, S.Y. Philip, J. Han, J. Wang, A framework for clustering evolving data streams. In: Proceedings 2003 VLDB conference, pp. 81–92. Elsevier, (2003)
3. A. Amini, H. Saboohi, T. Herawan, T.Y. Wah, Mudi-stream: a multi density clustering algorithm for evolving data stream. J. Netw. Comput. Appl. **59**, 370–385 (2016)
4. A. Amini, T.Y. Wah, H. Saboohi, On density-based data streams clustering algorithms: a survey. J. Comput. Sci. Technol. **29**(1), 116–141 (2014)
5. J.L. Bentley, Multidimensional binary search trees used for associative searching. Commun. ACM **18**(9), 509–517 (1975)
6. A.M. Berg, S.T. Mol, G. Kismihók, N. Sclater, The role of a reference synthetic data generator within the field of learning analytics. J. Learn. Anal. **3**(1), 107–128 (2016)
7. C.G. Bezerra, B.S.J. Costa, L.A. Guedes, P.P. Angelov, An evolving approach to data streams clustering based on typicality and eccentricity data analytics. Inf. Sci. **518**, 13–28 (2020)
8. F. Cao, M. Estert, W. Qian, A. Zhou. Density-based clustering over an evolving data stream with noise. In: Proceedings of the 2006 SIAM international conference on data mining, pp. 328–339. SIAM, (2006)
9. Y. Chen, L. Tu. Density-based clustering for real-time stream data. In: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 133–142, (2007)
10. S. Chung, D. Mcleod, *Dynamic topic mining from news stream data* (Springer, Berlin, 2003)
11. M. Cui, D. Han, J. Wang, An efficient and safe road condition monitoring authentication scheme based on fog computing. IEEE Internet Things J. **PP**(99), 1–1 (2019)
12. M. Cui, D. Han, J. Wang, K.C. Li, C.C. Chang, Arfv: an efficient shared data auditing scheme supporting revocation for fog-assisted vehicular ad-hoc networks. IEEE Trans. Veh. Technol. **69**(12), 15815–15827 (2020)
13. J. de Andrade Silva, E.R. Hruschka, J. Gama, An evolutionary algorithm for clustering data streams with a variable number of clusters. Expert Syst. Appl. **67**, 228–238 (2017)

14. J. Gama, P. P. Rodrigues, L. Lopes, Clustering distributed sensor data streams using local processing and reduced communication. Intell. Data Anal. **15**(1), 3–28 (2011)
15. M. Ghesmoune, M. Lebbah, H. Azzag, State-of-the-art on clustering data streams. Big Data Anal. **1**(1), 1–27 (2016)
16. S. Gong, Y. Zhang, Y. Ge, Clustering stream data by exploring the evolution of density mountain. Proc. VLDB Endow. **11**(4), 393–405 (2017)
17. D. Han, N. Pan, K.C. Li, A traceable and revocable ciphertext-policy attribute-based encryption scheme based on privacy protection. IEEE Trans. Depend. Secure Comput. **PP**(99), 1–1 (2020)
18. D. Han, Y. Zhu, D. Li, W. Liang, A. Souri, K.C. Li. A blockchain-based auditable access control system for private data in service-centric iot environments. IEEE Transactions on Industrial Informatics, (2021)
19. R. Hyde, P. Angelov, A.R. MacKenzie, Fully online clustering of evolving data streams into arbitrarily shaped clusters. Inf. Sci. **382**, 96–114 (2017)
20. C. Isaksson, M.H. Dunham, M. Hahsler, Sostream: self organizing density-based clustering over data stream. In: International workshop on machine learning and data mining in pattern recognition, pp. 264–278. Springer, (2012)
21. M.K. Islam, M.M. Ahmed, K.Z. Zamli, A buffer-based online clustering for evolving data stream. Inf. Sci. **489**, 113–135 (2019)
22. P. Kumar, Data stream clustering in internet of things. SSRG Int. J. Comput. Sci. Eng. **3**(8), 1–14 (2016)
23. V. Kumar, J.K. Chhabra, D. Kumar, Initializing cluster center for k-means using biogeography based optimization. In: international conference on advances in computing, communication and control, pp. 448–456. Springer, (2011)
24. H. Li, D. Han, M. Tang, A privacy-preserving storage scheme for logistics data with assistance of blockchain. IEEE Internet of Things J (2021)
25. J. Ren, R. Ma, Density-based data streams clustering over sliding windows. In: 2009 Sixth international conference on fuzzy systems and knowledge discovery, pp. 248–252. IEEE, (2009)
26. J. Steil, M.X. Huang, A. Bulling, Fixation detection for head-mounted eye tracking based on visual similarity of gaze targets. In: Proceedings of the 2018 ACM Symposium on eye tracking research & applications, pp. 1–9, (2018)
27. Q. Tian, D. Han, K.C. Li, X. Liu, L. Duan, A. Castiglione, An intrusion detection approach based on improved deep belief network. Appl. Intell. **50**(10), 3162–3178 (2020)
28. K. Udommanetanakit, T. Rakthanmanon, K. Waiyamai, E-stream: evolution-based technique for stream clustering. In: International Conference on advanced data mining and applications, pp. 605–615. Springer, (2007)
29. X. Ji, G. Wang, T. Li, W. Deng, G. Gou, Fat node leading tree for data stream clustering with density peaks. Knowl. Based Syst. **120**, 99–117 (2017)
30. C. Yin, L. Xia, J. Wang, Application of an improved data stream clustering algorithm in intrusion detection system. In: advanced multimedia and ubiquitous engineering, pp. 626–632. Springer, (2017)

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Submit your manuscript to a SpringerOpen journal and benefit from:

- ► Convenient online submission
- ► Rigorous peer review
- ► Open access: articles freely available online
- ► High visibility within the field
- ► Retaining the copyright to your article

Submit your next manuscript at ► springeropen.com