Streaming Hierarchical Clustering Based on Point-Set Kernel

Xin Han
School of Computer Science
Xi'an Shiyou University
China
xhan@tulip.academy

Ye Zhu
School of Information Technology
Deakin University
Australia
ye.zhu@ieee.org

Kai Ming Ting
De-Chuan Zhan
State Key Laboratory of Novel Software Technology
Nanjing University
China
{tingkm,zhandc}@nju.edu.cn

Gang Li
Centre for Cyber Security Research and Innovation
Deakin University
Australia
gang.li@deakin.edu.au

ABSTRACT

Hierarchical clustering produces a cluster tree with different granularities. As a result, hierarchical clustering provides richer information and insight into a dataset than partitioning clustering. However, hierarchical clustering algorithms often have two weaknesses: scalability and the capacity to handle clusters of varying densities. This is because they rely on pairwise point-based similarity calculations and the similarity measure is independent of data distribution. In this paper, we aim to overcome these weaknesses and propose a novel efficient hierarchical clustering called `StreaKHC` that enables massive streaming data to be mined. The enabling factor is the use of a scalable point-set kernel to measure the similarity between an existing cluster in the cluster tree and a new point in the data stream. It also has an efficient mechanism to update the hierarchical structure so that a high-quality cluster tree can be maintained in real-time. Our extensive empirical evaluation shows that `StreaKHC` is more accurate and more efficient than existing hierarchical clustering algorithms.

CCS CONCEPTS

• **Computing methodologies** → **Cluster analysis**.

KEYWORDS

Hierarchical Clustering, Streaming Data, Isolation Kernel

ACM Reference Format:
Xin Han, Ye Zhu, Kai Ming Ting, De-Chuan Zhan, and Gang Li. 2022. Streaming Hierarchical Clustering Based on Point-Set Kernel. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22), August 14–18, 2022, Washington, DC, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3534678.3539323

KDD '22, August 14–18, 2022, Washington, DC, USA

ACM ISBN 978-1-4503-9385-0/22/08…$15.00
https://doi.org/10.1145/3534678.3539323

1 INTRODUCTION

Hierarchical clustering is one of the most popular clustering methods [2]. Different from the partitioning clustering that discovers a single partitioning of a given dataset, hierarchical clustering produces a cluster tree or dendrogram that organises sub-clusters in a binary tree such that each leaf contains a data point and each internal node represents a sub-cluster. Because it produces a hierarchy of clusters with different granularities and provides richer information and insight into the data, hierarchical clustering has been widely seen in various applications, such as social networks analysis [27], bioinformatics [5], and financial market analysis [36].

The commonly used hierarchical clustering approach is agglomerative hierarchical clustering (AHC) [14]. An AHC algorithm iteratively merges two most similar subsets as measured by a linkage function until all points are merged into a single cluster. It is well known that traditional AHC algorithms suffer from scalability and rigidity problems [17], i.e., they cannot handle massive datasets and the cluster tree structure produced could not be changed easily, since their time complexities are at least $O\left(n^2\right)$.

Unlike static/batch datasets, the data generated from various sensors in a data stream require real-time analysis. For example, remote sensing instruments used in meteorology and telecare systems produce a huge number of images, text, audio and video data streams. To mine large-scale streaming data, an algorithm must perform incremental processing in the stream, while addressing the time and memory limitations [8, 9, 13, 31, 39].

In order to meet the demands of real-time clustering, incremental methods construct an updatable cluster tree and continuously revise the tree in response to newly-arrived data points [16, 22]. To reduce the heavy load of pairwise distance calculations, existing incremental clustering approaches adopt various data reduction methods, e.g., sampling (CURE [10], StreamKM++ [1] & BICO [7]), mini-batch (MB-KM [29], SparseHC [24] & RP-DBSCAN [32]) and approximation (PERCH [16], GRINCH [22], URRH [26] & SCC [21]). These approaches trade off the clustering quality for fast linkage/similarity calculations.

For example, PERCH [16], a recent incremental hierarchical clustering algorithm, adds a new point with the node containing its nearest leaf node in the tree. It uses a bounding box approximation to perform the nearest neighbour search. In addition, PERCH uses

a masking-based rotation procedure to balance the tree in order to reduce the time of adding new points, since it usually has logarithmic time for search and updating in a balanced tree. A balanced tree is a severe constraint that usually does not match the structure of the data in the real world. Thus, the balanced tree will be a suboptimal hierarchical tree for the data. In a nutshell, PERCH achieves high efficiency at the cost of reduced accuracy because of the box approximation and the balanced-tree constraint.

In this paper, we propose a kernel-based incremental hierarchical clustering algorithm StreaKHC (**Strea**ming point-set **K**ernel-based **H**ierarchical **C**lustering) to efficiently mine massive streaming data. It is distinguished from existing incremental hierarchical clustering algorithms in three aspects.

First, StreaKHC utilises a top-down search strategy to add a new point with the most similar child node recursively until it reaches a leaf node. Thus, StreaKHC is very fast as it only traverses from the root node to a leaf of the cluster tree when adding a new point. In contrast, PERCH relies on a bottom-up strategy that searches the most similar leaf node, and then adds the new point with this node to grow the tree. Because PERCH needs to search all leaf nodes for every new point, it becomes slower as the tree grows larger.

Second, StreaKHC is designed based on a scalable point-set kernel [33]. It calculates the similarity between any node in a tree and each streaming point in $O(1)$. This is achieved without any sampling, mini-batch or approximation method, unlike existing incremental hierarchical clustering methods.

Third, StreaKHC has the ability to detect clusters with varied densities, in which most AHC algorithms have difficulty in separating. This is because the point-set kernel used in StreaKHC has a data-dependent characteristic that produces the similarity adaptive to local density. Therefore, StreaKHC maintains a high-quality cluster tree continuously in real-time in a data stream.

The contributions of this paper are:

- Proposing the first kernel-based incremental hierarchical clustering algorithm StreaKHC for clustering massive streaming data. We are the first to adopt the recent Isolation kernel [25, 33–35] to measure the similarity between a new point and a node of a cluster tree in constant time[1]. This enables a high-quality cluster tree to be produced.
- Developing an efficient tree updating strategy in real-time for StreaKHC. We utilise a top-down search strategy to efficiently add a new point with its most similar nodes in the tree. This updating strategy does not rely on any sampling, mini-batch or approximation method.
- Verifying the effectiveness and efficiency of StreaKHC on 17 synthetic and real-world datasets.[2] Our empirical results on these datasets confirm that StreaKHC is more accurate and runs faster than existing hierarchical clustering algorithms.

[1](a) Although Isolation kernel has been shown to significantly improve the clustering performance of existing AHC algorithms on datasets with clusters of varied densities, the scalability issue is still unresolved due to the expensive point-wise similarity calculation [11]. (b) Isolation-based point-set kernel clustering [33] and kernel-based clustering techniques have been proposed to address the limitation of distance measure to better capture the complex structure in a dataset [15, 25, 28] in batch mode. But up to our knowledge, no work has employed a kernel for incremental clustering.

[2]StreaKHC is implemented based on Python and can be obtained from https://github.com/tulip-lab/open-code/tree/master/StreaKHC.

The rest of the paper is organised as follows. We first describe the related work of hierarchical clustering in Section 2. Then, we introduce the Isolation kernel and point-set similarity measure before proposing StreaKHC for clustering data streams in Section 3. We present an extensive empirical evaluation in Section 4, followed by a conclusion in Section 5.

2 RELATED WORK

2.1 Hierarchical Clustering in Data Streams

Hierarchical clustering includes agglomerative (bottom-up) and divisive (top-down) methods [23], according to the direction in creating the cluster tree. A classical AHC algorithm sequentially combines individual points into sub-clusters and then adds them into larger clusters until all points end up being in the same cluster. It needs a linkage function that employs a similarity/distance measure to calculate the similarity of two sub-clusters [37]. Traditional linkage functions such as single-linkage and average-linkage have $O\left(n^2\right)$ and $O\left(n^2 \log n\right)$ time complexity on a dataset of n data points, respectively. Since they cannot scale to large datasets, several incremental algorithms have been proposed to overcome this weakness. Furthermore, classical AHC is not suited in the data streaming clustering setting because only limited memory space is available.

Many scalable clustering algorithms have been developed recently based on different data reduction methods. Partitioning clustering algorithms usually rely on sampling for scalability, such as StreamKM++ [1] and BICO [7]. Some clustering algorithms process data in mini-batches in an online manner for efficiency. For example, MB-KM [29] uses mini-batch optimisation for k-means clustering [19]. SparseHC [24] is an online hierarchical clustering method that compresses the distance matrix chunk-by-chunk for a fast cluster distance comparison.

In addition, there are other methods proposed based on various approximations. PERCH [16] is a new incremental hierarchical clustering algorithm. It adds a new point to the nearest (the most similar) leaf node of an existing cluster tree and then re-arrange/rotate the tree once detecting a masking situation, i.e., a point from a node that is closer to a point from another node. To reduce the runtime, it utilises several approximations in the algorithm. It relies on a bounding box approximation for fast nearest neighbour node detection. For each node, it records the lower and the upper bounds (the smallest and the highest value) of each dimension of all points in the node. Then the distance between a point to a node is the sum of the minimum squared gap between the attribute value of the point and the bounds of the node over each dimension. Furthermore, PERCH uses a balance-based rotation procedure to produce a balanced binary tree in order to control the depth of the tree.

Based on PERCH, GRINGE [22] was proposed recently with additional rotate, graft and restruct routines for tree rearrangement. Moreover, GRINGE can employ any linkage function.

Traditional AHC algorithms have high computational complexity as the time complexity of similarity calculation between a point and a node is $O(n)$, where n is the number of points in the node. Based on the latest point-set Kernel [33], all points in a node can be represented as a vector in terms of kernel mean map. Thus, the

similarity between a new point and a node can be computed using a dot product of two vectors in the kernel feature space with constant time $O(1)$. This unique property allows us to design an efficient incremental hierarchical clustering algorithm without any sampling, mini-batch or approximation method deployed in existing algorithms. In other words, an exact cluster tree can be built efficiently using point-set kernel from a dataset, without resorting to those methods that trade off accuracy for efficiency.

2.2 Isolation Kernel and Point-Set Similarity

Different kernel methods have been developed to improve the performance of existing distance-based clustering algorithms such as kernel k-means [28, 30], density-based clustering [12, 25], spectral clustering [4, 15, 38]. Particularly, the new data-dependent kernel, Isolation kernel [25, 35], has shown promising performance on density and distance-based classification and clustering problems. It also significantly improves the performance of existing AHC algorithms on data with clusters of varied densities [11]. In this paper, we further leverage the computational advantages of the Isolation kernel for designing a new incremental hierarchical clustering algorithm.

Isolation kernel [25, 35] has two important characteristics: (1) *two points in a sparse region are more similar than two points of equal inter-point distance in a dense region.* This characteristic enables the clustering algorithm to better separate complex clusters with varied densities; (2) it has *a finite-dimensional feature map with binary features* that enables a more efficient point-set similarity calculation with linear time complexity.

The key idea of the Isolation kernel is using a space partitioning strategy to split the whole data space into ψ non-overlapping partitions based on a random sample of ψ points from a given dataset. The similarity between any two points is how likely these two points can be split into the same partition.[3]

Let $D = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}, \mathbf{x}_i \in \mathbb{R}^d$ be a dataset sampled from an unknown probability density function $\mathbf{x}_i \sim F$. In addition, let $\mathbb{H}_\psi(D)$ denote the set of all partitionings H admissible for the given dataset D, where each H covers the entire space of $\mathbb{R}^d$; and each of the ψ isolating partitions $\theta[\mathbf{z}] \in H$ isolates one data point $\mathbf{z}$ from the rest of the points in a random subset $\mathcal{D} \subset D$, and $|\mathcal{D}| = \psi$. In our implementation, H is a partitioning generated from $\mathcal{D}$ [25]. The definition is shown as follows.

Definition 1. *For any two points $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$, Isolation kernel of $\mathbf{x}$ and $\mathbf{y}$ wrt D is defined to be the expectation taken over the probability distribution on all partitions $H \in \mathbb{H}_\psi(D)$ that both $\mathbf{x}$ and $\mathbf{y}$ fall into the same isolating partition $\theta[\mathbf{z}] \in H, \mathbf{z} \in \mathcal{D}$:*

$$\kappa_\psi(\mathbf{x}, \mathbf{y}|D) = \mathbb{E}_{\mathbb{H}_\psi(D)}[\mathbb{1}(\mathbf{x}, \mathbf{y} \in \theta[\mathbf{z}] \mid \theta[\mathbf{z}] \in H)] \quad (1)$$

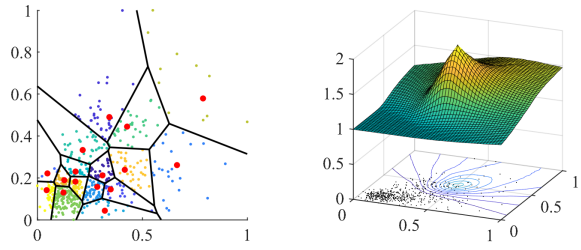
where $\mathbb{1}(\cdot)$ is an indicator function.

In practice, Isolation kernel κ_ψ is constructed using a finite number of partitionings $H_i, i = 1, \ldots, t$, where each H_i is created using $\mathcal{D}_i \subset D$:

$$\begin{aligned}\kappa_\psi(\mathbf{x}, \mathbf{y}|D) &= \frac{1}{t}\sum_{i=1}^{t} \mathbb{1}(\mathbf{x}, \mathbf{y} \in \theta \mid \theta \in H_i) \\ &= \frac{1}{t}\sum_{i=1}^{t}\sum_{\theta \in H_i} \mathbb{1}(\mathbf{x} \in \theta)\mathbb{1}(\mathbf{y} \in \theta) \quad (2)\end{aligned}$$

where θ is a shorthand for $\theta[\mathbf{z}]$, ψ is the sharpness parameter.

In this paper, we conduct the partitioning H using nearest neighbours [25], i.e., points in the data space sharing the same nearest neighbour from the subsample $\mathcal{D}$ are split into the same cell. Figure 1(a) illustrates a partitioning result on a real-word dataset with the subsample sizes $\psi = 16$, i.e., the data space are split into 16 cells based on the 16 subsample points, shown as red points.



(a) Sample partitioning **(b) Contour**

Figure 1: Demonstration of Isolation kernel ($\psi = 16$) on the WDBC dataset using the 4th vs 6th attributes: (a) An example partitioning H (b) Contours with reference to point (0.5, 0.5).

It is clear that the dense region is split into smaller cells than the sparse region. Thus, points in a dense region are more likely to be split into different cells and get a lower similarity score to each other. Thus, for points with equal inter-point distance from the reference point $\mathbf{x} = (0.5, 0.5)$, points in the spare region are more similar to $\mathbf{x}$ than points in the dense region to $\mathbf{x}$, as shown in Figure 1(b). Therefore, the Isolation kernel enables a distance or density-based clustering algorithm to detect complex clusters with varied densities [11, 25].

Definition 2. *Given a point $\mathbf{x}$ and a set $A = \{\mathbf{y}_i\}_{i=1}^{p}$, and $\mathbf{x}, \mathbf{y}_i \in \mathbb{R}^d$, the point-set similarity between $\mathbf{x}$ and A is the average pairwise similarity between $\mathbf{x}$ and every point in A, defined as follows:*

$$K_\psi(\mathbf{x}, A|D) = \frac{1}{|A|}\sum_{\mathbf{y} \in A} \kappa_\psi(\mathbf{x}, \mathbf{y}|D) \quad (3)$$

Isolation kernel has a finite feature map which is defined as follows [33]:

Definition 3. *For point $\mathbf{x} \in \mathbb{R}^d$, the feature mapping $\Phi : \mathbf{x} \to \{0, 1\}^{t\times\psi}$ of κ_ψ is a binary vector that represents the partitions in all the partitioning $H_i \in \mathbb{H}_\psi(D)$, $i = 1, \ldots, t$; where $\mathbf{x}$ falls into only one of ψ hyperspheres in each partitioning H_i.*

Based on the Isolation kernel feature map, we have $\kappa_\psi(\mathbf{x}, \mathbf{y}|D) = \frac{1}{t}\langle\Phi(\mathbf{x}), \Phi(\mathbf{y})\rangle$, and the point-set similarity [33] can be calculated

[3] For example, we can design a partitioning strategy to split a data space into 5 non-overlapping partitions, and independently conduct this partitioning strategy for 100 trials. If two points $\mathbf{x}$ and $\mathbf{y}$ are located in the same partition in 25 out of 100 trials, then the similarity between $\mathbf{x}$ and $\mathbf{y}$ is estimated as 0.25, since they have 25% probability to be split into the same partition.

as

$$K_\psi(\mathbf{x}, A|D) = \frac{1}{t|A|}\sum_{\mathbf{y}\in A}\langle\Phi(\mathbf{x}),\Phi(\mathbf{y})\rangle = \frac{1}{t|A|}\sum_{\mathbf{y}\in A}\Phi(\mathbf{x})^T\Phi(\mathbf{y}) = \frac{1}{t}\langle\Phi(\mathbf{x}),\widehat{\Phi}(A)\rangle \quad (4)$$

where $\widehat{\Phi}(A) = \frac{1}{|A|}\sum_{\mathbf{y}\in A}\Phi(\mathbf{y})$ is the kernel mean map of K_ψ.

Note that Equation 4 is a special case of Isolation Distribution Kernel [34]. Then we normalise it to $[0, 1]$ as follows:

$$\widehat{K}_\psi(\mathbf{x}, A|D) = \frac{\langle\Phi(\mathbf{x}),\widehat{\Phi}(A)\rangle}{\sqrt{\langle\Phi(\mathbf{x}),\Phi(\mathbf{x})\rangle}\sqrt{\langle\widehat{\Phi}(A),\widehat{\Phi}(A)\rangle}} \quad (5)$$

Because $\widehat{\Phi}(A)$ can be pre-calculated, estimating the similarity between a point and a set of $|D|$ points costs constant time $O(1)$. This provides a unique computational advantage for designing an incremental clustering algorithm. Also, recall that the feature mapping process of Isolation kernel has linear time complexity.

In contrast, commonly used kernels, such as Gaussian kernel and Laplacian kernel, have two key limitations: (1) their feature maps have intractable dimensionality; (2) their similarity is independent of a dataset, and thus insensitive to local data density.

3 KERNEL-BASED INCREMENTAL HIERARCHICAL CLUSTERING

In this section, we propose a novel clustering algorithm StreaKHC based on the scalable Isolation kernel, to address the problems of scalability and varied cluster density. It fully utilises the above-mentioned unique computational advantages of Isolation kernel for adding new points, i.e., using the point-set similarity to measure the similarity between a new point and a node in a hierarchical tree. Thus, StreaKHC has a linear time complexity that adds new points, without any mini-batch or approximation.

3.1 StreaKHC for Clustering Data Streams

Given a cluster tree and a new point $\mathbf{x}$, PERCH [16] relies on a greedy heuristic to update the tree via a bottom-up fashion. It first finds the most similar leaf node η containing the nearest neighbour of $\mathbf{x}$, disconnects η from its parent, and then creates a new node η' whose parent is η's former parent and with children η and a new node having $\mathbf{x}$. To increase efficiency, PERCH [16] uses a balance-based rotation procedure to control the depth of the tree. However, this heuristic is time-consuming as it still needs to search all leaf nodes for every new point from a stream.

To achieve a superfast updating process, StreaKHC employs a simple top-down search strategy to recursively add the new point $\mathbf{x}$ with $\mathbf{x}$'s most similar node at each level long the path from the root to a leaf node η. Then it replaces η with a subtree having two leaf nodes: η and its new sibling node containing $\mathbf{x}$. This search strategy (i) does not need to search all leaves since it only travels down one path from the root to a leaf of the tree, and the maximum search range is the height of the tree; and (ii) maintains a high intra-cluster similarity of the nodes in which the new point is added. This is because the new point only adds with the most similar child node at each level.

Algorithm 1: StreaKHC

Input : Data stream $\mathbb{D}$, subsample size ψ for IK
Output: Cluster tree $\mathcal{T}_S$

```
t = 300; # the default number of partitionings for IK.
l = 5000; # the default maximum number of leaf nodes
m = l; # the default dataset size used to build IK
Initialising Queue & S as empty sets;
Initialising an empty tree T_S;
D' ← first m points in the stream 𝔻;
Φ(·) ← Create.IKFeatureMap(D', ψ, t);
for x ← the latest point from the stream 𝔻 do
    y ← Φ(x);
    T_S ← GrowTree(T_S, y);
    S ← S ∪ {y};
    Queue.append(y);
    if |S| > l then
        o ← Queue.pop();
        T_S ← PruneTree(T_S, o);
        S ← S \ {o};
    end
end
```

StreaKHC which uses this strategy works both efficiently and effectively on most real-world datasets, and it does not need a rotation procedure to refine the tree structure. The details of StreaKHC are as follows.

Given a set of points $S = \{\mathbf{x}_1, \ldots, \mathbf{x}_l\}$, let $\mathcal{T}_S$ be a cluster tree over S as a binary tree with l leaf nodes, each containing a point $\mathbf{x}_i \in S$; and let S_η be the set of points at the root node η of $\mathcal{T}_S$.

StreaKHC generates a feature map of Isolation kernel (IK) with the first m points, and maps each point to its feature space before iteratively updating the hierarchical tree based on Algorithm 1.[4] Most parameters can be set to default values.[5]

3.1.1 Growing a Tree With a New Point. Algorithm 2 and Figure 2 show the process of growing a cluster tree $\mathcal{T}_S$ with a new point $\mathbf{x}$. It adds $\mathbf{x}$ to $\mathbf{x}$'s most similar node at each level and repeat the process at the next level until it reaches a leaf node η. Then, it replaces η with a subtree having two leaf nodes: η and its new sibling node containing $\mathbf{x}$.

3.1.2 Prune an Existing Leaf. We assume that a data stream is infinite, and a clustering system has limited memory. StreaKHC only retains the most recent data points to produce the cluster tree. Algorithm 3 is proposed to control the size of the hierarchical cluster tree, i.e., the number of leaves is limited to l.

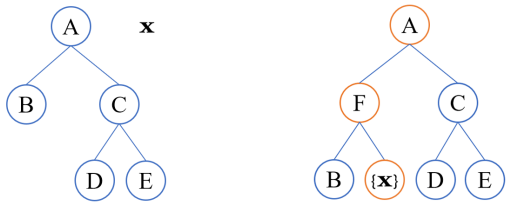
[4] We assume that the data points are uniformly distributed in a stream, such that the first m points are sufficient to represent the data distribution. The feature map $\Phi(\cdot)$ can be updated using the latest m points to adapt to the changes occurred in the steam, if necessary.

[5] The parameter t is the ensemble size which can be set to a large value such as 300. The higher the t, the more stable the kernel estimates but the longer the estimation time [25]. l is the tree size, which is set to the same value as in PERCH [16] and GRINCH [22]. m is the dataset size used to build Isolation kernel.

Algorithm 2: GrowTree

Input : Current cluster tree $\mathcal{T}_S$, new point $\mathbf{x}$
Output: Cluster tree $\mathcal{T}_{S\cup\{\mathbf{x}\}}$

$\eta \leftarrow \mathcal{T}_S.root$;
if $|S| > 0$ **then**
  while η *has children* **do**
    $S_\eta \leftarrow S_\eta \cup \{\mathbf{x}\}$;
    $\eta_l \leftarrow \eta.leftChild$;
    $\eta_r \leftarrow \eta.rightChild$;
    if $\widehat{K}_\psi(\mathbf{x}, S_{\eta_l}) \geq \widehat{K}_\psi(\mathbf{x}, S_{\eta_r})$ **then**
      $\eta \leftarrow \eta_l$;
    else
      $\eta \leftarrow \eta_r$;
    end
  end
  $S_{\eta^*} \leftarrow S_\eta \cup \{\mathbf{x}\}$;
  if $\eta.parent$ *is not nil* **then**
    $\eta^*.parent \leftarrow \eta.parent$;
  end
  $\eta^*.leftChild \leftarrow \eta$;
  $\eta^*.rightChild \leftarrow$ a new node containing $\{\mathbf{x}\}$;
end
Return $\mathcal{T}_{S\cup\{\mathbf{x}\}}$;



(a) Original tree and new point x **(b) Updated tree**

Figure 2: (a) An example cluster tree having sets $A, \ldots, E$ at the tree nodes and a new point x. (b) The updated tree when x is more similar to B than C. Since the node containing B has no child, it is replaced with a subtree having a new parent node containing $F = B \cup \{\mathbf{x}\}$ and the corresponding children nodes.

For simplicity, we use a queue structure to store the recent streaming data. Once adding a new point, the oldest point $\mathbf{o}$ on the tree will be removed. StreaKHC also utilises a top-down method to search $\mathbf{o}$ from all nodes, i.e., if a node contains $\mathbf{o}$, only its children will be searched, until reaching the leaf node containing $\mathbf{o}$.

3.2 Complexity Analysis of StreaKHC

StreaKHC has three key steps:

(1) Building an Isolation kernel and mapping points to its feature space. The former costs $O(\psi t)$, where $\psi \ll n$; and mapping n points costs $O(n\psi t)$.

Algorithm 3: PruneTree

Input : Current cluster tree $\mathcal{T}_S$, point $\mathbf{o}$
Output: Cluster tree $\mathcal{T}_{S\setminus\{\mathbf{o}\}}$

$\eta \leftarrow \mathcal{T}_S.root$;
if $|S| < 2$ **then**
  $\mathcal{T}_{S\setminus\{\mathbf{o}\}} \leftarrow Null$;
else
  while η *has children* **do**
    $S_\eta \leftarrow S_\eta \setminus \{\mathbf{o}\}$;
    if $\eta.leftChild$ *includes* $\mathbf{o}$ **then**
      $\eta \leftarrow \eta.leftChild$;
    else
      $\eta \leftarrow \eta.rightChild$;
    end
  end
  if $\eta.parent.parent$ *is not nil* **then**
    $\eta.sibling.parent \leftarrow \eta.parent.parent$;
  else
    Set $\eta.sibling$ as root;
  end
  Remove η and $\eta.parent$;
  Return the updated tree as $\mathcal{T}_{S\setminus\{\mathbf{o}\}}$;
end

(2) Adding points with an existing cluster tree. Once $\widehat{\Phi}(A)$ is pre-calculated[6], adding a point to a node containing A costs constant time because computing $\widehat{K}_\psi(\mathbf{x}, A)$ costs $O(\psi t)$. In the worst case, StreaKHC searches to the deepest of the tree along a traversal from the root node to a leaf. Therefore, the time complexity of adding n points costs $O(nh\psi t)$, where h is the height of the cluster tree.
(3) Removing points from a cluster tree. The complexity of removing one point is $O(h)$, and removing n points costs $O(nh)$.

In practice, we set $l, t, m, \psi \ll n$ as constants. Since the maximum height of a binary tree is always less than or equal to the leaf size l, h is a constant also. Therefore, the total time complexity of StreaKHC is linear as $O(n\psi t) + O(nh\psi t) + O(nh) = O(n)$.

Since StreaKHC stores the most recent l points only to maintain the hierarchical tree and ψt points for the Isolation kernel, its space complexity is thus $O(l\psi t)$. Note that each point in the IK feature space has ψt dimensions.

4 EMPIRICAL EVALUATION

4.1 Algorithms and Parameter Settings

We compared StreaKHC with the following six baseline hierarchical clustering algorithms[7]:

[6] Only the sum of the Isolation kernel features of all points in each node needs to be stored because the node size can be omitted in Equation 5.
[7] AHC algorithms were obtained from the `linkage` package of Matlab. The source codes of PHA and GRINCH were written in Matlab and Scala by their original authors, respectively. All experiments were run on a machine with 8 cores (Intel i7-11700K 3.60GHz) and 64GB memory.

1. Batch Hierarchical Clustering:

- AHC with average-linkage (Avg.). It repeatedly merges the two closest subclusters, which have the minimum *average* distance between two sets of points in the two subclusters, to form a larger subcluster.
- AHC with single-linkage (Sing.). It repeatedly merges the two closest subclusters, which have the *minimum* distance between a pair of points in the two subclusters, to form a larger subcluster.
- AHC with complete-linkage (Comp.). It repeatedly merges the two closest subclusters, which have the *maximum* distance between a pair of points in the two subclusters, to form a larger subcluster.
- PHA [20]. It uses a linkage function that measures the distance between two characteristic points from two clusters, where the characteristic point in a cluster is selected based on a hypothetical potential field.

2. Online Hierarchical Clustering:

- PERCH [16]. It creates the hierarchical tree one point at a time by adding points to the node having its nearest neighbour, and performing local tree re-arrangements in the form of rotations to produce a more balanced tree for efficiently updating new points.
- GRINCH [22]. It is similar to PERCH with an additional grafting subroutine that allows for more global rearrangements of the tree structure. We use two versions of GRINCH provided in the paper [22]: approximate average-linkage (GRINCH-A) and cosine similarity linkage (GRINCH-C).

Note that there is no parameter required to generate a hierarchical tree for all traditional AHC algorithms and PHA. PERCH and GRINCH have one parameter l that determines the maximum number of leaf nodes of the tree; and it is set to $l = 5000$.

`StreaKHC` employs the Isolation kernel. We search the best $\psi \in \{3, 5, 7, 13, 15, 17, 21, 25\}$ for Isolation kernel and fix all other parameters as default values. We set $m = n/4$ if a dataset has the number of points n less than $l = 5,000$ in `StreaKHC`.

4.2 Datasets

All algorithms are evaluated on 17 datasets as shown in the first four columns of Table 1. All real-world datasets are collected from *UCI Machine Learning Repository* [6] with different data sizes and dimensions, except ImageNet-10, STL-10 and CIFAR-10 from TensorFlow datasets[8]. To demonstrate the ability of a clustering algorithm on detecting clusters of varied densities, There are two 2-dimensional synthetic datasets with 4 Gaussian clusters. Synthetic-1 and Synthetic-2 are generated from the same probability density distribution function but with different samples. Figure 3 visualises the clusters in Synthetic-1 dataset.

When evaluating the clustering quality of online hierarchical clustering, we report the average result of five/ten randomly shuffled data streams for datasets having points higher/less than 60,000 points.[9] Note that AHC algorithms, PHA and GRINCH cannot scale to massive datasets. We only report the clustering results that can be obtained in a reasonable time on our machine.

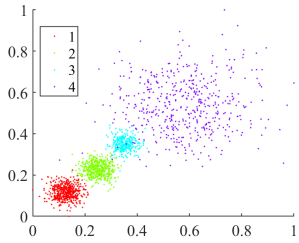


Figure 3: A synthetic 2-dimensional dataset with 4 Gaussian clusters of varied densities.

4.3 Evaluation Measure

Here we follow [16, 22] and adopt *Dendrogram Purity* to evaluate the quality of a dendrogram (as a hierarchical tree of clusters) produced by each hierarchical clustering algorithm. For any two leaf nodes having the same ground-truth cluster label, *Dendrogram Purity* finds the smallest subtree containing them and measures the fraction of leaf nodes in that subtree which are from the same cluster. The reported *Dendrogram Purity* is the average of the fractions of all unique pairs of different leaf nodes from the same cluster. The *Dendrogram Purity* is 1 if and only if all leaf nodes belonging to the same cluster are rooted in the same subtree, such that each cluster can be perfectly extracted by a local cut on the dendrogram. The *Dendrogram Purity* is calculated as follows.

Given a dendrogram $\mathcal{T}$ produced by a hierarchical clustering algorithm from a dataset $D = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$. Let $\Lambda_i, i = 1, \ldots, w$ be the true labels of w clusters, and $\mathcal{P} = \{(\mathbf{x}, \mathbf{x}') \,|\mathbf{x} \neq \mathbf{x}' \in D, \ell(\mathbf{x}) = \ell(\mathbf{x}')\}$ be the set of pairs of different points that have the same ground-truth cluster label, where $\ell(\mathbf{x})$ denotes the true cluster label of $\mathbf{x}$. Formally, the *Dendrogram Purity* of $\mathcal{T}$ is defined as

$$\text{Purity}(\mathcal{T}) = \frac{1}{|\mathcal{P}|} \sum_{i=1}^{w} \sum_{(\mathbf{x},\mathbf{x}') \in \mathcal{P}, \ell(\mathbf{x})=\Lambda_i} \mathfrak{f}\left(\mathfrak{g}\left(\mathfrak{h}\left(\mathbf{x}, \mathbf{x}'\right)\right), \Lambda_i\right)$$

where $\mathfrak{h}(\mathbf{x}, \mathbf{x}')$ is the least common ancestor of $\mathbf{x}$ and $\mathbf{x}'$ in $\mathcal{T}$, $\mathfrak{g}(\eta) \subset D$ is the set of points in all the descendant leaf nodes of η in $\mathcal{T}$, and $\mathfrak{f}(S, \Lambda_i) = \frac{|\{\mathbf{x} \in S \mid \ell(\mathbf{x})=\Lambda_i\}|}{S}$ computes the fraction of S that matches the ground-truth label Λ_i.

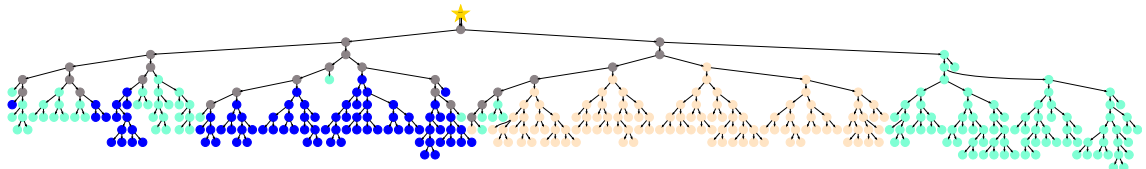
In this paper, we only used the *Dendrogram Purity* for clustering quality evaluation. When a cluster tree has a higher *Dendrogram Purity* score, the clusters extracted by a good strategy are expected to be purer. However, there is no uniform guide to identify the best and optimal strategy for subtree extraction. For a fair comparison, we only measure the quality of the dendrogram rather than the clustering results based on different cluster extraction strategies.

[8] We utilised a recent unsupervised deep learning method, i.e., instance-level contrastive head from Contrastive Clustering [18], to extract the learned representation from these three images as the input for all clustering algorithms in Table 1.

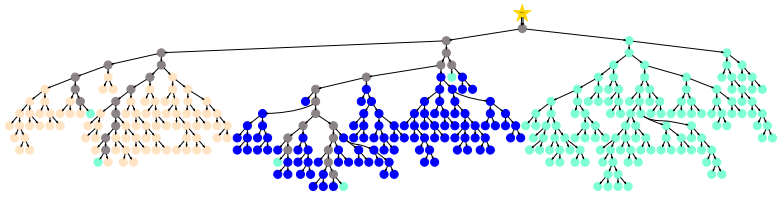
[9] To shuffle a dataset, we randomly permuted the dataset and streamed it for every algorithm in the same pre-processed order.

Table 1: Clustering results in *Dendrogram Purity*. The best 2 performers on each dataset are boldfaced. GRINCH-A and GRINCH-C use approximate average-linkage and cosine similarity linkage, respectively. PERCH-IK is the same PERCH algorithm running on the transformed dataset mapped by the Isolation kernel feature map. `StreamKHC-D` is `StreaKHC` using the average-linkage with Euclidean distance. "-" means that we cannot get the result in one day or with 64 GB RAM.

Dataset	#Points	#Dim.	#Clus.	Avg.	Sing.	Comp.	PHA	PERCH	GRINCH-C	GRINCH-A	StreaKHC	StreaKHC-D	PERCH-IK
ALLAML	72	7,129	2	.60	.68	.67	.68	.61 ± .04	.67 ± .03	.67 ± .01	**.69 ± .03**	.68 ± .05	**.72 ± .04**
LSVT	126	310	2	.63	.58	.62	.59	**.65 ± .03**	.61 ± .01	.62 ± .01	**.65 ± .01**	.61 ± .01	**.67 ± .02**
Wine	178	13	3	.89	.68	**.92**	.73	.72 ± .09	.80 ± .06	.88 ± .04	**.91 ± .03**	.85 ± .02	.87 ± .03
Seeds	210	7	3	**.85**	.69	.75	**.84**	.69 ± .08	.68 ± .04	.79 ± .04	.83 ± .01	.81 ± .01	.79 ± .04
Musk	476	166	2	.55	.54	**.56**	.54	.55 ± .01	.55 ± .01	.55 ± .00	.55 ± .01	.54 ± .01	**.57 ± .01**
WDBC	569	30	2	**.86**	.71	.79	.73	.71 ± .05	.64 ± .02	.83 ± .05	**.89 ± .01**	.83 ± .02	.82 ± .05
LandCover	675	147	9	**.56**	.30	.52	.44	.42 ± .03	.43 ± .04	.50 ± .03	**.55 ± .03**	.45 ± .02	.48 ± .02
Hill	1,212	100	2	.50	.50	.50	.50	.50 ± .00	**.57 ± .01**	.50 ± .00	**.51 ± .00**	.50 ± .00	**.51 ± .00**
Banknote	1,372	4	2	.68	**.92**	.63	.62	.66 ± .04	.63 ± .03	.71 ± .05	**.80 ± .05**	.63 ± .01	.77 ± .07
Synthetic-1	1,800	2	4	**.95**	.77	.88	.86	.78 ± .07	.28 ± .00	.87 ± .05	**.95 ± .00**	.89 ± .03	.94 ± .02
Spam	4,601	57	2	.58	.59	.57	.55	.57 ± .01	.56 ± .02	.58 ± .01	**.68 ± .01**	.62 ± .02	**.63 ± .02**
ImageNet-10	13,000	128	10	**.87**	.77	.27	.69	.67 ± .06	.70 ± .04	.71 ± .02	**.86 ± .01**	.76 ± .03	.70 ± .01
STL-10	13,000	128	10	**.62**	.50	.22	.53	.41 ± .04	.41 ± .02	.42 ± .01	**.61 ± .01**	.54 ± .02	.42 ± .02
CIFAR-10	60,000	128	10	-	-	-	-	.42 ± .02	.43 ± .03	.45 ± .03	**.68 ± .01**	**.61 ± .03**	.45 ± .02
Mnist	70,000	128	10	-	-	-	-	.20 ± .00	.23 ± .02	.24 ± .02	**.41 ± .01**	**.36 ± .01**	.32 ± .02
CoverType	581,012	54	7	-	-	-	-	**.45 ± .00**	.43 ± .00	.43 ± .00	.43 ± .01	.41 ± .01	**.46 ± .03**
Synthetic-2	1,800,000	2	4	-	-	-	-	**.73 ± .02**	-	-	**.81 ± .03**	-	-
Average of the first 13 datasets				.70	.63	.61	.64	.61	.58	.66	.73	.67	.68



(a) PERCH with *Dendrogram Purity* of 0.80.



(b) `StreaKHC` with *Dendrogram Purity* of 0.95.

Figure 4: Clustering dendrograms (cluster trees) on the Wine dataset. Each leaf node contains a data point, and each colour represents a ground-truth label of a node if all the points in it belong to the same label, otherwise, the node is coloured grey.

4.4 Performance Comparison

Clustering Outcomes. Table 1 shows the clustering quality of ten algorithms. It shows that `StreaKHC` outperforms existing algorithms on most datasets. The differences in *dendrogram purities* are small on other datasets. StreaKHC achieved the highest average *Dendrogram Purity* score over the 13 small datasets.

It is worth noting that on the Synthetic-1 dataset having clusters of varied densities, `StreaKHC` performs the best while GRINCH-C, single-linkage and Perch perform the worst. A similar improvement happens on the Wine and WDBC datasets, which have been shown to contain clusters of different densities [40, 41].

To visually compare the clustering results, Figure 4 shows the dendrograms generated by `StreaKHC` and PERCH on the Wine dataset. Recall that the tree generated by PERCH is based on balance-rotation. Since there are three clusters with different sizes, PERCH has to split the largest cluster and merged its cluster members with other smaller clusters in order to obtain a balanced tree, i.e., the cyan cluster members are merged with blue and pink clusters in Figure 4(a). Therefore, the balanced tree built by PERCH gets a much lower purity score than that of the subtrees built by `StreaKHC`. It is worth mentioning that `StreaKHC` has incorrectly added only 5 cyan points into subtrees of different clusters in Figure 4(b).

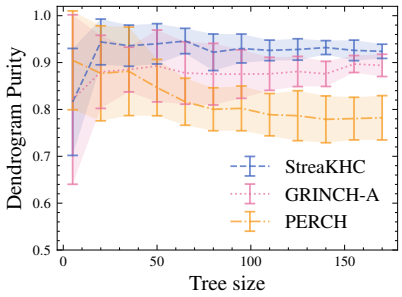


Figure 5: Dendrogram purity wrt tree size. The results are based on ten randomly shuffled data streams.

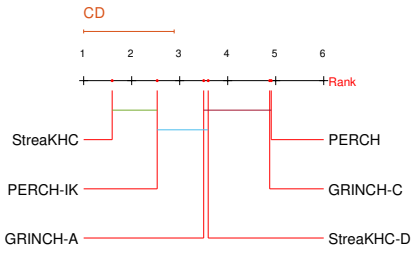


Figure 6: Critical difference (CD) diagram of the post-hoc Nemenyi test ($\alpha = 0.10$) for the results shown in Table 1. Two algorithms are not significantly different if there is a line linking them.

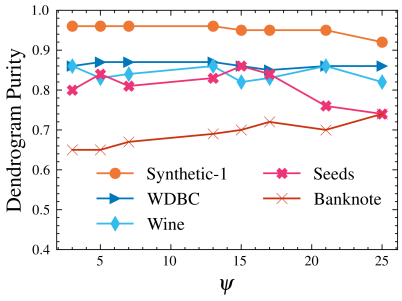


Figure 7: Parameter sensitivity analysis results of StreaKHC.

Figure 5 presents the changes to the dendrogram purity scores of the three clustering algorithms during the tree expanding stage on the Wine dataset. Both StreaKHC and GRINCH-A maintain a stable dendrogram purity score after obtaining 20 points, while PERCH gradually lowers its scores.

In addition, because PERCH utilises an approximation in similarity calculations and performs subtree rotation when adding data incrementally, its clustering quality has been degraded to the level of the classic AHC algorithm with a complete-linkage function (see their results in the last row in Table 1.)

We conducted a Friedman test with the post-hoc Nemenyi test [3] to examine whether the difference in clustering quality of any two algorithms is significant on all datasets except Synthetic-2. We first ranked all algorithms based on their *Dendrogram Purity* on each dataset, where the best one ranks 1 and so on. Two clustering algorithms are significantly different if the difference in their average ranks is larger than the critical difference (CD).

Figure 6 shows that StreaKHC performs significantly better than GRINCH-A, StreachKHC-D, PERCH and GRINCH-C. Note that PERCH is ranked last, and it is significantly worse than the two top-ranked algorithms. GRINCH [22] was an improvement over PERCH; but our result shows that there is no significant difference between GRINCH-A/C and PERCH.

Parameter Sensitivity. Here we examine the effect of the parameter ψ in StreaKHC. Figure 7 reports the *Dendrogram Purity* results of StreaKHC on six datasets using different ψ values. It shows that StreaKHC maintains a stable clustering quality within a certain parameter range. In practice, ψ could be adjusted appropriately around 15 to achieve good clustering quality on most of our evaluation datasets.

Runtime Comparison. All traditional AHC algorithms have at least time complexity of $O(n^2)$, thus, they cannot handle massive datasets. Figure 8 shows the scalability of 5 incremental algorithms. StreaKHC is the fastest algorithm. Although both StreaKHC and PERCH have time complexity of $O(n)$ when the tree size is bounded, PERCH has higher runtime than StreaKHC as it scans all leaf nodes when adding a new point.

Ablation Studies. We conducted two ablation studies here. First, we evaluate the effect of not using the data-dependent Isolation

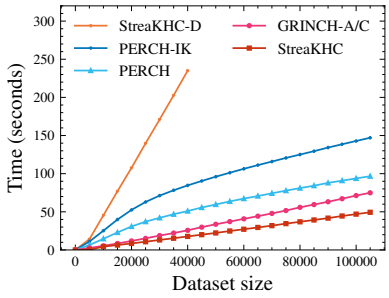


Figure 8: Run times of different incremental algorithms.

kernel [35] in StreaKHC. The average-linkage function is used instead of Equation 5 to measure the similarity between a new point and a node. This distance version StreaKHC-D has a significantly higher *Dendrogram Purity* score than PERCH and GRINCH-C on the Seeds, WDBC, Synthetic-1 and CIFAR-10 datasets, as shown in Table 1. However, StreaKHC-D runs much slower because of the expensive average distance calculation, as shown in Figure 8. This study shows the advantage of Isolation kernel (used in StreaKHC) over the distance-based measure (used in StreaKHC-D) in terms of both clustering outcome and runtime.

Second, we created PERCH-IK which is the same PERCH algorithm running on the dataset transformed by the feature map (as stated in Definition 3) of Isolation kernel. Table 1 shows that PERCH-IK performs comparably with StreaKHC. In addition, PERCH-IK is significantly better than PERCH and GRINCH-C, as shown in Figure 6. One key effect is that the Isolation kernel enables both StreaKHC and PERCH-IK to effectively identify clusters of varied densities on some datasets.

5 CONCLUSION

In this paper, we propose the first kernel-based incremental hierarchical clustering algorithm StreaKHC that mines potentially infinite streaming data efficiently and effectively. Our empirical results on a variety of real-world datasets show that StreaKHC is more accurate and runs faster than existing state-of-the-art online hierarchical clustering algorithms.

This is because StreaKHC differs from existing algorithms in three aspects. First, to add each new point, StreaKHC conducts its search in a top-down manner, avoiding searching all nodes in the cluster tree as required by existing methods. Second, it is designed based on the point-set kernel that has constant time for each similarity computation, StreaKHC updates the cluster tree very efficiently with each emerging new data point, and maintains a high-quality cluster tree in real-time. Finally, it utilises the data-dependent property of Isolation kernel to effectively detect clusters of varied densities in which most existing algorithms have difficulty separating.

In the future, we will investigate the capability of StreaKHC to deal with the concept drift (unforeseen change of distributions) in data streams.

ACKNOWLEDGMENTS

This project is supported by State Key Laboratory of Novel Software Technology, Nanjing University (Grant No. KFKT2019A32). Kai Ming Ting is supported by the National Natural Science Foundation of China (Grant No. 62076120).

REFERENCES

[1] Marcel R Ackermann, Marcus Märtens, Christoph Raupach, Kamil Swierkot, Christiane Lammersen, and Christian Sohler. 2012. Streamkm++: A clustering algorithm for data streams. *Journal of Experimental Algorithmics (JEA)* 17 (2012), 2–1.

[2] Vincent Cohen-Addad, Varun Kanade, Frederik Mallmann-Trenn, and Claire Mathieu. 2019. Hierarchical clustering: Objective functions and algorithms. *Journal of the ACM (JACM)* 66, 4 (2019), 26.

[3] Janez Demšar. 2006. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research* 7 (2006), 1–30.

[4] Inderjit S Dhillon, Yuqiang Guan, and Brian Kulis. 2004. Kernel k-means: spectral clustering and normalized cuts. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 551–556.

[5] Ibai Diez, Paolo Bonifazi, Iñaki Escudero, Beatriz Mateos, Miguel A Muñoz, Sebastiano Stramaglia, and Jesus M Cortes. 2015. A novel brain partition highlights the modular skeleton shared by structure and function. *Scientific reports* 5 (2015), 10532.

[6] Dheeru Dua and Casey Graff. 2017. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml

[7] Hendrik Fichtenberger, Marc Gillé, Melanie Schmidt, Chris Schwiegelshohn, and Christian Sohler. 2013. BICO: BIRCH meets coresets for k-means clustering. In *European Symposium on Algorithms*. Springer, 481–492.

[8] Mohamed Medhat Gaber, Arkady Zaslavsky, and Shonali Krishnaswamy. 2005. Mining data streams: a review. *ACM Sigmod Record* 34, 2 (2005), 18–26.

[9] Shufeng Gong, Yanfeng Zhang, and Ge Yu. 2017. Clustering Stream Data by Exploring the Evolution of Density Mountain. *Proc. VLDB Endow.* 11, 4 (dec 2017), 393–405. https://doi.org/10.1145/3164135.3164136

[10] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. 1998. CURE: An efficient clustering algorithm for large databases. *ACM Sigmod record* 27, 2 (1998), 73–84.

[11] Xin Han, Ye Zhu, Kai Ming Ting, and Gang Li. 2020. The Impact of Isolation Kernel on Agglomerative Hierarchical Clustering Algorithms. *arXiv preprint arXiv:2010.05473* (2020).

[12] Alexander Hinneburg and Hans-Henning Gabriel. 2007. Denclue 2.0: Fast clustering based on kernel density estimation. In *International symposium on intelligent data analysis*. Springer, 70–80.

[13] Ling Huang, Chang-Dong Wang, Hong-Yang Chao, and S Yu Philip. 2019. MVStream: Multiview data stream clustering. *IEEE transactions on neural networks and learning systems* 31, 9 (2019), 3482–3496.

[14] Anil K. Jain, M. Narasimha Murty, and Patrick J. Flynn. 1999. Data Clustering: A Review. *ACM computing surveys (CSUR)* 31, 3 (1999), 264–323.

[15] Zhao Kang, Chong Peng, Qiang Cheng, and Zenglin Xu. 2018. Unified Spectral Clustering With Optimal Graph. *Proceedings of the AAAI Conference on Artificial Intelligence* 32, 1 (Apr. 2018). https://ojs.aaai.org/index.php/AAAI/article/view/11613

[16] Ari Kobren, Nicholas Monath, Akshay Krishnamurthy, and Andrew McCallum. 2017. A hierarchical algorithm for extreme clustering. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 255–264.

[17] I. Lee and J. Yang. 2009. Common Clustering Algorithms. In *Comprehensive Chemometrics*, Steven D. Brown, Romá Tauler, and Beata Walczak (Eds.). Elsevier, Oxford, 577–618. https://doi.org/10.1016/B978-044452701-1.00064-8

[18] Yunfan Li, Peng Hu, Zitao Liu, Dezhong Peng, Joey Tianyi Zhou, and Xi Peng. 2021. Contrastive clustering. In *2021 AAAI Conference on Artificial Intelligence (AAAI)*.

[19] Stuart Lloyd. 1982. Least squares quantization in PCM. *IEEE transactions on information theory* 28, 2 (1982), 129–137.

[20] Yonggang Lu and Yi Wan. 2013. PHA: A fast potential-based hierarchical agglomerative clustering method. *Pattern Recognition* 46, 5 (2013), 1227–1239.

[21] Nicholas Monath, Kumar Avinava Dubey, Guru Guruganesh, Manzil Zaheer, Amr Ahmed, Andrew McCallum, Gokhan Mergen, Marc Najork, Mert Terzihan, Bryon Tjanaka, et al. 2021. Scalable Hierarchical Agglomerative Clustering. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 1245–1255.

[22] Nicholas Monath, Ari Kobren, Akshay Krishnamurthy, Michael R Glass, and Andrew McCallum. 2019. Scalable Hierarchical Clustering with Tree Grafting. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1438–1448.

[23] Fionn Murtagh. 1983. A survey of recent advances in hierarchical clustering algorithms. *The computer journal* 26, 4 (1983), 354–359.

[24] Thuy-Diem Nguyen, Bertil Schmidt, and Chee-Keong Kwoh. 2014. SparseHC: A Memory-efficient Online Hierarchical Clustering Algorithm. *Procedia Computer Science* 29 (2014), 8 – 19. https://doi.org/10.1016/j.procs.2014.05.001 2014 International Conference on Computational Science.

[25] Xiaoyu Qin, Kai Ming Ting, Ye Zhu, and Vincent Lee. 2019. Nearest-neighbour-induced isolation similarity and its impact on density-based clustering. In *Proceedings of the 33rd AAAI Conference on AI (AAAI 2019), AAAI Press*.

[26] Anand Rajagopalan, Fabio Vitale, Danny Vainstein, Gui Citovsky, Cecilia M Procopiuc, and Claudio Gentile. 2021. Hierarchical Clustering of Data Streams: Scalable Algorithms and Approximation Guarantees. In *International Conference on Machine Learning*. PMLR, 8799–8809.

[27] Anand Rajaraman and Jeffrey David Ullman. 2011. *Mining of Massive Datasets*. Cambridge University Press.

[28] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. 1998. Nonlinear component analysis as a kernel eigenvalue problem. *Neural computation* 10, 5 (1998), 1299–1319.

[29] D. Sculley. 2010. Web-Scale k-Means Clustering. In *Proceedings of the 19th International Conference on World Wide Web* (Raleigh, North Carolina, USA) *(WWW '10)*. Association for Computing Machinery, New York, NY, USA, 1177–1178. https://doi.org/10.1145/1772690.1772862

[30] John Shawe-Taylor, Nello Cristianini, et al. 2004. *Kernel methods for pattern analysis*. Cambridge university press.

[31] Jonathan A Silva, Elaine R Faria, Rodrigo C Barros, Eduardo R Hruschka, André CPLF de Carvalho, and João Gama. 2013. Data stream clustering: A survey. *ACM Computing Surveys (CSUR)* 46, 1 (2013), 1–31.

[32] Hwanjun Song and Jae-Gil Lee. 2018. RP-DBSCAN: A superfast parallel DBSCAN algorithm based on random partitioning. In *Proceedings of the 2018 International Conference on Management of Data*. 1173–1187.

[33] Kai Ming Ting, Jonathan R Wells, and Ye Zhu. 2022. Point-Set Kernel Clustering. *IEEE Transactions on Knowledge and Data Engineering* (2022). https://doi.org/10.1109/TKDE.2022.3144914

[34] Kai Ming Ting, Bi-Cun Xu, Takashi Washio, and Zhi-Hua Zhou. 2020. Isolation Distributional Kernel: A New Tool for Kernel based Anomaly Detection. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 198–206.

[35] Kai Ming Ting, Yue Zhu, and Zhi-Hua Zhou. 2018. Isolation Kernel and its effect on SVM. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2329–2337.

[36] Michele Tumminello, Fabrizio Lillo, and Rosario N. Mantegna. 2010. Correlation, Hierarchies, and Networks in Financial Markets. *Journal of Economic Behavior & Organization* 75, 1 (2010), 40–58.

[37] Nishant Yadav, Ari Kobren, Nicholas Monath, and Andrew Mccallum. 2019. Supervised Hierarchical Clustering with Exponential Linkage. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, Long Beach, California, USA, 6973–6983.

[38] Lihi Zelnik-Manor and Pietro Perona. 2005. Self-tuning spectral clustering. In *Advances in neural information processing systems*. 1601–1608.

[39] Yu Zhang, Kanat Tangwongsan, and Srikanta Tirthapura. 2017. Streaming k-means clustering with fast queries. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. IEEE, 449–460.

[40] Ye Zhu, Kai Ming Ting, and Mark J Carman. 2016. Density-ratio based clustering for discovering clusters with varying densities. *Pattern Recognition* 60 (2016), 983–997.

[41] Ye Zhu, Kai Ming Ting, Mark J. Carman, and Maia Angelova. 2021. CDF Transform-and-Shift: An effective way to deal with datasets of inhomogeneous cluster densities. *Pattern Recognition* 117 (2021), 107977.