Adaptive Stream Clustering Using Incremental Graph Maintenance

Marwan Hassani HASSANI@CS.RWTH-AACHEN.DE
Data Management and Data Exploration Group, RWTH Aachen University, Germany

Pascal Spaus SPAUS@CS.RWTH-AACHEN.DE
Data Management and Data Exploration Group, RWTH Aachen University, Germany

Alfredo Cuzzocrea ALFREDO.CUZZOCREA@DIA.UNITS.IT
DIA Department, University of Trieste and ICAR-CNR, Italy

Thomas Seidl SEIDL@CS.RWTH-AACHEN.DE
Data Management and Data Exploration Group, RWTH Aachen University, Germany

Editors: Wei Fan, Albert Bifet, Qiang Yang and Philip Yu

Abstract

Challenges for clustering streaming data are getting continuously more sophisticated. This trend is driven by the the emerging requirements of the application where those algorithms are used and the properties of the stream itself. Some of these properties are the continuous data arrival, the time-critical processing of objects, the evolution of the data streams, the presence of outliers and the varying densities of the data. Due to the fact that the stream evolves continuously in the process of its existence, it is crucial that the algorithm autonomously detects clusters of arbitrary shape, with different densities, and varying number of clusters. Recently, the first hierarchical density-based stream clustering algorithm based on cluster stability, called HASTREAM, was proposed. Although the algorithm was able to meet the above mentioned requirements, it inherited the main drawback of density-based hierarchical clustering algorithms, namely the efficiency issues. In this paper we propose *I-HASTREAM*, a first density-based hierarchical clustering algorithm that has considerably less computational time than HASTREAM. Our proposed method utilizes and introduces techniques from the graph theory domain to devise an incremental update of the underlying model instead of repeatedly performing the expensive calculations of the huge graph. Specifically the Prim's algorithm for constructing the minimal spanning tree is adopted by introducing novel, incremental maintenance of the tree by vertex and edge insertion and deletion. The extensive experimental evaluation study on real world datasets shows that I-HASTREAM is considerably faster than a state-of-the-art hierarchical density-based stream clustering approach while delivering almost the same clustering quality.

Keywords: Adaptive stream clustering, Density-based stream clustering, Hierarchical clustering, Incremental maintenance of evolving graphs, Minimal spanning tree.

1. Introduction

Streaming data differs from a static dataset in various ways. Some of the main differences are the continuous arrival of streaming data and the necessity to process them of the time of arrival. Furthermore, due to the huge amount of data, it can not be stored persistently.

Another implication of the endless character of the stream is the impossibility of performing multiple passes over the data, and thus every new object can only be maximally processed once. Every object has to be processed as fast as possible, thus the processing has to be efficient. Buffering objects is also not feasible, since this would only delay the overflow issue and it will additionally make buffered objects irrelevant even before they have been processed. An additional challenge is the evolution of data streams, thus the algorithm has to recognize and *continuously* adapt to these changes in the data.

Most stream clustering algorithms follow the online-offline-phases model. In the online phase, a summary of the evolving data is performed and continuously maintained to follow the changing distribution of the data. The offline phase is then performed upon a user request, and uses one of the well known static clustering algorithms to deliver the final clustering. CluStream Aggarwal et al. (2003) for example, performs a k-means variant in the offline phase over the data summaries (also called microclusters in most algorithms). Motivated by the need to detect arbitrarily-shaped evolving clusters, and being less sensitive to outliers in the stream, density-based stream algorithms were developed like DenStream Cao et al. (2006) which obtains the final clustering by performing a variant of DBSCAN Ester et al. (1996) over the microclusters.

Some of the advantages of density-based clustering algorithms are their ability to find clusters of arbitrary shape and the autonomously detection of the number of clusters. However, a significant drawback of available density-based approaches, is that they are not able to find clusters of different densities, due to the "static" density threshold parameter that is usually used in such algorithms . Having clusters of different densities at the same time is very common in streaming data. Additionally, the same clusters may evolve over the stream such that they will have different densities over the time. Setting the density threshold parameter to a single value all over the stream (like e.g. in DenStream), will lead to missing many clusters either by considering their objects as outliers or merging them within larger clusters.

Recently, the first self-adaptive hierarchical density-based stream clustering algorithm based on *cluster stability*, called *HASTREAM*, was proposed in Hassani et al. (2014b); Hassani (2015). This hierarchical algorithm uses in its offline phase an OPTICS-like concept Ankerst et al. (1999) to overcome the drawback of the single-threshold issue of DenStream Cao et al. (2006). To achieve that, a graph-based representation of the microclusters is presented in HASTREAM using a minimal spanning tree that considers the microclusters as weighted vertices and the distances between them as weighted edges (cf. Section 2).

Despite its adaptive feature and higher accuracies compared to state-of-the-art density-based stream clustering algorithms, HASTREAM inherited partially the efficiency issues that OPTICS Ankerst et al. (1999) has compared to DBSCAN. This is particularly obvious when the stream evolves, and new microclusters are inserted and outdated ones are deleted. HASTREAM needs in this case to perform an expensive calculation of a complete graph (called $\mathcal{MRG}$ in Figure (1) cf. also Section 2) that computes the pair-wise distances between all *recent* microclusters. This is needed for the calculation of the new minimal spanning tree, which has also to be built from scratch as explained in Figure (1). However, the evolution of data is a natural characteristic of streams, and thus higher frequencies of user requests of an offline phase are reasonable, and such heavy computation makes HASTREAM inappropriate for a fast reaction in streaming environments. Additionally, within relatively small intervals of time, only particular microclusters are affected by the insertion or the deletion of microclusters, while the others remain valid and no need to update them completely.

In this paper, we propose *I-HASTREAM*, an incremental hierarchical density-based stream clustering algorithm based on cluster stability. The algorithm retains the strong features of HASTREAM Hassani et al. (2014b), and maintains its efficiency drawbacks by efficiently tackling the above-mentioned issues. To achieve this, *I-HASTREAM* in this work has the following important contributions:

1. It localizes the inserted and deleted microclusters since the previous iteration (called Δ in Figure (1)), and accurately finds the affected microclusters according to Δ (called $Aff(\Delta)$ in Figure (1)).

2. It contributes, to the best of our knowledge, the first technique to update the evolving graph of the minimal spanning tree after deleting vertices. *I-HASTREAM* uses this novel technique and adopts an available one for insertion, in order to maintain the minimal spanning tree using only the information in $\Delta + Aff(\Delta)$ without the need to perform the expensive computation of completely calculating the mutual reachability graph $\mathcal{MRG}$.

3. It uses the updated minimal spanning tree to efficiently build a dendrogram and then to extract the most stable flat clustering in an adaptive manner without a user intervention.

4. It saves more than 30% of the total runtime consumed by HASTREAM over a real dataset and above 78% of the time consumed by the offline phase of HASTREAM over another real dataset.

5. It delivers a better clustering quality than a state-of-the-art hierarchical stream clustering algorithm called MR-Stream Wan et al. (2009) and almost the same clustering quality as HASTREAM. *I-HASTREAM* effectively generates updated minimal spanning trees of nearly the same lengths as the ones generated from scratch. To evaluate this, we contribute additionally a novel *divergence* evaluation measure.

The remainder of this paper is presented as follows: Section 2 gives an overview on different areas of related work. Section 3 presents some needed definitions and the novel introduced method for an incremental maintenance of the streaming graph. In Section 4, we present our *I-HASTREAM* algorithm in details. In Section 5, we thoroughly use two real datasets to evaluate two variants of *I-HASTREAM* against state-of-the-art algorithms before concluding the paper in Section 6.

2. Related Work

In this section we first list the state-of-the-art density-based stream clustering algorithms. Most of them are used as competitors to our approach in the evaluation section. Then, we discuss most prominent algorithms for the graph-based clustering, the construction of a minimal spanning tree and the update of a minimal spanning tree. Most **density-based stream clustering algorithms** have two phases, an online and an offline phase. The online phase

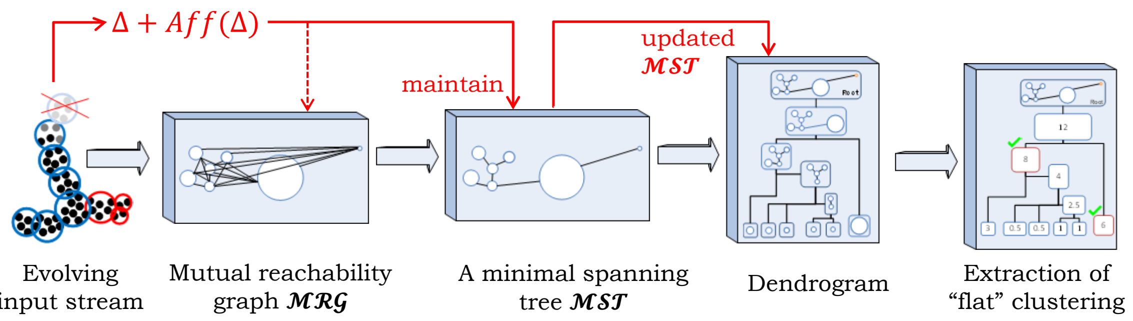
$\Delta + Aff(\Delta)$
maintain
updated $\mathcal{MST}$
Evolving input stream
Mutual reachability graph $\mathcal{MRG}$
A minimal spanning tree $\mathcal{MST}$
Dendrogram
Extraction of "flat" clustering

Figure 1: The general concept of the incremental HASTREAM algorithm in blue. The red arrows represent some of the contributions of *I-HASTREAM*.

is responsible of summarizing the data stream, whereas the offline phase is responsible of the final clustering. **DenStream** Cao et al. (2006) uses the microcluster structure. DenStream maintains two lists of microclusters, the potential and the outlier microclusters. The microclusters are restricted by a maximal radius and new streaming data is only merged into an existing microcluster if its distance between the new data object and the closest center of a microcluster does not exceed the maximal radius threshold. The offline phase is a DBSCAN variant which considers only potential microclusters that exceed a certain density threshold. **LiarTree** Hassani et al. (2011) is an anytime stream clustering algorithm which provides a compact and self-adaptive indexing structure for the microclusters to allow handling of fast and slow data streams. **SubClusTree** Hassani et al. (2014a) is an anytime grid-based subspace clustering version of LiarTree that finds also hidden clusters in the subspaces of the stream at anytime. **MR-Stream** Wan et al. (2009) is a grid-based hierarchical stream clustering algorithm that uses a tree structure to represent the data stream. The data space is partitioned into equally sized cells. At each level of the tree structure, each cell is divided into subcells. The offline clustering is performed on the cells at a user defined hierarchy level of the tree, by clustering all reachable dense cells to a cluster. Thus, different to our approach, the algorithm does not solve the main drawback of density-based stream clustering algorithms discussed here, as substitutes the density threshold with a hierarchy threshold that needs a non-trivial user intervention. Additionally it suffers from the efficiency issues that grid-based stream clustering algorithms usually have. Tu et al. [1] suggested a simple clustering method that keeps blindly merging the microclusters until a user defined number of clusters is reached. **HASTREAM** Hassani et al. (2014b) is the first self-adaptive hierarchical stream clustering algorithm. It is overviewed in the blue parts of Figure (1). Since the contribution of HASTREAM concentrates on the offline phase, its inputs are the current summaries, (microclusters) of the evolving stream. The algorithm first builds a complete graph, called the mutual reachability graph $\mathcal{MRG}$, out of the current microclusters. Then, a minimal spanning tree $\mathcal{MST}$ is extracted out of the $\mathcal{MRG}$. The weight of a microcluster represents a statistic about the number and the recency of the data stream objects that are summarized by that microcluster. Microclusters, representing more objects recently arriving from the stream, have more weight than the ones representing less or outdated objects. According to the weights of the microclusters and the distances between them in $\mathcal{MST}$,

1. Tu, Q., et al. Density-based hierarchical clustering for streaming data. *Pattern Recognition Letters*, 2012

a dendrogram is built in the next step to reflect the hierarchical clustering. HASTREAM contributed additionally a novel weighted cluster stability measure, that extracts the most stable flat clustering Hassani et al. (2014b) in the final step, that adapts to the distribution of the stream without any need to a user intervention.

Graph-based clustering of *static* data is a well-established research area Aggarwal and Wang (2010). A popular method is the concept of minimal spanning trees. Given a connected and undirected graph, a Minimal Spanning Tree (MST) is a subgraph which connects all the vertices and minimizes the sum of the edge costs. Furthermore a MST does not contain any cycles. Minimal spanning tree clustering techniques are used in various algorithms Wang et al. (2012); Grygorash et al. (2006) as well as the presented algorithm HDBSCAN Campello et al. (2013). The **construction of a minimal spanning tree** is an expensive task, which made it less appealing for streaming applications. Let $\mathcal{G}(V, E)$ be a graph with set of vertices V and set of edges E. The complexity of building an MST from graph $\mathcal{G}$ depends on the used approach. The most well known approaches for building the MST are Borůvka's algorithm Boruvka (1926), Prim's algorithm Prim (1957), Kruskal's algorithm Kruskal (1956) and Dijkstra's algorithm Dijkstra (1959). Karger et al. Karger et al. (1995) proposed a randomized algorithm for building the MST with a linear expected complexity. Prim's algorithm Prim (1957) starts with an arbitrary vertex $v \in V$. Then it proceeds by augmenting the MST with the edge $e \in E$ that has the lowest cost and is adjacent to a vertex $w \in V$ that is not yet contained in the MST. If such an edge e exists, the adjacent vertex v is also incorporated into the MST. This process is repeated until every vertex of $\mathcal{G}(V, E)$ is contained in the MST. The runtime of Prim's algorithm using an adjacency matrix is $O(|V|^2)$, and can be improved to $O(|E| + |V| \log|V|)$ by using the Fibonacci heap. The **maintenance of a minimal spanning tree** is not a straightforward task as it needs to consider the insertion of a vertex, the deletion of a vertex, the insertion of an edge and the deletion of an edge. Chin and Houck Chin and Houck (1978) presented methods for updating the MST for the insertion and deletion of vertices. They showed that the insertion of a new vertex can be done in linear time. On the other hand, the proposed deletion method of a vertex has quadratic complexity with respect to the number of vertices. Furthermore the authors show that the insertion and the deletion of edges can be realized under the same runtime complexity constraints. The deletion case is the more complex task and, to the best of our knowledge, no existing approach can solve the problem in a linear time. Das and Loui Das and Loui (1999) used the concept of supervertices to improve the runtime complexity for deleting a vertex when maintaining an MST. A supervertex is a set of vertices, representing a connected component of an MST. The time complexity of the proposed algorithm is $O(|E|log|V|)$. Nardelli et al. Nardelli et al. (2004) proposed an nearly linear time algorithm for reconstructing the MST of a communication-based network after a node failure, which can be regarded as deleting a vertex. The time complexity is $O(|E|\alpha(|E|, |V|))$, where α is the inverse Ackerman function.

3. Preliminaries and Main Concepts of *I-HASTREAM*

We introduce in the following some important definitions and preliminaries that are needed to present our approach later.

Definition 1 (Core Distance) *The core distance of a microcluster mc_p is defined as the smallest distance to its minPts-nearest neighbor microcluster, i.e. the smallest distance that contains minPts neighboring microclusters to the current microcluster. We refer to it as* $\text{core-dist}_{\text{minPts}}(mc_p)$.

Definition 2 (Mutual Reachability Distance) *The mutual reachability distance between two microclusters mc_p and mc_q is defined as the maximum of the Euclidean distance between the centers of the microclusters and their respective core-distances.*

$$\begin{aligned}\text{dist}_{mr}(mc_p, mc_q) = \max \{\, & \text{dist}_{Euc}(mc_p, mc_q)\,, \\ & \text{core-dist}_{\text{minPts}}(mc_p), \text{core-dist}_{\text{minPts}}(mc_q)\}\end{aligned}$$

Definition 3 (Mutual Reachability Graph) *The mutual reachability graph $\mathcal{MRG}(V,E)$ is a complete graph. The set of vertices V is represented by the microclusters available at timestamp t. The weight of an edge from the set of edges E represents the mutual reachability distance between the two microclusters represented by vertices $u, v \in V$.*

$$\mathcal{MRG}(V,E) = \begin{cases} V = \textit{the set of available microclusters at time } t \\ E = \{e(u,v) \mid u, v \in V \textit{ with weight } w(e) = \text{dist}_{mr}(u,v)\} \end{cases}$$

Definition 4 (Adjacency List) *Given a connected graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ (a graph where there is a path of one or more edges $e \in \mathcal{E}$ between each pair of vertices $v, u \in \mathcal{V}$), the adjacency list of vertex v contains all vertices which are directly reachable from v by one edge.*

$$\text{Adj}_E(v) = \{u \in \mathcal{V} \mid e(v,u) \in E\}$$

Definition 5 (Connected Component) *A connected component is a set of vertices $V \subseteq \mathcal{V}$, where each vertex has at least one adjacent vertex, for $|V| \geq 2$.*

$$\text{Conn-Component}(V,E) = \Big\{ v \in V \mid \text{Adj}_E(v) \neq \varnothing \wedge \text{Adj}_E(v) \neq \{v\} \Big\}$$

A single vertex is also a connected component.

Definition 6 (Total Weight of a Connected Component) *The total weight of a connected component* Conn-Component(V,E) *is the sum of the weights of each $v \in V$.*

$$\textit{Total Weight}(\text{Conn-Component}(V,E)) = \sum_{v \in V} \textit{weight of } v$$

Definition 7 (Minimal Spanning Tree) *Given a connected and undirected graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, a minimal spanning tree $\mathcal{MST}(V,T)$ is a connected subgraph of $\mathcal{G}$ which contains no cycles and minimizes the total weight w.r.t. the edges.*

$$\mathcal{MST}(V,T) = \begin{cases} V = \mathcal{V} \\ T = \Big\{ e(u,v) \in \mathcal{E} \;\Big|\; \textit{minimize } \sum_{e \in \mathcal{E}} w(e) \Big\} \\ \text{Conn-Component}(V,T) \end{cases}$$

3.1. The Incremental Maintenance of the Minimal Spanning Tree

This section presents a method for computing an $\mathcal{MST}$ at time $t+1$ by updating the $\mathcal{MST}$ from timestamp t. The motivation for updating the $\mathcal{MST}$, instead of recomputing it, is obvious. The user requests for a new result are in many cases so close to each other that many underlying, available microclusters from the previous results are still representing the distribution of the data even with very fast data streams. Usually, only a small set of new microclusters are generated and a small set of microclusters are removed.

3.1.1. Insertion and Deletion of Microclusters in $\mathcal{MRG}$

Instead of recomputing all the core-distances which are needed for building the $\mathcal{MRG}$ from scratch, it can be observed that only the core-distances of a subset of the microclusters change. This subset contains the affected microclusters due to the insertions or the deletion. The affected microclusters can exist only within the old core-dist$_{\text{minPts}}$-neighborhood of the deleted or the inserted microclusters Pokrajac et al. (2007). An insertion of a new microcluster within the old core-distance neighborhood shrinks the core-distance of the microcluster (cf. Figure (2(a))). A deletion of a microcluster from the old core-dist$_{\text{minPts}}$-neighborhood, increases the core distance (cf. Figure (2(b))).

Definition 8 (Updating the Core-Distance: Insertion Case) *The old core-distance* core-dist$^{old}_{\text{minPts}}(mc)$ *of a microcluster mc needs to be recomputed only if the distance between mc and the newly inserted microcluster i is lower than* core-dist$^{old}_{\text{minPts}}(mc)$*:*

$$\text{core-dist}^{new}_{\text{minPts}}(mc) := \begin{cases} \textit{update} & \textit{if } \text{dist}_{Euc}(mc, i) < \text{core-dist}^{old}_{\text{minPts}}(mc) \\ \textit{no change} & \textit{otherwise} \end{cases}$$

In case of a recomputation of the core-distance, the new core-distance core-dist$^{new}_{\text{minPts}}(mc)$ *is smaller than* core-dist$^{old}_{\text{minPts}}(mc)$*.*

Definition 9 (Updating the Core-Distance: Deletion Case) *The old core-distance* core-dist$^{old}_{\text{minPts}}(mc)$ *of a microcluster mc needs to be recomputed only if the distance between mc and the deleted microcluster d is lower than* core-dist$^{old}_{\text{minPts}}(mc)$*:*

$$\text{core-dist}^{new}_{\text{minPts}}(mc) := \begin{cases} \textit{update} & \textit{if } \text{dist}_{Euc}(mc, d) < \text{core-dist}^{old}_{\text{minPts}}(mc) \\ \textit{no change} & \textit{otherwise} \end{cases}$$

In case of a recomputation of the core-distance, the new core-distance core-dist$^{new}_{\text{minPts}}(mc)$ *is larger than* core-dist$^{old}_{\text{minPts}}(mc)$*.*

Updating the core-distances affects the mutual reachability distance between the microclusters . This implies that there is no guarantee that the minimal spanning tree of the previous iteration is still valid. Thus the vertices, whose mutual reachability distances are affected, have to be considered during the update process. The updating of the minimal spanning tree consists of two phases. First, the deletion of the obsolete microclusters is performed, then the insertion of the newly created microclusters is performed as follows. This order is chosen to minimize the number of updated vertices in the insertion case as we do not need to update the core distances of the already deleted microclusters.

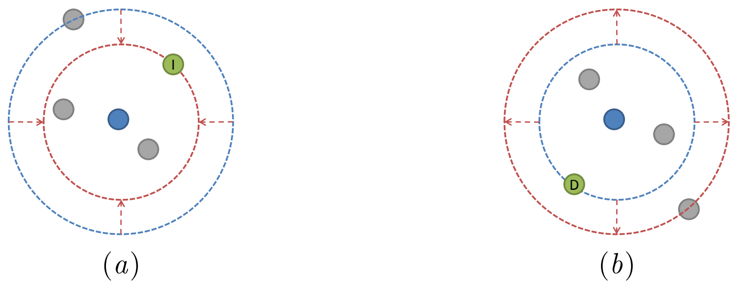

Figure 2: Shrinking and Growth of Core-Distances: Effects of inserting (a) or deleting (b) a green microcluster (vertex), $minPts = 4$. The red line represents the new clustering.

3.1.2. Deletion of Vertices From the Minimal Spanning Tree

Removing vertices from an $\mathcal{MST}$ splits it into two or more connected subgraphs we call them *subcomponents* in the context of $\mathcal{MST}$. To connect the subcomponents again, it is not sufficient to search for the lowest weighted edges which connect the subcomponents. Deleting a microcluster mc_d affects the core-distances of some other microclusters, and thus directly affects the mutual reachability distance. First, the weights of the affected edges have to be updated. Only microclusters which had mc_d in their core-dist$_{\text{minPts}}$-neighborhood are affected by the deletion. This set of affected microcluster is referred to as AFFECTED$_d$. This set can be computed by a linear scan over all the microclusters. For each microcluster in AFFECTED$_d$, it is checked if the old core-distance is affected, as defined in Definition (9). Since only the core-distances of the microclusters in AFFECTED$_d$ are affected, only a subset of the edges from the $\mathcal{MRG}$ has to be updated. The mutual reachability distance between two microclusters that are not affected remains the same. After updating the $\mathcal{MRG}$, the microcluster mc_d is removed from the $\mathcal{MST}$. From each subcomponent, the affected microclusters AFFECTED$_d$ are removed. The reason for this is that the weights of the edges are also affected and those edges might no longer belong to the minimal spanning tree. On the other hand, the remaining connected subcomponents are still part of the minimal spanning tree since the mutual reachability distances of those microcluster have not changed. The goal is to reconnect the microclusters from AFFECTED$_d$ and the remaining connected subcomponents to form a $\mathcal{MST}$. Therefore, the approach of supervertices is used Das and Loui (1999). The reason of using this approach is to reduce the number of vertices and edges when performing Prim's algorithm in the final step. A supervertex is a vertex which represents a set of vertices which already are a connected component. In this setup, each microcluster $mc \in$ AFFECTED$_d$ and each connected subcomponents is represented by a supervertex. Based on these supervertices, an additional complete graph is generated. The edge between two supervertices is the lowest possible weighted edge from the $\mathcal{MRG}$ which connects two vertices from the two supervertices (cf. Definition (10)). The final step is to perform Prim's algorithm on the reduced complete graph and extract the new $\mathcal{MST}$. In the worst case, i.e. when all microclusters are affected by deleting mc_d, this approach requires the same runtime as rebuilding the $\mathcal{MST}$, since the minimal spanning tree is rebuilt over all microclusters.

Definition 10 *Given a mutual reachability graph $\mathcal{MRG}(V,E)$, the edge between two super-vertices sv_1 and sv_2 is defined as follows:*

$$e(sv_1, sv_2) = e(u,v), \quad \textit{with } w(e(u,v)) \leq w(e(u',v'))$$

for $u, u', v, v' \in V$ and $u, u' \in sv_1$ and $v, v' \in sv_2$

3.1.3. Insertion of Vertices to the Minimal Spanning Tree

Similar to the deletion case, the insertion of new microclusters affects the core-distances of the existing microclusters. The insertion case is easier to handle than the deletion case. The update of the minimal spanning tree can be realized in linear time as presented in Chin and Houck (1978). First, all the microclusters which are affected by inserting mc_i are collected and are referred to as AFFECTED$_i$. A linear scan is performed and for each microcluster, it is checked whether the old core-distance is affected, as defined in Definition (8). The affected microclusters have to be reinserted into the $\mathcal{MST}$. The reason is that the weight of the edges changed due to the existence of mc_i and the existing $\mathcal{MST}$ might no longer be the correct $\mathcal{MST}$ w.r.t. the new weights of the edges. The algorithm in Chin and Houck (1978) is able to update a $\mathcal{MST}$ when the weight of the edges decreases. After having reconstructed the $\mathcal{MST}(V,T)$ w.r.t. the changed weight of the edges, the new microcluster mc_i is inserted into the $\mathcal{MST}$. The basic idea of Chin and Houck (1978) is to insert a new vertex, thus a new microcluster, to the $\mathcal{MST}$ is the following. A random vertex $r \in V$ is selected as root and at each timestamp the algorithm adds a new edge to the tree T. When a cycle is created, the largest edge from that cycle is removed. This is accomplished by a depth-first search. At the end, the result is a minimal spanning tree. The algorithm is explained in more detail in Section (4).

3.1.4. The Effect of Microclusters Movement

The previous incremental update techniques consider the inserted and the deleted microclusters and the accordingly affected ones when updating the $\mathcal{MST}$. Thus, the method does not make any assumption about any distribution of the data, and follows the drift correctly. However, in streaming data another effect is there. The centers of available microclusters who are not considered as outdated might shift which has an effect over the generated $\mathcal{MST}$. We introduce a movement threshold MT that defines the maximal tolerated shift of the microcluster center, after which the microcluster is added to the AFFECTED set regardless whether it was affected by other insertions or deletions. It is set as a percentage of the online epsilon ε_{online} from the online phase. The effect of this threshold is studies thoroughly in Section 5.

4. The *I-HASTREAM* Algorithm

In the following, we will explain mainly the algorithms that are related to our contribution in I-HASTREAM. For the other parts (represented by blue in Figure (1)), we refer the reader to Hassani et al. (2014b) due to space limitations. Algorithm 1 abstracts the general method. Each time a microcluster is created or deleted it is buffered in order to maintain the $\mathcal{MST}$. As described in Section (3.1), the procedure requires two phases. The deletion phase is illustrated in Algorithm 2. This algorithm updates the core-distances of

Algorithm 1 I-HASTREAM_MST_Update()

```
1: determine AFFECTED$_d$
2: $updateMST\_delete()$;    // cf. Algorithm 2
3: determine AFFECTED$_i$
4: $updateMST\_insert()$;    // cf. Algorithm 3
5: return $\mathcal{MST}$
```

each affected microcluster, due to the removal of the microcluster from AFFECTED$_d$. After the removal of the microclusters, a reduced mutual reachability graph is generated, based on the supervertices composed of the remaining connected components. By applying Prim's algorithm, a new $\mathcal{MST}$ is generated on the reduced graph.

Algorithm 2 updateMST_delete()

```
1: update core distances w.r.t. AFFECTED$_d$
2: remove AFFECTED$_d$ from $\mathcal{MST}$, generate supervertices from components
3: generate complete graph $\mathcal{G}_S$ on supervertices
4: perform Prim's algorithm on $\mathcal{G}_S$
5: return $\mathcal{MST}$
```

Afterwards, the insertion phase is executed, which is illustrated in Algorithm 3 . The algorithm starts by reconstructing the $\mathcal{MST}$, which is necessary as newly inserted microclusters can change the core-distances and thus the mutual reachability distances. Each

Algorithm 3 updateMST_insert()

```
1: update core distances w.r.t. AFFECTED$_i$
2: reconstruct $\mathcal{MST}$
3: for each new vertex $v$ do
4:     select random root vertex $r$
5:     $insertToMST(r, v)$    // cf. Algorithm 4
6: end for
7: return $\mathcal{MST}$
```

affected vertex v from AFFECTED$_i$ is iteratively reinserted into the $\mathcal{MST}$. For each reinserted vertex v a fake vertex f is generated, which has the same edges as v to the other vertices, i.e. $w(e(v,f)) = w(e(u,f))$ for $u \in V$ and $u \neq v$. The edge $e(v,f)$ between v and f has a weight smaller than the smallest edge from the $\mathcal{MST}$. This ensures that this edge is contained in the new $\mathcal{MST}$. After performing Algorithm 4, the vertex f is removed, thus the edge $e(v,f)$ is also removed. The remaining task is to map the edges $e(v,f)$ and $e(u,f)$ correctly, by selecting the smaller edge w.r.t the weight. Algorithm 4 illustrates the insertion of a new microcluster mc_i into the reconstructed $\mathcal{MST}$. The algorithm performs a depth first search. The algorithm needs a root vertex, which can be chosen arbitrary. It adds the new edges to the $\mathcal{MST}$. By adding edges, cycles can occur. Each time a cycle occurs, the largest edge of the cycle is removed. This guarantees that the resulting set of edges does belong to $\mathcal{MST}$.

Algorithm 4 insertToMST(Vertex r, Vertex v)

```
1: mark $r$ as processed
2: newEdge $\leftarrow e(r, v)$
3: for each $u \in \text{Adj}_r$ do
4:   if $u$ not processed then
5:     insertToMST$(u, v)$   // recursive call. To resolve cycles:
6:     $l \leftarrow$ maximum of the edges: $\{t$ and $e(u, r)\}$
7:     $s \leftarrow$ minimum of the edges: $\{t$ and $e(u, r)\}$
8:     add $s$ to tree edges $T'$
9:     if $w(l) < w(newEdge)$ then
10:      newEdge $\leftarrow l$
11:    end if
12:  end if
13: end for
14: $t \leftarrow$ newEdge
```

5. Experimental Evaluation

Four variants of I-HASTREAM are compared to two variants of HASTREAM Hassani et al. (2014b), MR-Stream Wan et al. (2009) and DenStream Cao et al. (2006). One variant applies the index structure IS of the microclusters in the online phase, and three use the list structure LS. The three LS variants test the effect of the movement threshold (cf. Section 3.1.4), by setting MT to $0.5 \times \varepsilon_{online}$ in one variant, $1 \times \varepsilon_{online}$ in another and ignoring the movement IM in the third. All algorithms were implemented within MOA Framework MOA. The different parameters were set as in Hassani et al. (2014b).

The clustering quality of the algorithms is evaluated using the Purity measure Zhao and Karypis (2004) and the Cluster Mapping Measure (CMM) Kremer et al. (2011) . To examine how strong the movement of the microclusters affects the found clustering of I-HASTREAM, we introduce the **divergence** of the minimal spanning tree as an evaluation measure of the incremental update. We define the total weight of a minimal spanning tree $\mathcal{MST}(V, T)$ as the total weight of its edges: $w_{\text{total}}(\mathcal{MST}(V, T)) = \sum_{e \in T} w(e)$. The deviation, or the erroneous increase of the length, of the updated minimal spanning tree *update* from the ground truth gt (the one computed from the $\mathcal{MRG}$) at timestamp t is derived as follows: $divergence_t(gt, update) = \left|1 - \frac{w_{\text{total}}(updated)}{w_{\text{total}}(gt)}\right|$. The smaller the divergence value the better the incremental update quality of $\mathcal{MST}$. Additionally, the runtime is used to measure the efficiency of the different algorithms. It can be seen from Figure (3) that all I-HASTREAM variants outperform DenStream. Additionally, all variants of I-HASTREAM with the list structure LS have equal clustering purity to the LS variant of HASTREAM all over the stream. An interesting observation is that the IS variant of I-HASTREAM has slightly higher clustering purity than the IS variant of HASTREAM, although the latter performs an expensive calculation of the $\mathcal{MST}$ after each iteration. I-HASTREAM benefits in this case from the purity characteristic of penalizing the existence of different classes within the current cluster. This is due to the relatively lazy nature of following the shifting of microclusters that the IS variant of I-HASTREAM has compared to the other variants.

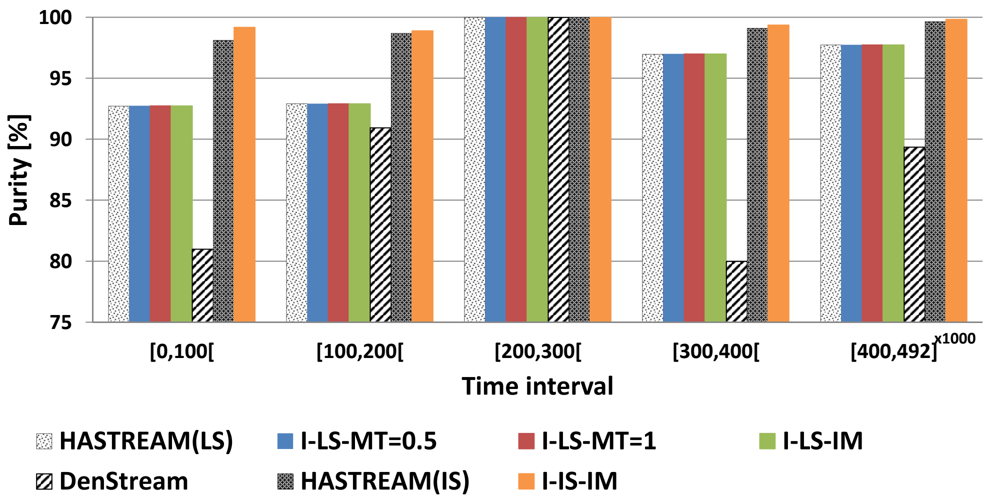

Figure 3: Purity results of the Network Intrusion Dataset.

Figure (4) represents the CMM results of the real world dataset: the Physiological Dataset Physiological (2004) over different time intervals over the stream. The averaged CMM for time intervals of length 100 is presented. Again, all HASTREAM variants have higher clustering qualities than DenStream over all the stream. Additionally, all I-HASTREAM variants of LS have equal CMM almost always, and these are also equal to the CMM value of HASTREAM$_{LS}$. This shows that, on the one hand, that the introduced MT threshold has almost a neglected effect over the quality of the final results, and on the other hand, an equal quality of the results of HASTREAM. By comparing the IS variant of I-HASTREAM to HASTREAM$_{LS}$, one can observe here a slightly better clustering quality in HASTREAM$_{LS}$, this is expected as HASTREAM calculates continuously a minimal spanning tree out of the recent microclusters. To observe the difference between the $\mathcal{MST}$s calculated by the different variants of I-HASTREAM and compare them to the one calculated by HASTREAM, the divergence over the Physiological Dataset is visualized in Figure (6). Figure (5) depicts a comparison of the purity results between two variants of HASTREAM (LS and IS) and their I-HASTREAM versions (in blue and orange). Additionally, the results for the comparison of I-HASTREAM against DenStream Cao et al. (2006) and MR-Stream Wan et al. (2009) (cf. Section 2) are presented in Figure (5). For MR-Stream we have chosen the following parameter settings, either because suggested by the authors of MR-Stream or after performing several iterations and picking the best ones. The fading parameters: $\lambda = 1.002$ and $a = 0.1$, the maximum weight of a sparse node: $C_L = 0.8$ and of a dense node: $C_H = 3$, the constant of the density threshold function: $c = 0.5$, the neighborhood estimation threshold: $\epsilon = 5$, the minimum weight of a cluster: $\beta = 0$, the minimum size of a cluster: $\mu = 2$ and the maximum height of the tree $H = 5$. To obtain the final clustering we have selected the Level 5 of the tree which contains the most fine-grained grid representation of the stream, and thus yields the best clustering quality.

One observation from Figure (5) is the increased purity of the two I-HASTREAM compared to their HASTREAM counterparts over most time intervals. This is against the intuition, as I-HASTREAM approximates the "perfect" minimal spanning trees obtained by HASTREAM. It can be probably explained by the differences between the maximized values

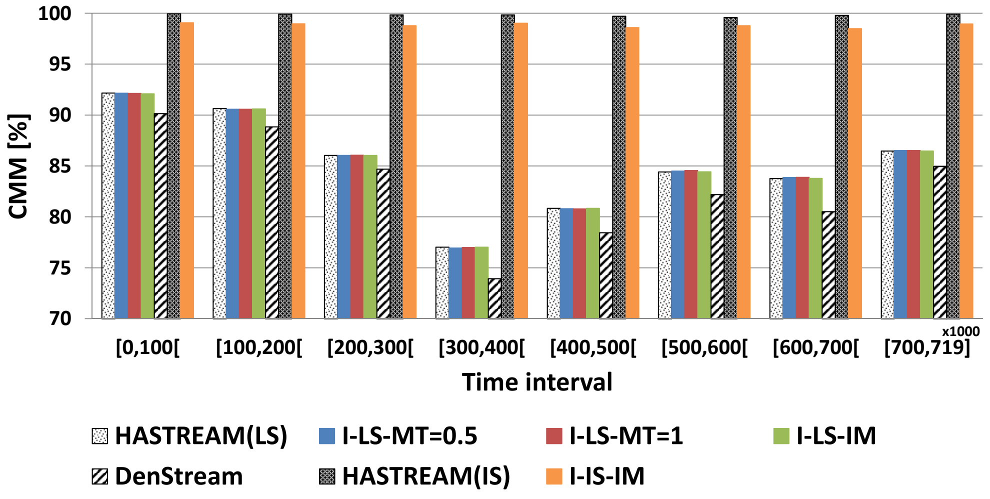

Figure 4: CMM results for the Physiological Dataset.

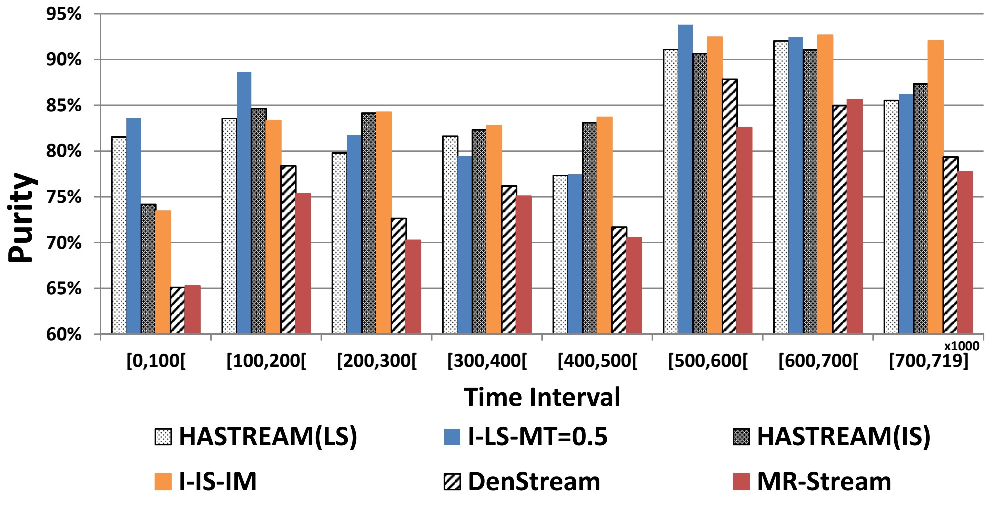

Figure 5: Purity comparison to MR-Stream using the Physiological Dataset.

of the purity measure and the stability function. Another observation is the superiority of both variants of I-HASTREAM over the state-of-the-art hierarchical stream clustering algorithm MR-Stream over all time intervals. This is due to the *static* setting of the clustering hierarchy level which was manually set to 5. As the stream evolves, there will be necessity to adapt the clustering hierarchy level in order to reflect the best and the most stable flat clustering. I-HASTREAM enables this without any user intervention through its clustering stability function.

Figure (6) shows that allowing a higher tolerance w.r.t. the movement of the microclusters increases the inaccuracy of the updated minimal spanning tree. However, completely ignoring the movement of the microclusters improves the quality of the updated $\mathcal{MST}$ (see the graph of IM in Figure (6)). This is due to the fewer number of microclusters added to the AFFECTED set in the case of the IM variant of I-HASTREAM. The other observation we had is that the CMM and purity values were not strongly affected when using

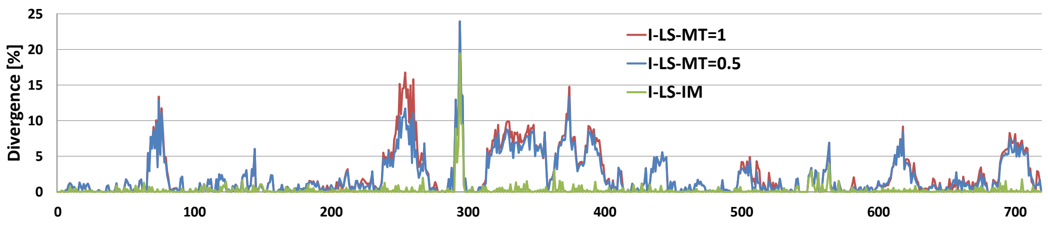

Figure 6: Incremental MST divergence: Physiological Dataset

Algorithm	Runtime at some selected timestamps in [ms]			
	125 ×1000	250 ×1000	375 ×1000	492 ×1000
HASTREAM(LS)	3389008	3926979	5282872	7905545
I-LS-MT=0.5	**2666543**	**3147540**	**3909946**	**5486665**
I-LS-MT=1	**2663978**	**3143275**	**3931565**	**5530857**
I-LS-IM	**2535244**	**3005440**	**3767156**	**5198459**
DenStream	2475282	2945663	3596089	4846652
HASTREAM(IS)	400661	494548	597135	626882
I-IS-IM	466571	581299	732806	768318

Algorithm	Avg. after each evaluation [ms]	Total offline [ms]
HASTREAM(LS)	28.2357	20329.7
I-LS-MT=0.5	**6.0170**	**4332.2**

Table 1: Runtime evaluation [Top]: total runtime of the Network Intrusion Dataset, [Bottom]: the runtime of the offline phase of the Physiological Dataset.

the IM variant of I-HASTREAM. Table (1) [Top] illustrates the accumulated runtime of each of the evaluated algorithms for the real world dataset: Network Intrusion Dataset KDDCup (1999) at some selected timestamps. It can be seen that the three LS variants of I-HASTREAM are considerably faster than the LS variant of the hierarchical clustering approach HASTREAM. This is due to it novel method of incrementally updating the $\mathcal{MST}$ as the stream evolves. HASTREAM$_{LS}$ needs more time to perform the expensive $\mathcal{MRG}$ as the stream evolves. It can only then calculate the $\mathcal{MST}$. The efficiency of the LS variants of I-HASTREAM is very close to that of DenStream, although the latter is not a hierarchical clustering one and performs a single scan of DBSCAN over the microclusters. This shows an interesting efficiency result of a hierarchical approach over streaming data. An additional observation from Table (1) [Top] is the low running times of the IS variants of both HASTREAM and I-HASTREAM. The main reason for this comes from the online phase where the logarithmic complexity of the index structure allows for a fast maintenance of the microclusters. Another observation is that HASTREAM$_{LS}$ is slightly faster than the IS variant of I-HASTREAM. One reason for this is probably the cost of searching for the affected microclusters due to an insertion or a deletion which might not be in the index structure as straight-forward as in the list structure. Table (1) [Bottom] shows the runtime results of the offline phase only of the LS variant of I-HASTREAM compared to HASTREAM$_{LS}$ using the Physiological Dataset. The evaluation shows a huge saving of I-HASTREAM in the average consumed time in the offline phase during each offline clustering (after each 1000 points) compared to HASTREAM. The total running time of the offline phase is reduced more than five times when using the novel incremental update methods of I-HASTREAM.

6. Conclusion

In this paper, we proposed I-HASTREAM, a novel incremental algorithm for updating a streaming minimal spanning tree for a hierarchical density-based clustering on evolving data streams. The proposed algorithm uses a localization technique to narrow the effect of an insertion or a deletion of microclusters and then maintain only that area of the clustering output which was affected by the change. The algorithm performs this maintenance by using the contributed methods for updating a minimal spanning tree of the microclusters without rebuilding it. The extensive experimental evaluation study on real world datasets showed that I-HASTREAM is considerably faster than a state-of-the-art hierarchical density-based stream clustering approach while delivering almost the same clustering quality. Thanks to its adaptive method of the density thresholds, I-HASTREAM has shown also through an evaluation that it delivers significantly better clustering results when compared to another state-of-the-art density-based hierarchical clustering algorithm.

References

Charu C. Aggarwal and Haixun Wang. *Managing and Mining Graph Data.* Advances in Database Systems. Springer, 2010. ISBN 9781441960450.

Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip S. Yu. A framework for clustering evolving data streams. In *VLDB'03*, pages 81–92, 2003.

Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. OPTICS: Ordering Points To Identify the Clustering Structure. *SIGMOD Rec.*, 1999.

Otakar Boruvka. O Jistém Problému Minimálním (About a Certain Minimal Problem) (in Czech, German summary). *Práce Mor. Prírodoved. Spol. v Brne III*, 1926.

Ricardo J. G. B. Campello, Davoud Moulavi, and Jörg Sander. Density-based clustering based on hierarchical density estimates. In *PAKDD'13 (2)*, pages 160–172, 2013.

Feng Cao, Martin Ester, Weining Qian, and Aoying Zhou. Density-based clustering over an evolving data stream with noise. In *SDM'06*, pages 328–339, 2006.

Francis Chin and David Houck. Algorithms for updating minimal spanning trees. *Journal of Computer and System Sciences*, 16:333 – 344, 1978.

Bevan Das and Michael C. Loui. Reconstructing a minimum spanning tree after deletion of any node. *Algorithmica*, pages 530–547, 1999.

E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, pages 269–271, 1959.

Martin Ester, Hans-Peter Kriegel, S Jörg, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, pages 226–231, 1996.

Oleksandr Grygorash, Yan Zhou, and Zach Jorgensen. Minimum spanning tree based clustering algorithms. In *Intl. Conf. on Tools with Artificial Intelligence*, pages 73–81, 2006.

Marwan Hassani. *Efficient Clustering of Big Data Streams.* PhD thesis, RWTH Aachen University, 2015.

Marwan Hassani, Philipp Kranen, and Thomas Seidl. Precise anytime clustering of noisy sensor data with logarithmic complexity. In *SensorKDD @KDD*, pages 52–60, 2011.

Marwan Hassani, Philipp Kranen, Rajveer Saini, and Thomas Seidl. Subspace anytime stream clustering. In *SSDBM'14*, page 37, 2014a.

Marwan Hassani, Pascal Spaus, and Thomas Seidl. Adaptive multiple-resolution stream clustering. In *MLDM'14*, pages 134–148, 2014b.

David R. Karger, Philip N. Klein, and Robert E. Tarjan. A randomized linear-time algorithm to find minimum spanning trees. *J. ACM*, pages 321–328, 1995.

KDDCup. Dataset. http://kdd.ics.uci.edu/databases/kddcup99, 1999.

Hardy Kremer, Philipp Kranen, Timm Jansen, Thomas Seidl, Albert Bifet, Geoff Holmes, and Bernhard Pfahringer. An effective evaluation measure for clustering on evolving data streams. In *Proc. KDD'11*, pages 868–876, 2011.

Joseph B. Kruskal. On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. *Proceedings of the American Mathematical Society*, pages 48–50, 1956.

MOA. Framework. http://moa.cms.waikato.ac.nz/details/stream-clustering/.

Enrico Nardelli, Guido Proietti, and Peter Widmayer. Nearly linear time minimum spanning tree maintenance for transient node failures. *Algorithmica*, pages 119–132, 2004.

Physiological. Dataset. http://www.cs.purdue.edu/commugrate/data/2004icml/, 2004.

D. Pokrajac, A. Lazarevic, and L.J. Latecki. Incremental local outlier detection for data streams. In *CIDM'07*, pages 504–515, 2007.

R. C. Prim. Shortest connection networks and some generalizations. *The Bell Systems Technical Journal*, pages 1389–1401, 1957.

Li Wan, Wee Keong Ng, Xuan Hong Dang, Philip S. Yu, and Kuan Zhang. Density-based clustering of data streams at multiple resolutions. *TKDD*, pages 14:1–14:28, 2009.

Xiaochun Wang, Xiali Wang, and D. Mitch Wilkes. A minimum spanning tree-inspired clustering-based outlier detection technique. In *ICDM*, pages 209–223. Springer, 2012.

Ying Zhao and George Karypis. Empirical and theoretical comparisons of selected criterion functions for document clustering. *Mach. Learn.*, pages 311–331, 2004.