Chapter 2
Overview of Efficient Clustering Methods for High-Dimensional Big Data Streams

Marwan Hassani

2.1 Introduction

Clustering is a well-established data mining concept that aims at automatically grouping *similar* data objects while separating *dissimilar* ones. This process is strongly dependent on the notion of *similarity*, which is often based on some distance measure. Thus, similar objects are usually close to each other while dissimilar ones are far from each other. The clustering task is performed without a previous knowledge of the data, or in an *unsupervised* manner.

During the early stages of data mining research, the whole data objects were considered to be statically and permanently stored in the memory. This allowed the designed data mining technique to perform as much passages over the objects as needed to deliver the desired patterns. In the era of big data, the recent growth of the data size and the easiness of collecting data made the previous settings no more convenient. The size of the continuously generated data and the limited storage capacity allow in many scenarios for a single passage over the data, and users are interested in gaining a real-time knowledge about the data as they are produced.

A data stream is an ordered sequence of objects that can be read once or very small number of times using limited processing and computing storage possibilities. This sequence of objects can be endless and flows usually at high speeds with a varying underlying distribution of the data. This fast and infinite flow of data objects does not allow the traditional permanent storage of the data and thus multiple passages are not any more possible. Many domains are dealing essentially with data streams. The most prominent examples include network traffic data, telecommuni-

M. Hassani (✉)
Analytics for Information Systems Group, Eindhoven University of Technology, Eindhoven, The Netherlands
e-mail: m.hassani@tue.nl

O. Nasraoui, C.-E. Ben N'Cir (eds.), *Clustering Methods for Big Data Analytics*, Unsupervised and Semi-Supervised Learning,
https://doi.org/10.1007/978-3-319-97864-2_2

cation records, click streams, weather monitoring, stock trading, surveillance data, health data, customer profile data, and sensor data. There is many more to come. A very wide spectrum of real-world streaming applications is expanding. Particularly in sensor data, such applications spread from home scenarios like the smart homes to environmental applications, monitoring tasks in the health sector [13], in the digital humanities using eye-tracking [21] or gesture monitoring but do not end with military applications. Actually, any source of information can easily be elaborated to produce a continuous flow of the data. Another emerging streaming data sources are social data. In a single minute, 456,000 tweets are happening, 2,460,000 pieces of content are shared on Facebook, Google conducts 3,607,080 searches, the weather channel receives 18,055,555.56 forecast requests, Uber riders take 45,787.54 trips, and 72 h of new videos are uploaded to YouTube while 4,146,600 videos are watched.[1]

Users are interested in gaining the knowledge out of these information during their same minute of generation. A delay, say till the next minute, might result in an outdated knowledge. Furthermore, with the introduction of the new regulations for data privacy protection like GDPR[2] starting into effect at the end of May 2018, businesses dealing with sensitive user profile data are not allowed anymore to store them. Thus, they would need to switch to a flexible and streaming structure to manage and analyze real-time customer behavior data.

The abovementioned emerging applications motivated dozens of research topics and development frameworks during the previous one and a half decades. Extracting a real-time knowledge out of large numbers of objects with an evolving nature required a different look at data than the traditional *static* one. Particularly, an unsupervised mining of evolving objects in the real-time was needed in multiple applications. In this chapter, we give an overview of the challenges as well as the contributions in this emerging field by highlighting some of the main algorithms there.

The remainder of this chapter is organized as follows: Sect. 2.2 introduces data streams with some applications. In Sect. 2.3, we list the challenges one has to face while designing stream clustering algorithms for mining big data. In Sect. 2.4, we present, at a high level of abstraction, recent approaches in the field of big data stream mining that overcame the challenges mentioned in Sect. 2.3. Finally, Sect. 2.5 concludes this chapter.

2.2 Streaming Data

Objects from the perspective of static mining approaches are all available in the memory while running the algorithm. As a result, the algorithm can scan objects as much as needed to generate the final output without assuming any order for reading

[1] Sources: domo.com and statisticbrain.com.

[2] https://www.eugdpr.org/.

these objects. In the streaming setting, an endless flow of objects o_1, o_2, …, o_i, … of the dataset D is seen at timestamps t_1, t_2, …, t_i, …, respectively, where $t_{i+t} > t_i$ for all i values. Each object o_i seen at t_i is a d-dimensional tuple: $o_i = (o_{i1}, o_{i2}, \ldots, o_{id}) \in D$.

Due to the endless flow of streaming objects, it is not realistic to assume a possibility of memory storage of all $o_i \in D$. This is mainly due to limitations of storage, processing power, expected response time, and even available energy. Efficient mining approaches aim at minimizing the number of scans they perform over the objects before generating the final output. Thus, both the needed storage and the required processing power are limited such that the final output that includes objects $o_i, o_{i+1}, \ldots o_j$ is generated before the arrival of object o_{j+1} at timestamp t_{j+1}.

Figure 2.1 gives some examples about real world application that produce data streams. Most of these scenarios are covered within the scope of the algorithms presented in this chapter. Figure 2.1a shows an example about wired streaming data that monitor some flowing phenomenon like network traffic data, click streams, or airport camera monitoring. Figure 2.1b presents a visualization of streaming tweets with a certain tag and within a certain time using the *Streamgraph* framework [6]. Figure 2.1c depicts an application of a wireless sensor network deployment

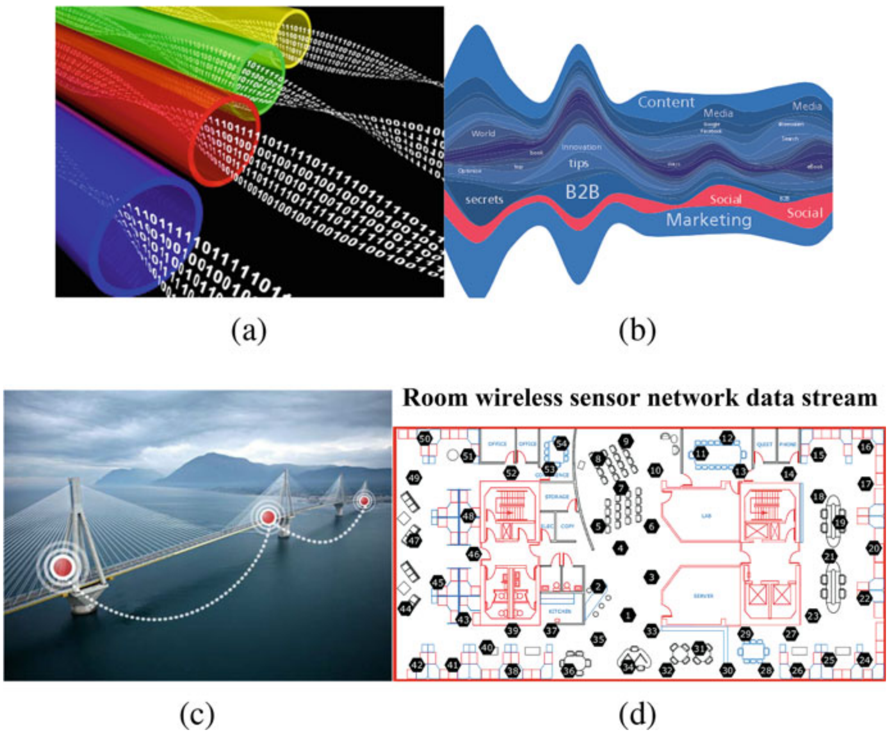


Fig. 2.1 Reference [12]. Examples of big data streams

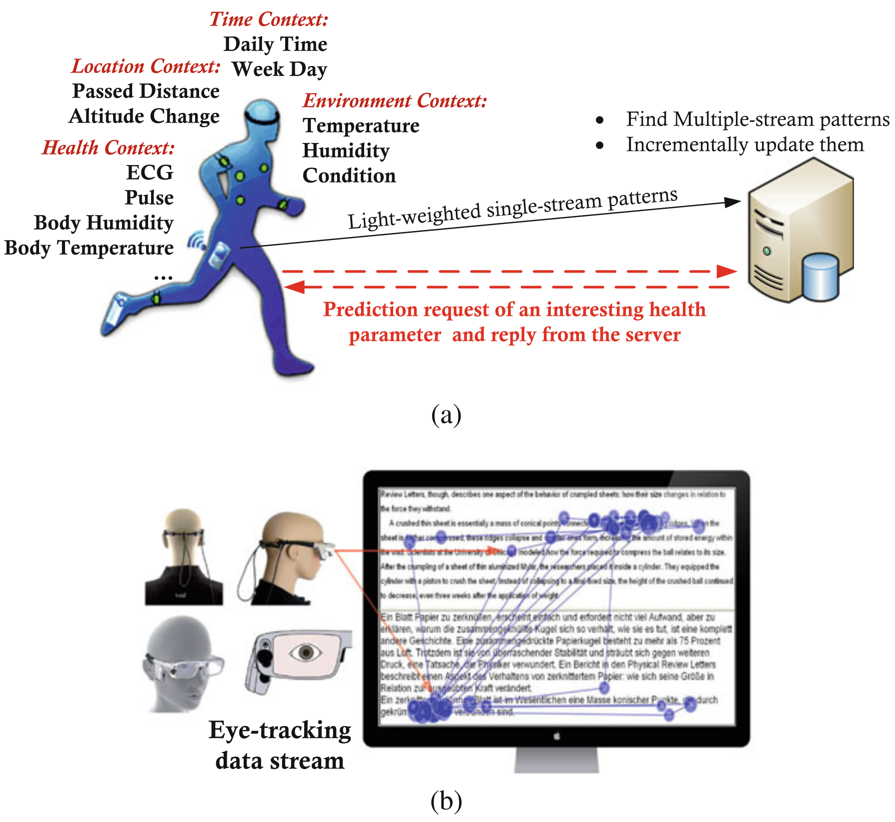


Fig. 2.2 Reference [12]. Two applications of mining body-generated streaming data. (**a**) In a health care scenario [13] and (**b**) in a translation scenario in collaboration with psycholinguists in the humanities area [21]

in a bridge for surveillance or load observation. Sensors are producing continuous streams of readings, and experts need to collect a real-time knowledge about the stability of the bridge in the case of emergency, or gather regular reports in the normal case. Similarly, Fig. 2.1d shows an example of sensors collecting temperature, humidity, and light information from multiple offices in Intel Berkeley Research Lab [10]. Such sensors are usually of limited storage, processing power, and battery life. In Fig. 2.2a, a body sensor network is producing multiple streams about the health status of the runner. Other sensors are collecting streams of other contextual information like the weather and location information. These can be processed on a local mobile device or a remote server to gain, for instance, some knowledge about the near-future status. Figure 2.2b presents another type of sensor streaming data where an eye-tracking system is used to record the duration and the position of each eye fixation over the monitor during a human reading or writing process. One task could be here finding interesting patterns that represent important correlations between eye gazes and key strokes [21].

Stream clustering aims at detecting clusters that are formed out of the evolving streaming objects. These clusters must be *continuously* updated as the stream emerges to follow the current distribution of the data. These clusters represent mainly the gained knowledge out of the clustering task. In this chapter, advanced stream clustering models are introduced. These models are mainly motivated by the basic challenges that we have observed for clustering of streaming data in real world scenarios, particularly sensor streaming data (cf. Fig. 2.1).

2.3 Challenges of Stream Clustering of Big Data

Designing stream clustering approaches has some unique special challenges. We list in the following the different paradigms that make it challenging to design a stream clustering approach.

2.3.1 Adaptation to the Stream Changes and Outlier Awareness

The algorithm must incrementally cluster the stream data points to detect evolving clusters over the time, while forgetting outdated data. New trends of the data must be detected at the same time of their appearance. Nevertheless, the algorithm must be able to distinguish new trends of the stream from outliers. Fulfilling the up-to-date requirement contradicts the outlier awareness one. Thus, meeting this tradeoff is one of the basic challenges of any stream clustering algorithm.

2.3.2 Storage Awareness and High Clustering Quality

Due to the huge sizes and high speeds of streaming data, any clustering algorithm must perform as few passages over the objects as possible. In most cases, the application and the storage limitations allow only for a single passage. However, high-quality clustering results are requested to make the desired knowledge out of the data stream. Most static clustering models tend to deliver an initial, sometimes random, clustering solution and then optimize it by revisiting the objects to maximize some similarity function. Although such multiple-passages possibility does not exist for streaming algorithms, the requirement of an optimized, high-quality clustering does still exist.

2.3.3 Efficient Handling of High-Dimensional, Different-Density Streaming Objects

The current huge increase of the sizes of data was accompanied with a similar boost in their number of dimensions. This applies of course to streaming data too. For such kinds of data with higher dimensions, distances between the objects grow more and more alike due to an effect termed *curse of dimensionality* [4]. According to this effect, applying traditional clustering algorithms in the full-space merely will result in considering almost all objects as outliers, as the distances between them grow exponentially with their dimensionality d. The latter fact motivated the research in the area of *subspace clustering* over static data in the last decade, which searches for clusters in all of the $2^d - 1$ subspaces of the data by excluding a subgroup of the dimensions at each step. Apparently, this implies higher complexity of the algorithm even for static data, which makes it even more challenging when considering streaming data.

Additionally, as the stream evolves, the number, the density, and the shapes of clusters may dramatically change. Thus, assuming a certain number of clusters like in k-means-based clustering models or setting a static density threshold as in the DBSCAN-based clustering models is not convenient for a stream clustering approach. A self-adjustment to the different densities of the data is strongly needed while designing a stream clustering algorithm. Again, this requirement is in conflict with the storage awareness necessity.

2.3.4 Flexibility to Varying Time Allowances Between Streaming Objects

An additional, natural characteristic of data streams (e.g., sensor data) is the fluctuating speed rate. Streaming data objects arrive usually with different time allowances between them, although the application settings would assume a constant stream speed. Available stream clustering approaches, called budget algorithms in this context, strongly restrict their model size to handle minimal time allowance to be on the safe side (cf. Fig. 2.6). In the case of reduced stream speed, the algorithm remains idle during the rest of the time, till the next streaming object arrives. *Anytime mining algorithms*, designed recently for static data, try to make use of any given amount time to deliver some result. Longer given times imply higher clustering quality. This idea was adopted for clustering streaming data. Although this setting can be seen as an opportunity for improving the clustering quality rather than a challenge, it is not trivial to have a flexible algorithmic model that is able to deliver some result even with very fast streams.

2.3.5 Energy Awareness and Lightweight Clustering of Sensor Data Streams

Wireless sensor nodes are equipped with a small processing device, a tiny memory, and a small battery in addition to the sensing unit [27]. This encouraged the research in the area of in-sensor-network mining, where the nodes do some preprocessing of the sensed data instead of simply forwarding it. In many of these applications, sensor nodes are distributed in unreachable areas without a cheap possibility of changing the battery. Thus, the usability time of the node is bounded by the battery lifetime. In this manner, besides the previously mentioned challenges, clustering sensor streaming data has to carefully consume the processing and energy resources. In fact, clustering and aggregation approaches are used within wireless sensor networks to save energy by preprocessing the data in the node, and forwarding the relevant ones merely.

2.4 Recent Contributions in the Field of Efficient Clustering of Big Data Streams

In this section, we present, at a high level of abstraction, novel, efficient stream clustering algorithms that consider all of the above challenges mentioned in Sect. 2.3. These contributions [12] are structured in the following four subsections. In Sect. 2.4.1, we present novel high-dimensional density-based stream clustering techniques. In Sect. 2.4.2, we introduce advanced anytime stream clustering approaches. In Sect. 2.4.3, we present efficient methods for clustering sensor data and aggregating sensor nodes. Finally, in Sect. 2.4.4, we present unique subspace stream clustering framework as well as the subspace cluster mapping evaluation measure. In all of the following subsections, the first and the second challenges mentioned in Sects. 2.3.1 and 2.3.2 are carefully considered. Each of the rest of the challenges (Sects. 2.3.3–2.3.5) is the main focus in one of the following subsections, as we will explain.

2.4.1 High-Dimensional, Density-Based Stream Clustering Algorithms

In this line of research to address big data stream clustering, we refer to three density-based stream clustering algorithms. Here, the third challenge mentioned in Sect. 2.3.3 is mainly considered.

In [18], an efficient projected stream clustering algorithm called *PreDeCon-Stream* is introduced for handling high-dimensional, noisy, evolving data streams. This technique is based on a two-phase model (cf. Fig. 2.3). The first phase repre-

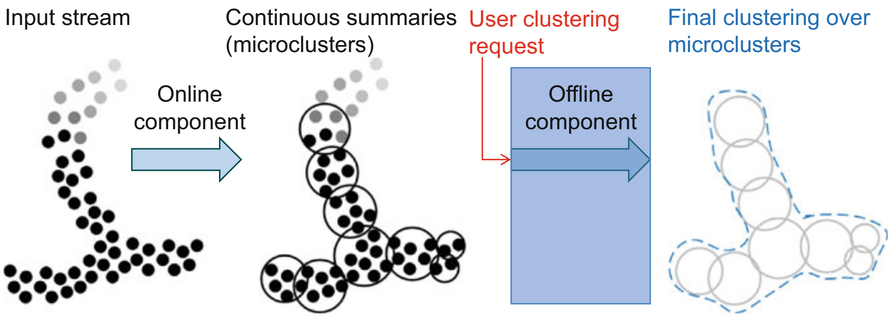

Input stream
Online component
Continuous summaries (microclusters)
User clustering request
Offline component
Final clustering over microclusters

Fig. 2.3 Reference [12]. The online–offline model of stream clustering algorithms. Decayed input objects have lighter colors than recent ones

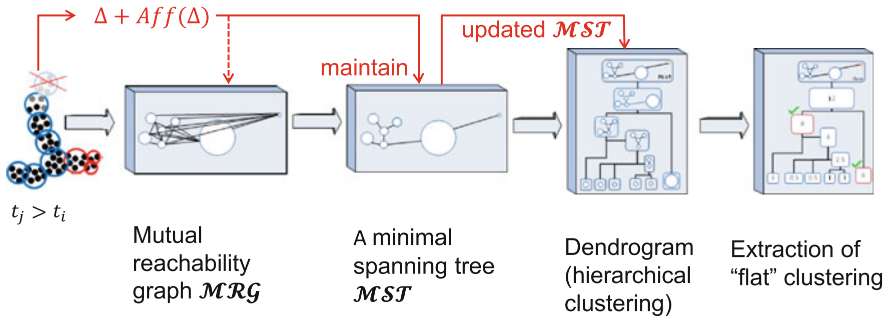

$\Delta + Aff(\Delta)$
maintain
updated $\mathcal{MST}$
$t_j > t_i$
Mutual reachability graph $\mathcal{MRG}$
A minimal spanning tree $\mathcal{MST}$
Dendrogram (hierarchical clustering)
Extraction of "flat" clustering

Fig. 2.4 Reference [12]. The steps of HASTREAM algorithm [19]. The incremental part is explained in red arrows to maintain the clustering at timestamp t_j after the red insertions and deletions of microclusters introduced to the old ones from timestamp t_i

sents the process of the online maintenance of data summaries, called microclusters, that are then passed to an offline phase for generating the final clustering. The technique works on incrementally updating the output of the online phase stored in a microcluster structure. Taking those microclusters that are fading out over time into consideration speeds up the process of assigning new data points to the existing clusters. The algorithm localizes the change to the previous clustering result, and smartly uses a clustering validity interval to make an efficient offline phase.

In *HASTREAM* [19], a hierarchical, self-adaptive, density-based stream clustering model is contributed (cf. Fig. 2.4). The algorithm focuses on smoothly detecting the varying number, densities, and shapes of the streaming clusters. A cluster stability measure is applied over the summaries of the streaming data (the microclusters in Fig. 2.3), to extract the most stable offline clustering. Algorithm 1 gives a pseudo-code of HASTREAM (cf. Fig. 2.4 and [19] for further details). In order to improve the efficiency of the suggested model in the offline phase,

Algorithm 1 **HASTREAM**(DataStream ds, $minClusterWeight$, bool $incUpdate$)

```
1:  initialization phase
2:  repeat
3:      get next point $o_i \in ds$ with current timestamp $t_i$;
4:      insert $o_i$ in the microclusters using online parameter settings;
5:      if ($t_i$ mod $updateFrequency == 0$) then
6:          if $incUpdate$ then
7:              incrementally update the minimal spanning tree $MST$ by maintaining it;
8:          else
9:              compute the mutual reachability graph $MRG$ and corresponding $MST$ from scratch
10:         end if
11:         $HC \leftarrow$ extractHierarchicalClusters($MST$, $minClusterWeight$);
12:         $C \leftarrow$ extractFlatClustering($HC$);
13:         return $C$;
14:     end if
15: until data stream terminates
```

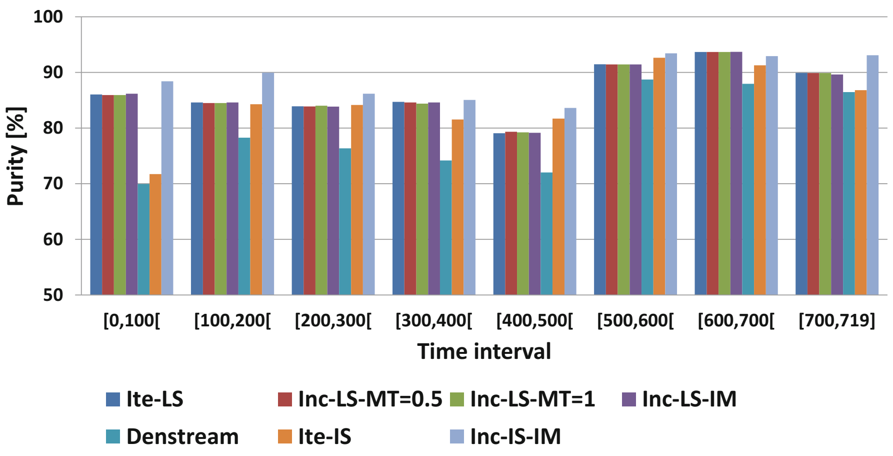


Fig. 2.5 Purity of detected clustered in five variants of HASTREAM [19] compared to DenStream [7] for the physiological dataset [26]. Timestamps are $\times 1000$

some methods from the graph theory are adopted and others were contributed, to incrementally update a minimal spanning tree of microclusters (cf. the red arrows in Fig. 2.4). This tree is used to continuously extract the final clustering, by localizing the changes that appeared in the stream, and maintaining the affected parts merely. The averaged purity for this dataset is shown in Fig. 2.5. All four variants of HASTREAM [19] have a higher averaged purity than that of DenStream [7] over the physiological dataset [26].

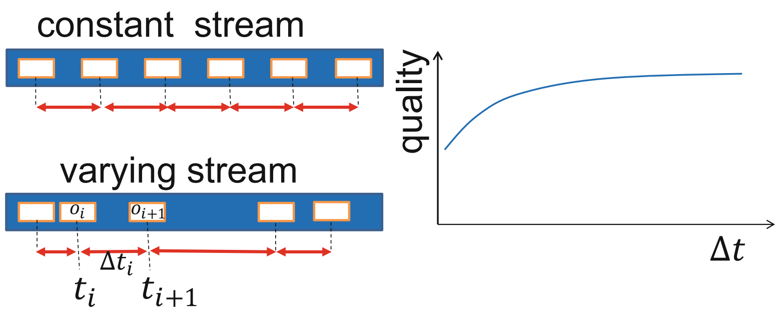


Fig. 2.6 The concept of anytime stream mining algorithms [12]

2.4.2 *Advanced Anytime Stream Clustering Algorithms*

By considering all other challenges, the main focus of the two algorithms presented in this section are the third and the fourth challenges mentioned in Sects. 2.3.3 and 2.3.4. Anytime algorithms build upon the realistic assumption of the varying time allowances between streaming objects (cf. Fig. 2.6). They aim at increasing the quality of their output if they were given more time (i.e., the time allowance Δt is bigger) instead of being idle as in traditional algorithms.

The *LiarTree* algorithm [17] is contributed on the online phase (cf. Fig. 2.3) to provide precise stream summaries and to effectively handle noise, drift, and novelty at any given time. It is proven that the runtime of the anytime algorithm is logarithmic in the size of the maintained model opposed to a linear time complexity often observed in previous approaches. The main contributions of this technique are enabling the anytime concept to fast adapt to the new trends of the data, filtering noise and keeping a logarithmic complexity.

In the *SubClusTree* algorithm [20], even another complexity dimension to the problem addressed in LiarTree [17] is added. The high-dimensionality paradigm of big streaming data (cf. Sect. 2.3.3) is considered together with the varying arrival times and the streaming aspects of the data (cf. Sect. 2.3.4). SubClusTree is a subspace anytime stream clustering algorithm, that can flexibly adapt to the different stream speeds and makes the best use of available time to provide a high-quality subspace clustering. It uses compact index structures to maintain stream summaries in the subspaces in an online fashion. It uses flexible grids to efficiently distinguish the relevant subspaces (i.e., subspaces with clusters) from irrelevant ones. Algorithm 2 contains a pseudo-code of SubClusTree. An object is inserted in all one-dimensional trees and if there is more time, the object is inserted into following most potential higher-dimensional tree.

In Fig. 2.7, one can obviously observe the anytime effect of SubClusTree using a 25-dimensional dataset with five clusters hidden in only 13 relevant dimensions out of the 25. For a very fast stream, a lower clustering quality is achieved. It can be seen that an average interval of 50 steps between objects is very close to the best possible value.

Algorithm 2 SubClusTree

```
1: Initialization: store subspace trees in the bitmap with bit-vector as key
2: repeat
3:     insert $o_i$ and update the global cluster feature to time $t_i$;
4:     for $j = 1$ to $d$ (number of dimensions) do
5:         insert $o_{ij}$ into the one-dimensional tree of subspace $j$;
6:     end for
7:     if ($t_i$ mod $updateFrequency == 0$) then
8:         create $candidateTrees$ and rank them according to their expected potential;
9:         remove trees with insufficient $potential$; // but keep 1-dimensional trees
10:    end if
11:    while next object $o_{i+1}$ did not arrive yet and $moreTreesAvailable$ do
12:        insert $o_i$ into next subspace tree
13:    end while
14: until data stream terminates
```

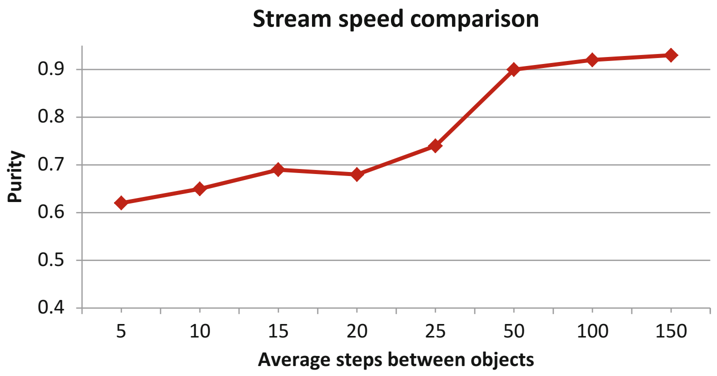


Fig. 2.7 Reference [12]. Average purity achieved by varying the inter-arrival times of the objects Δt

2.4.3 Energy-Efficient Algorithms for Aggregating and Clustering Sensor Streaming Data

In the three algorithms contributed in this line of research, the main focus is the fifth challenge mentioned in Sect. 2.3.5, while keeping the main challenges from Sects. 2.3.1 and 2.3.2 in mind.

The *EDISKCO* algorithm [15] is an Energy-Efficient Distributed In-Sensor-Network K-Center Clustering algorithm with Outliers. Sensor networks have limited resources in terms of available memory and residual energy. As a dominating energy consuming task, the communication between the node and the sink has to be reduced for a better energy efficiency. Considering memory, one has to reduce the amount of stored information on each sensor node. EDISKCO performs an outlier-aware k-center clustering [11] over the sensed streaming data in each node and forwards the clustering result to the neighboring coordinator node. For that,

Algorithm 3 EDISKCO on Node side

```
1: **Initialization:** Select the first *InitPts* objects from the stream;
2: perform an offline $k$-center clustering over the *InitPts* objects;
3: send the $k$ centers and the cluster radius $R$ to the coordinator
4: **repeat**
5:     insert $o_i$ in available clusters and update cluster centers $c_1, c_2, \ldots c_k$;
6:     if changed, send the new centers to the coordinator;
7:     **if** $o_i$ does not fit in any available cluster **and** number of clusters $== k$ **then**
8:         recluster after getting the coordinator acknowledgment;
9:     **end if**
10: **until** data stream terminates
```

Algorithm 4 EDISKCO on Coordinator side

```
1: **repeat**
2:     receive $k$-center solutions from all nodes and perform another $k$-center clustering on them;
3:     Acknowledge reclustering by broadcasting the biggest current radius from all nodes;
4:     Change coordinator w.r.t. residual energy;
5: **until** data stream terminates
```

each node consumes considerably less energy than the cost of forwarding all of its readings to the sink. The update from the node to the coordinator happens only upon a certain deviation of the cluster radii, controlled by an ϵ threshold, or upon the birth or the deletion of a cluster. One node is selected as a coordinator from each spatially correlated subgroup of the nodes, depending on the amount of the residual energy. The coordinator performs another k-center clustering over the multiple clustering solutions arriving from its neighboring nodes and forwards the solution to the far sink. These are major contributions as one can perform a single passage over the data by using a $O(k)$ storage over the nodes and getting finally a high clustering quality of a $(4 + \epsilon)$-approximation to the optimal clustering. But, the main contribution is performing up to 95% less communication tasks on the nodes compared to a state-of-the-art technique. Thus, huge savings of energy are achieved. Algorithm 3 summarizes the node side procedure of EDISKCO while Algorithm 4 abstracts the approach on the coordinator side.

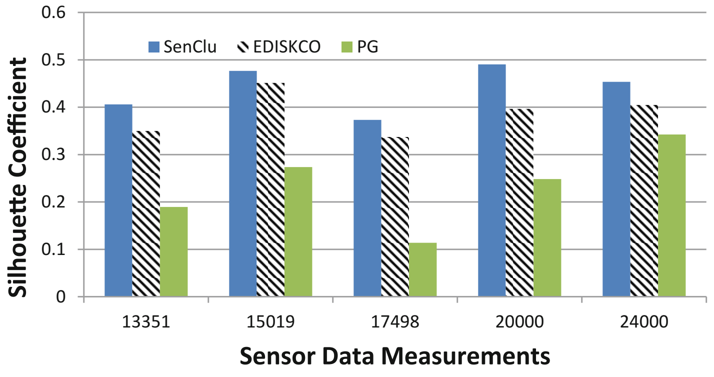
A weighted version of EDISKCO, called *SenClu*, is contributed in [14]. It guarantees a faster adaptation to the new trends of the drifting data streams. The technique gives more importance to new data points, while slowly forgetting older ones by giving them less weight. To achieve this, a novel, light-weighted decaying function is contributed, that can be implemented on the tiny processing unit and works on the limited storage capacity of sensor nodes. SenClu achieves even better clustering quality than EDISKCO while draining almost the same amount of energy.

One can see from Table 2.1 that on the real dataset (i9-Sensor Dataset), SenClu [14] consumes less than two Joules more than EDISKCO [15], and absorbs considerably less energy than the state-of-the-art competitor PG [8]. When using the Physiological Sensor Dataset [26], SenClu consumes less energy than both competitors. Figure 2.8a shows that SenClu [14] and EDISKCO [15] always have a better clustering quality than PG [8] on the node side. Because PG is more sensitive

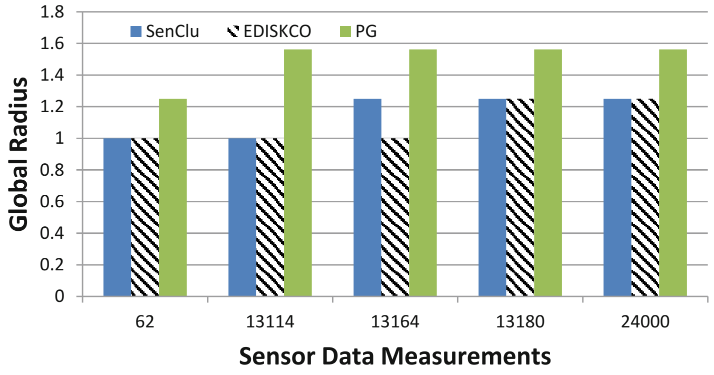
Table 2.1 Average energy consumption in Joule of a single node in the network by the end of each dataset when using SenClu, EDISKCO, and PG

Dataset	Size	Nodes	SenClu [14]	EDISKCO [15]	PG [8]
i9-Sensor	40,589	19	28,770.2	**28,768.38**	28,792.5
Physio [26]	24,000	12	**17,074.3**	17,074.4	17,078.9

Lowest energy consumption is in bold



(a)



(b)

Fig. 2.8 Reference [12]. The clustering quality using the Real Physiological Sensor Dataset [26] over different parts of the input stream data. (**a**) Silhouette coefficient (higher is better), (**b**) R_{global} (lower is better)

to noise than SenClu and EDISKCO, it is performing considerably worse than the others on this relatively noisy dataset. Figure 2.8b is showing that on the node side, SenClu is having most of the time the same global radius as EDISKCO. Only for a short time, SenClu is having a bigger radius than EDISKCO.

A further challenge for aggregating streaming data within the sensor network is tackled. The physical clustering of sensor nodes depending on their similarity is considered in the presented *ECLUN* algorithm [16]. The readings of a carefully selected representative node are used to simulate the measurements of similar nodes. While the recent approaches concentrated on the full-dimensionality correlation between the readings, ECLUN selects the representatives depending on the subspace correlation between some attributes of the measurements as well as the spatial similarity. Additionally, the usage of energy between the nodes is uniformly distributed, and, thus, the cases of single-node clusters are handled by changing representatives according to the residual energy. This results in a longer lifetime of the whole sensor network as nodes die close to each other.

2.4.4 A Framework and an Evaluation Measure for Subspace Stream Clustering

This section presents some contributions mainly in the *Evaluation and Visualization* step of the KDD (Knowledge Discovery in Databases) process.

The first subspace clustering evaluation framework over data streams, called *Subspace MOA*, is presented in [22]. This open-source framework is based on the MOA stream mining framework [5], and has three phases (cf. Fig. 2.9). In the online phase, users have the possibility to select one of three most famous summarization techniques to form the microclusters. Upon a user request for a final clustering, the regeneration phase constructs the data objects out of the current microclusters. Then, in the offline phase, one of five subspace clustering algorithms can be selected. In addition to the previous combinations, the framework contains available projected stream clustering algorithms like PreDeConStream [18] and HDDStream [25]. The framework is supported with a subspace stream generator, a visualization interface, and various subspace clustering evaluation measures. With the increase of the size of high-dimensional data, applying traditional subspace clustering algorithms is impossible due to their exponential complexities. Figure 2.10 shows, for instance,

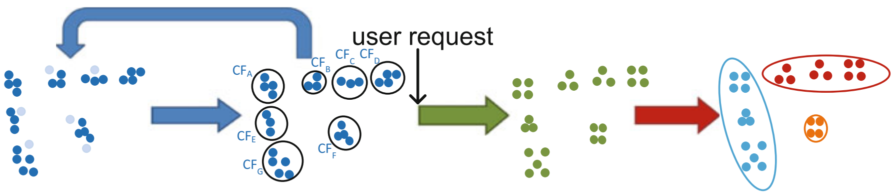
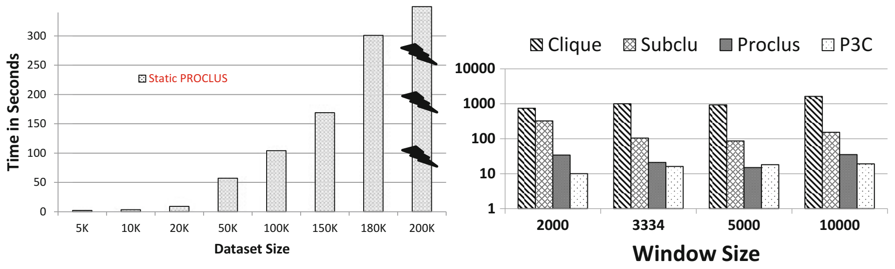


Fig. 2.9 Reference [12]. *Subspace MOA* model for stream subspace clustering for big data. The blue arrows represent the online phase, the green arrow represents the regeneration phase, and the red arrow represents the offline phase



(a) (b)

Fig. 2.10 Reference [12]. (**a**): The runtime of the *static* subspace clustering algorithm: PROCLUS [1]. Beginning from a sub-dataset size of 200 K objects only, the algorithm fails to successfully finish the running. (**b**): A successful run of PROCLUS (and other subspace clustering algorithms Clique [3], SubClu [23], and P3C [24]) over the whole KDD dataset [9] when applying the *streaming* PROCLUS using Subspace MOA with CluStream [2] in the online phase on the same machine

that when using the same machine, it was only possible by using Subspace MOA, to get a relatively large dataset clustered with a subspace clustering algorithm (PROCLUS [1]).

In [22], a novel external evaluation measure for stream subspace clustering algorithms called *SubCMM*: Subspace Cluster Mapping Measure is contributed. SubCMM is able to handle errors caused by emerging, moving, or splitting subspace clusters. This first evaluation measure that is designed to reflect the quality of stream subspace algorithms is directly integrated in the Subspace MOA framework. This new measure was extensively compared against state-of-the-art full-space stream clustering evaluation measures. The experimental evaluation, performed using the Subspace MOA framework, depicts the ability of *SubCMM* to reflect the different changes happening in the subspaces of the evolving stream.

2.5 Conclusion

In this chapter, we have mainly addressed the following three v's of big data: velocity, volume, and variety. Various streaming data applications, with huge volumes and varying velocities, were considered in different scenarios and applications.

A list of the recent challenges that face the designer of a stream clustering algorithm was shown and deeply discussed. Finally, some recent contributions on four main research lines in the area of big data stream mining were presented and their fulfillment to the design requirements was highlighted. Table 2.2 summarizes the main properties of some stream clustering algorithms mentioned in this chapter.

Table 2.2 Properties of the different stream clustering algorithms discussed in this chapter (*na* = not applicable, ∼ = indirectly available)

Algorithm	Online–offline model	Density-based (in online phase)	Density-based (in offline phase)	Hierarchical	Outlier-aware (cf. Sect. 2.3.1)	Density-adaptive (cf. Sect. 2.3.3)	Anytime (cf. Sect. 2.3.4)	Lightweight clustering (cf. Sect. 2.3.5)	Incremental (in offline phase)	Overlapping clusters and subspaces	Data structure of micro-clusters
Full-space stream clustering algorithms											
CluStream [2]	✓									*na*	*Pyramidal time frame*
DenStream [7]	✓	✓	✓		✓					*na*	*List*
LiarTree [17]	✓	✓	*na*		✓		✓			*na*	*Index*
HASTREAM [19]	✓	✓	✓	✓	✓	✓				*na*	*Index* or *List*
EDISKCO [15]	∼			∼	✓			✓		*na*	*List*
SenClu [14]	∼			∼	✓			✓	✓	*na*	*List*
Subspace stream clustering algorithms											
PreDeConStream [18]	✓	✓	✓		✓						*List*
HDDStream [25]	✓	✓	✓		✓						*List*
SubClusTree [20]	✓	✓	*na*		✓		✓			✓	*Index*
ECLUN [16]	∼			∼	✓			✓	✓	*na*	*List*

References

1. C.C. Aggarwal, J.L. Wolf, P.S. Yu, C. Procopiuc, J.S. Park, Fast algorithms for projected clustering, in *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, SIGMOD '99, pp. 61–72 (ACM, New York, 1999)
2. C.C. Aggarwal, J. Han, J. Wang, P.S. Yu, A framework for clustering evolving data streams, in *Proceedings of the 29th International Conference on Very Large Data Bases*, VLDB '03, pp. 81–92 (VLDB Endowment, Los Angeles, 2003)
3. R. Agrawal, J. Gehrke, D. Gunopulos, P. Raghavan, Automatic subspace clustering of high dimensional data for data mining applications, in *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, SIGMOD '98, pp. 94–105 (ACM, New York , 1998)
4. K.S. Beyer, J. Goldstein, R. Ramakrishnan, U. Shaft, When is "nearest neighbor" meaningful? in *Proceedings of the 7th International Conference on Database Theory*, ICDT '99, pp. 217–235 (Springer, Berlin, 1999)
5. A. Bifet, G. Holmes, R. Kirkby, B. Pfahringer, MOA: Massive online analysis. J. Mach. Learn. Res. **11**, 1601–1604 (2010)
6. L. Byron, M. Wattenberg, Stacked graphs - geometry & aesthetics. IEEE Trans. Vis. Comput. Graph. **14**(6), 1245–1252 (2008)
7. F. Cao, M. Ester, W. Qian, A. Zhou, Density-based clustering over an evolving data stream with noise, in *Proceedings of the 6th SIAM International Conference on Data Mining*, SDM '06, pp. 328–339 (2006)
8. G. Cormode, S. Muthukrishnan, W. Zhuang, Conquering the divide: continuous clustering of distributed data streams, in *IEEE 23rd International Conference on Data Engineering*, ICDE '07, pp. 1036–1045 (IEEE Computer Society, Washington, 2007)
9. N.I. Dataset, KDD cup data (1999). http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html
10. I. Dataset, Dataset of Intel Berkeley Research Lab (2004) URL: http://db.csail.mit.edu/labdata/labdata.html
11. S. Guha, Tight results for clustering and summarizing data streams, in *Proceedings of the 12th International Conference on Database Theory*, ICDT '09, pp. 268–275 (ACM, New York, 2009)
12. M. Hassani, Efficient clustering of big data streams, Ph.D. thesis, RWTH Aachen University, 2015
13. M. Hassani, T. Seidl, Towards a mobile health context prediction: sequential pattern mining in multiple streams, in *Proceedings of the IEEE 12th International Conference on Mobile Data Management*, vol. 2 of *MDM '11*, pp. 55–57 (IEEE Computer Society, Washington, 2011)
14. M. Hassani, T. Seidl, Distributed weighted clustering of evolving sensor data streams with noise. J. Digit. Inf. Manag. **10**(6), 410–420 (2012)
15. M. Hassani, E. Müller, T. Seidl, EDISKCO: energy efficient distributed in-sensor-network k-center clustering with outliers, in *Proceedings of the 3rd International Workshop on Knowledge Discovery from Sensor Data*, SensorKDD '09 @KDD '09, pp. 39–48 (ACM, New York, 2009)
16. M. Hassani, E. Müller, P. Spaus, A. Faqolli, T. Palpanas, T. Seidl, Self-organizing energy aware clustering of nodes in sensor networks using relevant attributes, in *Proceedings of the 4th International Workshop on Knowledge Discovery from Sensor Data*, SensorKDD '10 @KDD '10, pp. 39–48 (ACM, New York, 2010)
17. M. Hassani, P. Kranen, T. Seidl, Precise anytime clustering of noisy sensor data with logarithmic complexity, in *Proceedings of the 5th International Workshop on Knowledge Discovery from Sensor Data*, SensorKDD '11 @KDD '11, pp. 52–60 (ACM, New York, 2011)
18. M. Hassani, P. Spaus, M.M. Gaber, T. Seidl, Density-based projected clustering of data streams, in *Proceedings of the 6th International Conference on Scalable Uncertainty Management*, SUM '12, pp. 311–324 (2012)
19. M. Hassani, P. Spaus, T. Seidl, Adaptive multiple-resolution stream clustering, in *Proceedings of the 10th International Conference on Machine Learning and Data Mining*, MLDM '14, pp. 134–148 (2014)

20. M. Hassani, P. Kranen, R. Saini, T. Seidl, Subspace anytime stream clustering, in *Proceedings of the 26th Conference on Scientific and Statistical Database Management*, SSDBM' 14, p. 37 (2014)
21. M. Hassani, C. Beecks, D. Töws, T. Serbina, M. Haberstroh, P. Niemietz, S. Jeschke, S. Neumann, T. Seidl, Sequential pattern mining of multimodal streams in the humanities, in *Datenbanksysteme für Business, Technologie und Web (BTW), 16. Fachtagung des GI-Fachbereichs "Datenbanken und Informationssysteme" (DBIS), 4.-6.3.2015 in Hamburg, Germany. Proceedings*, pp. 683–686 (2015)
22. M. Hassani, Y. Kim, S. Choi, T. Seidl, Subspace clustering of data streams: new algorithms and effective evaluation measures. J. Intell. Inf. Syst. **45**(3), 319–335 (2015)
23. K. Kailing, H.-P. Kriegel, P. Kröger, Density-connected subspace clustering for high-dimensional data, in *Proceedings of the SIAM International Conference on Data Mining*, SDM '04, pp. 246–257 (2004)
24. G. Moise, J. Sander, M. Ester, P3C: a robust projected clustering algorithm, in *Proceedings of the 6th IEEE International Conference on Data Mining*, ICDM '07, pp. 414–425 (IEEE, Piscataway, 2006)
25. I. Ntoutsi, A. Zimek, T. Palpanas, P. Kröger, H.-P. Kriegel, Density-based projected clustering over high dimensional data streams, in *Proceedings of the 12th SIAM International Conference on Data Mining*, SDM '12, pp. 987–998 (2012)
26. Physiological dataset. http://www.cs.purdue.edu/commugrate/data/2004icml/
27. J. Polastre, R. Szewczyk, D. Culler, Telos: enabling ultra-low power wireless research, in *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks*, IPSN '05 (IEEE Press, , Piscataway, 2005)