SECLEDS: Sequence Clustering in Evolving Data Streams via Multiple Medoids and Medoid Voting

Azqa Nadeem(✉) and Sicco Verwer

Delft University of Technology, Delft, The Netherlands
{azqa.nadeem,s.e.verwer}@tudelft.nl

Abstract. Sequence clustering in a streaming environment is challenging because it is computationally expensive, and the sequences may evolve over time. K-medoids or Partitioning Around Medoids (PAM) is commonly used to cluster sequences since it supports alignment-based distances, and the k-centers being actual data items helps with cluster interpretability. However, offline k-medoids has no support for concept drift, while also being prohibitively expensive for clustering data streams. We therefore propose SECLEDS, a streaming variant of the k-medoids algorithm *with constant memory footprint*. SECLEDS has two unique properties: i) it uses multiple medoids per cluster, producing stable high-quality clusters, and ii) it handles concept drift using an intuitive *Medoid Voting scheme* for approximating cluster distances. Unlike existing adaptive algorithms that create new clusters for new concepts, SECLEDS follows a fundamentally different approach, where the clusters themselves evolve with an evolving stream. Using real and synthetic datasets, we empirically demonstrate that SECLEDS produces high-quality clusters regardless of drift, stream size, data dimensionality, and number of clusters. We compare against three popular stream and batch clustering algorithms. The state-of-the-art BanditPAM is used as an offline benchmark. SECLEDS achieves comparable F1 score to BanditPAM while reducing the number of required distance computations by 83.7%. Importantly, SECLEDS outperforms all baselines by 138.7% when the stream contains drift. We also cluster real network traffic, and provide evidence that SECLEDS can support network bandwidths of up to 1.08 Gbps while using the (expensive) dynamic time warping distance.

Keywords: Sequence clustering · K-medoids · Data streams · Concept drift · Network traffic

1 Introduction

Stream clustering is the problem of clustering a potentially unbounded stream of items in a single pass, where the items arrive sequentially without any particular

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/978-3-031-26387-3_10.

M.-R. Amini et al. (Eds.): ECML PKDD 2022, LNAI 13713, pp. 157–173, 2023.
https://doi.org/10.1007/978-3-031-26387-3_10

order, *e.g.,* network traffic, financial transactions, and sensor data. Stream clustering algorithms must have low memory overhead, be computationally efficient, and robust to concept drift, *i.e.,* evolving data distributions [25]. Maintaining high cluster quality in a fully online setting is extremely difficult. Therefore, hybrid online-offline algorithms are popular among existing approaches, *e.g.,* CluStream [2], StreamKM++ [1], DenStream [6], BIRCH [31]. These algorithms have an online component that summarizes the data stream, and an offline component that periodically uses that information to create the final clusters. There also exist algorithms that store part of the stream for handling outliers, *e.g.,* BOCEDS [15], MDSC [9]. Existing stream clustering algorithms handle concept drift by having variable number of clusters: they add new clusters for newly observed behavior and discard clusters that contain too many old data items. This leads to higher memory requirements for managing buffers and intermediate solutions. Batch clustering algorithms can also be used in a streaming setting by considering a batch size of one, *e.g.,* Minibatch k-means [24]. However, they start to under-perform when the stream contains drift.

In recent years, sequential data has increasingly become popular because of the powerful insights that it provides regarding behavior analytics [5], *e.g.,* for attacker strategy profiling [21], fraud detection [13], human activity recognition [7]. Clustering sequences in an offline setting is challenging in itself because sequences are often out-of-sync, requiring expensive alignment-based distance measures, which are often not supported by many clustering algorithms. K-medoids or Partitioning Around Medoids (PAM) has often been used to cluster sequences because the `k`-centers are represented by actual data items, called medoids or prototypes [27,28]. This has multiple benefits: i) it makes the cluster interpretation simpler; ii) it enables the use of non-metric distances such as dynamic time warping (DTW); and iii) it allows to estimate exact storage requirements based on the `k`-fixed clusters. Although the state-of-the-art offline k-medoids algorithms, *i.e.,* FastPAM1 [23] and BanditPAM [26] have reduced the runtime complexity to $\mathcal{O}(nlogn)$, they are still not efficient enough to be used in streaming settings, and the cluster quality will degrade over time as the stream evolves. To the best of our knowledge, there exists no streaming version of the k-medoids algorithm that can efficiently cluster sequential data.

Contributions. In this paper, we propose *SECLEDS*, a lightweight streaming version of the k-medoids algorithm with *constant memory footprint*. SECLEDS has two unique properties: Firstly, it uses `p`-medoids per cluster to maintain stable high-quality clusters. Note the difference from IMMFC [30], which uses the information of multiple medoids in independent sub-solutions to select the final medoids. We initialize the `p`-medoids using a non-uniform sampling strategy similar to k-means++. Secondly, a Medoid Voting scheme is used to estimate a cluster's center of mass. The offline k-medoids has a Swap step that tests each point in a cluster to determine the next medoid. SECLEDS cannot do this because it does not store any part of the stream. Instead, it maintains votes for each medoid that estimate how representative (valuable) it is given the data seen so far. A user-supplied decay factor enables SECLEDS to slowly forget the votes

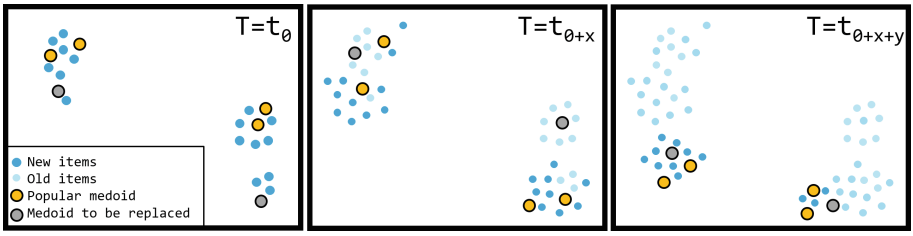


Fig. 1. An illustration of SECLEDS' clusters following an evolving data stream. The medoids close to recent data gain more votes, while the medoids with the least votes are replaced with new data items from the stream. In effect, the k-clusters handle concept drift by capturing different concepts in the stream at different time steps.

regarding past data. The least representative medoids are then replaced with new data items. This way, rather than creating new clusters for new concepts, the k-clusters themselves evolve with the data stream. Figure 1 shows how the clusters follow a data stream as it evolves. Thus, the k-clusters represent different concepts in the stream at different time steps. SECLEDS addresses the following real-world constraints:

I. A runtime efficient medoid-based clustering algorithm with a fixed memory footprint that can handle high-bandwidth data streams,
II. An algorithm that produces high-quality clusters in a streaming environment while being able to deal with concept drift,
III. Accurate sequence clusters using alignment-based distances, and
IV. Minimal parameter settings to support ease-of-use.

Empirical Results. We experiment on several real and synthetic data streams that contain 2D points and univariate sequences. We empirically demonstrate that SECLEDS produces high-quality clusters regardless of drift, stream size, and number of clusters. We use the following state-of-the-art and popular clustering algorithms as baselines: a) Streaming: CluStream, StreamKM++; b) Batch: Minibatch k-means; c) Offline: BanditPAM. Particularly, BanditPAM is used as a benchmark for the best achievable clustering on a static dataset. The results show that i) SECLEDS achieves comparable F1 score to BanditPAM, while reducing the required number of distance computations by 83.7%; ii) SECLEDS outperforms all baselines by 138.7% when the stream contains drift; iii) SECLEDS is faster than BanditPAM and CluStream on most clustering tasks.

We also discuss a use-case from the network security domain where network traffic is often randomly sampled to keep the storage requirements within a predefined budget. Consequently, temporal patterns in the network traffic are lost that could have been useful for downstream tasks, *e.g.,* behavior analytics. We propose a smarter sampling technique that uses medoid-based stream clustering (SECLEDS) to summarize the network traffic: SECLEDS clusters sequences of network traffic and periodically stores the medoids of each cluster, thus reducing the storage needs while preserving temporal patterns in the data. By clustering

real-world network traffic, we provide evidence that SECLEDS (and SECLEDS-dtw) can support network bandwidths of up to 2.79 Gbps (and 1.08 Gbps), respectively. We release SECLEDS as open-source[1].

2 Preliminaries

Stream. Given a sensor that receives an unbounded stream of multi-dimensional data points $X = \{x_1, x_2, \dots\}$ with dimensionality d, arriving at time steps $T = \{t_1, t_2, \dots\}$, a sequential data stream is defined as $S = \{s_1, \dots s_n, \dots\}$, where s_i is a time window w over X such that $s_i = \{x_i, x_{i+1}, \dots x_{i+w}\}$, and y_i is its associated class label. Traditional point clustering considers $w = 1$, while for sequence clustering, we consider $w > 1$. We use two configurations, *i.e.,* $d = 2, w = 1$ (2D point clustering) and $d = 1, w = 100$ (univariate sequence clustering). A case of bivariate variable length sequences is given in appendix.

Concept Drift. Real-world data streams often change unexpectedly over time. This shift alters the statistical properties of their underlying distribution. In machine learning, this is called concept drift [18,32]. Concept drift is typically categorized into four types [17]: (i) *Sudden drift* where a new concept arises abruptly; (ii) *Gradual drift* where an old concept is slowly replaced by a new one; (iii) *Incremental drift* where a concept incrementally turns into another one; and (iv) *Recurring concepts* are old concepts that reappear from time to time. Years of research has gone into developing *concept drift detectors* that either monitor the underlying data distribution, error rate or perform hypothesis testing to trigger model retraining [4,17]. Typical stream clustering algorithms handle concept drift by introducing new clusters for new concepts, and discarding old irrelevant clusters [14]. Although intuitively appealing, this requires user-supplied parameters that define what 'new' means.

3 SECLEDS: Sequence Clustering in Evolving Streams

SECLEDS is a lightweight streaming variant of the classical k-medoids (PAM) algorithm. To support high bandwidth data streams, SECLEDS does not store any part of the stream in memory—it receives an item, assigns it to one of the k-clusters, and then discards it. This way, SECLEDS has a guaranteed constant memory footprint [**I**]. However, this requirement cannot be achieved using the offline Build and Swap steps of PAM. Instead, SECLEDS performs a non-uniform sampling (similar to k-means++) on an initial batch of the stream to initialize the medoids. It also makes use of *multiple medoids per cluster* to provide a stable cluster definition in a streaming setting where noise and concept drift are common properties [**II**].

The efficiency of a chosen distance measure is usually the primary performance bottleneck in sequence clustering. Thus, minimizing the number of distance computations is key to scaling SECLEDS to large data streams. This is

[1] SECLEDS: https://github.com/tudelft-cda-lab/SECLEDS.

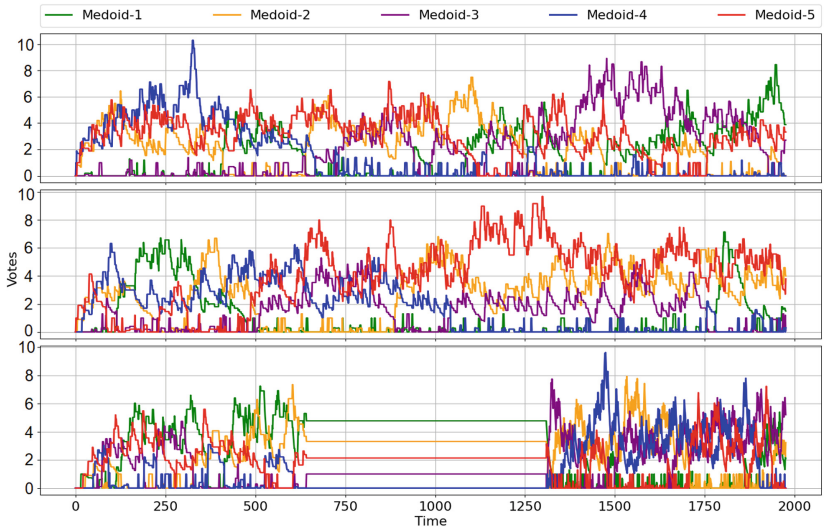


Fig. 2. The effect of cluster initialization and concept drift on medoid selection of a cluster, where $p = 5$. (Top): Given a uniformly distributed stream, every medoid becomes popular at some point. (Middle): For an incrementally drifted stream, the medoid close to drifted data becomes popular. (Bottom): For a class-ordered stream, all clusters start from one class, until one medoid migrates to the correct class. In this case, the correct class is observed from t_{1300}. Medoid-3 migrates first and becomes popular, while medoid-1 migrates last.

achieved by introducing a *Medoid Voting scheme* whose purpose is twofold: (i) it determines the center of mass of a cluster, thus is able to estimate how representative a medoid is given the data seen so far; and (ii) by using this information, old irrelevant medoids are replaced by new ones that are located near recent data. Hence, better medoids can be found without having to perform additional distance computations [**I**]. This also allows SECLEDS to support robust but computationally expensive distance measures specifically meant for sequential data, *e.g.,* dynamic time warping [**III**]. Finally, SECLEDS handles concept drift by regularly *forgetting* past data and occupying newer regions/concepts in the data stream. This is achieved by applying exponential decay λ to the medoid votes at each time step [**II**].

SECLEDS has a modular implementation in Python. The `k`-clusters, `p`-medoids per cluster, and decay rate λ are the only three user-supplied parameters needed for the algorithm, making it useful for exploratory data analysis [**IV**]. We believe these parameters are easier to tune compared to many radius- or density-based hyperparameters in existing clustering algorithms, which require a deeper understanding of the data distribution in advance.

3.1 Stable Cluster Definition via Multiple Medoids

A new data item s is assigned to a cluster *cid* with the least average distance to its medoids. With multiple medoids per cluster, this provides a robust cluster assignment. Additionally, the medoid voting scheme encourages the medoids to represent different sections of a class so they can gain votes: if they are too close together, some of them do not receive votes and get replaced eventually. It also ensures that outliers are quickly replaced because of fewer votes.

Concept drift and cluster initialization determine how the medoids behave. Figure 2 illustrates three scenarios with varying medoid behavior for a single cluster as a function of votes gained over time: (a) Assuming no concept drift, when the stream is roughly evenly shuffled, all the medoids receive uniform votes on average. This is because all medoids are close to parts of the stream at different time steps. At a specific time step, the medoid close to the most amount of recent data becomes popular. The top figure shows that each medoid becomes popular at some point in the stream, indicating that the medoids represent different sections of the underlying class. (b) When the stream incrementally drifts, the cluster follows the evolving stream by replacing the least popular medoids with recent data items from the stream. Since the new medoids are now closer to new data, they gain more votes and become popular. This has roughly the same effect as the first case. (c) When the data arrives one class at a time, all clusters are initialized in a single class. As data from a new class appears, one medoid from the closest cluster migrates to it and starts gaining votes. Over time, the older popular medoids lose their votes because of exponential decay, and eventually migrate to the new class. This is shown from t_{1300} onward in the bottom figure, highlighting the importance of multiple medoids in noisy streams.

3.2 Center of Mass Estimation

The voting scheme provides an estimate of a cluster's center of mass by assigning more votes to recently observed data in the stream S, while exponential decay helps to forget votes regarding older data. Without decay, older clusters with popular medoids never evolve. Thus, these properties help to replace irrelevant medoids, *e.g.,* those that are located i) close to the least amount of recent data, or ii) in a region where new data no longer arrives. Note that we only apply exponential decay to the *most recently updated cluster*, so that we do not forget valuable information about other clusters while the data from this class arrives.

4 The SECLEDS Algorithm

SECLEDS has three modules: an initialization module (Init), an assignment module (Assign), and an update module (Update). The task is to assign each item in S to one of the `k`-clusters. SECLEDS maintains and updates a model of the stream seen so far in the form of `k`-clusters, $\mathcal{C} = \{C_1, \ldots C_k\}$. For clarity, we use t to denote the clusters at time t. These superscripts are removed from

Algorithm 1. Each cluster is represented by a set of p-medoids and their votes, *i.e.,* for $1 \leq i \leq k$, $C_i^t = \{(m_{i,1}^t, v_{i,1}^t) \ldots (m_{i,p}^t, v_{i,p}^t)\}$, where for $1 \leq j \leq p$: $m_{i,j}^t \in S$ is the j^{th} medoid of the i^{th} cluster at time t having $v_{i,j}^t \in \mathbb{R}$ votes.

Init. A batch $\mathbb{B}$ from the start of S is used to initialize the clusters. The batch can be small but enough to select $k \cdot p$ medoids. In the experiments, we used a batch size of $(1.5 \cdot k \cdot p)$. We use a non-uniform sampling strategy, similar to the Lloyd's algorithm [16], to select the primary medoid of each cluster: SECLEDS selects the first medoid of the first cluster $(m_{1,1}^1)$ arbitrarily from the batch. Another $k-1$ medoids are sampled with a probability proportional to the squared distance between $m_{1,1}^1$ and other items in $\mathbb{B}$. This initializes the primary medoid of each cluster. The other `p-1` medoids for each cluster C_i^1 are the items in $\mathbb{B}$ that are closest to its primary medoid $m_{i,1}^1$. This way, the medoids maintain cluster separation by reducing the risk of medoids from multiple clusters overlapping each other. All medoids start with 0 votes.

Assign and Update. With the clusters initialized, the stream processing begins. ASSIGN and UPDATE are called for each item in S. ASSIGN has 3 steps: (i) An incoming item s at time t is assigned to the cluster C_{cid}^t for which its previous medoids C_{cid}^{t-1} have the least average distance to s, formally defined in Eq. (1) for any given distance function $d(.,.)$. (ii) The closest medoid to s receives a vote, while exponential decay λ is applied to the other medoids *i.e.,* for all $1 \leq j \leq p$ and $j' \neq j$: $v_{cid,j}^t = (v_{cid,j}^{t-1} + 1)$ if $j = \arg\min_j d(s, m_{cid,j}^{t-1})$, otherwise $v_{cid,j'}^t = v_{cid,j'}^{t-1} \cdot (1 - \lambda)$. This way, the medoids maintain an estimate of their centers of mass without storing any part of the stream. (iii) The votes of all other clusters remain the same, *i.e.,* $v_{i,j}^t = v_{i,j}^{t-1}$ for all $i \neq cid$ and $1 \leq j \leq p$.

$$C_{cid}^t = \underset{1 \leq cid \leq k}{\arg\min} \frac{\sum_{j=1}^{p} d(s, m_{cid,j}^{t-1})}{p} \tag{1}$$

At every time step t, the new data item s is promoted to be a medoid of C_{cid}^t: the medoid having the least votes which is not the newest medoid is replaced by s, *i.e.,* $\{m_{cid,j}^t = s,\ v_{cid,j}^t = 0\}$ where $j = \arg\min_j v_{cid,j}^{t-1}$ and $m_{cid,j}^{t-1} \neq \eta_{cid}^t$, where η_{cid}^t keeps track of the newest medoid of cluster cid at time t. Inspired by Tabu search [11], including η_{cid}^t ensures that the most-recently updated medoid is not selected to be replaced each time. Tabu search is a local search meta-heuristic that selects which values to change except for the last δ ones ($\delta = 1$ in this case).

Time Complexity. Given `k` clusters, `p` medoids, `b` batch size, and `n` items in the stream, SECLEDS has a time complexity of $\mathcal{O}(\mathtt{n})$: SECLEDS selects the first medoid at random from the initial batch, and then performs `b` distance computations to find the other `k-1` primary medoids. The rest of the `k(p-1)` medoids are also selected using the same distance information. In total, this requires $\mathcal{O}(\mathtt{kb})$ distance computations. For every $s \in S$, SECLEDS computes the average distance to each cluster, which requires `kp` distance computations. Over an entire run, this gives `nkp` distance computations. In the UPDATE module, SECLEDS reallocates medoid votes without any distance computations, making

Algorithm 1: SECLEDS for clustering sequences in evolving streams

Input: Data stream, nclusters, nprototypes: $S,\ k,\ p$

function SECLEDS(S, k, p)
  $b \leftarrow 1.5 \cdot k \cdot p$
  $\mathbb{B} \leftarrow$ Collect b items from S
  $\mathcal{C} \leftarrow$ INIT($\mathbb{B}$, k, p) // INIT
  forall s *in* $S[b:]$ **do**
    $cid \leftarrow \arg\min_{1 \leq cid \leq k} \frac{1}{p} \cdot \sum_{j=1}^{p} d(s, m_{cid,j})$ // ASSIGN
    $j \leftarrow \arg\min_j d(s, m_{cid,j})$ for all $1 \leq j \leq p$
    $v_{cid,j} \leftarrow (v_{cid,j} + 1)$, $v_{cid,j'} \leftarrow v_{cid,j'} \cdot (1 - \lambda)$ for $j' \neq j$
    $j \leftarrow \arg\min_j v_{cid,j}$ where $m_{cid,j} \neq \eta_{cid}$ for all $1 \leq j \leq p$ // UPDATE
    $m_{cid,j} \leftarrow s,\ v_{cid,j} \leftarrow 0$
    yield cid

function INIT($\mathbb{B}, k, p$)
  Choose $m_{1,1} \in \mathbb{B}$ arbitrarily. Let $C_1 \leftarrow \{(m_{1,1}, 0)\}$
  for $i \leftarrow 2 \ldots k$ **do**
    Choose $m_{i,1} \in \mathbb{B}$ with probability $d(m_{i,1}, m_{1,1})^2$, $m_{i,1} \neq m_{1,1}$
    Let $C_i \leftarrow \{(m_{i,1}, 0)\}$
  for $i \leftarrow 1 \ldots k$ **do**
    $dist \leftarrow d(b, m_{i,1})$ for all $b \in \mathbb{B}$ and $b \neq m_{i,1}$
    Choose $\{m_{i,2} \ldots m_{i,p}\}$ having smallest values in $dist$
    Update $C_i \leftarrow \{(m_{i,1}, 0) \ldots (m_{i,p}, 0)\}$
  return $\{C_1, \ldots, C_k\}$

the runtime negligible. Since `k` and `p` are small user-supplied parameters, the overall runtime complexity is $\mathcal{O}(\mathtt{n})$.

Space Complexity. After initialization, SECLEDS only stores the `p` medoids and their votes for the `k` clusters. Since these are (small) user-defined parameters, the space complexity of SECLEDS is $\mathcal{O}(1)$.

5 Experimental Setup

Datasets. We use three synthetic and a real dataset containing 2D points and univariate sequences, see Table 1. The data generation process is given in appendix. The synthetic datasets are released in the SECLEDS code repository.

Blobs: The blob dataset was created using `scikit-learn` [22]. The dataset contains $n = 100,000$ two-dimensional points ($d = 2$), equally distributed in $k = 10$ classes, with varying standard deviations.

Table 1. Summary of experimental datasets.

Dataset	Type	Drift	Stream size (n)	Clusters (k)	Dimensions (d,w)
Blobs	Synthetic	No	100,000	10	(2,1)
Sine-curve	Synthetic	No	50,000	4	(1,100)
Sine-curve-drifted	Synthetic	Yes	50,000	4	(1,100)
CTU13-9	Real-world	Yes	213,386	2	(1,100)

Sine-Curve: A sine-curve generator was used to create $k = 4$ synthetic univariate sine curves of length $1,250,000$ each, using varying *frequency*, *phase* and *error* (see appendix). Each curve is partitioned using a non-overlapping window of length $w = 100$ to obtain the experimental dataset. In total, $n = 50,000$ curves are obtained, equally divided across $k = 4$ classes.

Sine-Curve-Drifted: Incremental concept drift is added to the Sine-curve dataset by shifting the phase of each curve by a factor of $(drift \cdot c\_id)$, where c_id is the curve index in the stream, and $drift = 0.05$. Note that adding drift to the frequency of the sine curves produces similar results.

CTU13-9: CTU13 [10] is an open source dataset composed of network traffic (netflows) coming from real botnet-infected hosts and normal hosts. We use scenario-9, containing 10 bot-infected hosts and 6 benign hosts. A total of $2,087,509$ (normal and botnet) netflows were captured over 5 h and 37 min. We obtain $n = 213,386$ univariate sequences of length $w = 100$ using a sliding window model [33] with $step\_size = 1$ (see appendix).

We use Euclidean distance for all datasets. For the Sine-curve and network traffic datasets, we additionally use Dynamic time warping (DTW). Note that Euclidean distance can only be used with fixed-length sequences and often produces less accurate results compared to DTW [12,29].

Stream Configuration. A data stream S of size n is constructed from a chosen experimental dataset. For each experiment, the clustering task is executed $trials$-times, randomly shuffling the stream each time, to make the results data-order-invariant. A clustering task invokes SECLEDS and the baselines such that each algorithm observes the exact same order of data arrival. In this paper, we set $trials = 10$, unless otherwise reported. All experiments are run on Intel Xeon E5620 quad-core processor with 74 GB RAM.

Evaluation. We use two metrics for performance evaluation: i) *runtime* to cluster a stream size of n; ii) *F1 score* computed from the pairwise co-occurrences of items in the stream using Eq. (2), as originally defined in [19].

$$eval(a,b) = \begin{cases} y_a = y_b \wedge C_x = C_y, & \text{true positive} \\ y_a = y_b \wedge C_x \neq C_y, & \text{false negative} \\ y_a \neq y_b \wedge C_x = C_y, & \text{false positive} \\ y_a \neq y_b \wedge C_x \neq C_y, & \text{true negative} \end{cases} \tag{2}$$

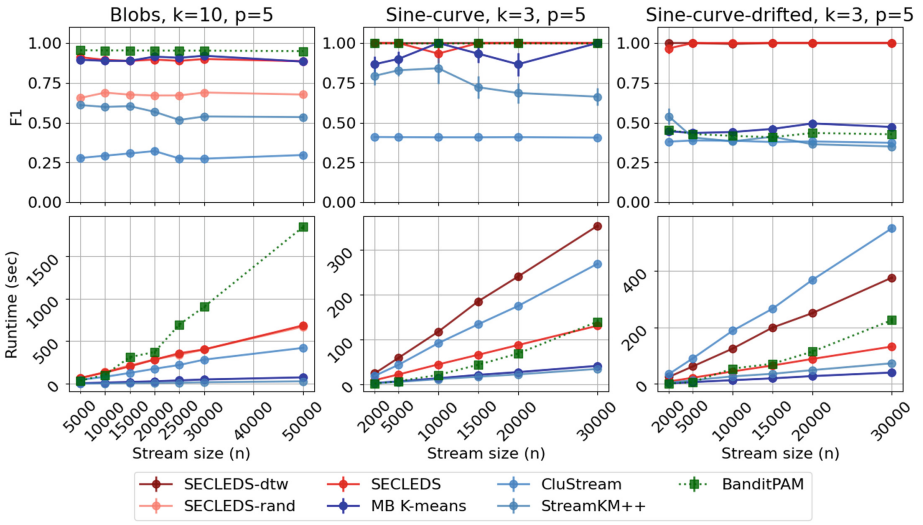


Fig. 3. Clustering Blobs and Sine-curves: SECLEDS's runtime grows approximately linearly with stream size, while maintaining competitive F1 score with the best-performing baselines, *i.e.,* BanditPAM and Minibatch k-means. SECLEDS consistently performs better than all baselines in the presence of concept drift.

where y_a and y_b are labels of items a and b that are placed in clusters C_x and C_y. Since clusters do not have pre-defined labels, data from one class may be assigned to arbitrary clusters in different runs. Thus, instead of looking at the predicted label, we measure F1 using the pairwise co-occurrences of true labels.

Baselines. We compare SECLEDS with state-of-the-art open-source partition-based clustering algorithms with k-fixed clusters: a) Streaming: CluStream, StreamKM++; b) Batching: MiniBatch k-means; c) Offline: BanditPAM. Mini-Batch k-means and StreamKM++ are online versions of k-means, while CluStream is an adaptive, online-offline algorithm. BanditPAM (v1.0.5) is used as a benchmark for the best achievable clustering on a static dataset. We set time_window=1, max_micro_clusters=$k \cdot p$, halflife=0.5 for CluStream; chunk_size=1, halflife=0.5 for StreamKM++; batch_size=1, max_iter=1 for Minibatch k-means.

6 Empirical Results

Key Findings. In this section, we empirically demonstrate the following results:

1. SECLEDS produces high-quality clusters, regardless of concept drift, stream size n, data dimensionality (d, w), and number of clusters k. SECLEDS shows competitive F1 compared to the best performing baseline (BanditPAM), while reducing the number of required distance computations by 83.7%.

2. SECLEDS outperforms *all baselines* by 138.7% when the stream contains concept drift. SECLEDS outperforms the *best-performing streaming baseline* by 58.2% on Blobs, 33.3% on Sine-curve, and 143.7% on Sine-curve-drifted.
3. SECLEDS-dtw clusters ∼5.5 h of network traffic in just 8% of the time. Thus, it can handle networks with bandwidths of up to 1.08 Gbps, which is significantly higher than the requirements of a typical enterprise network.

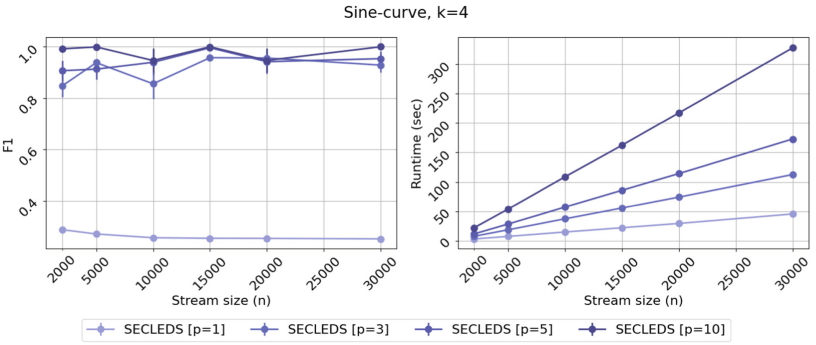
Point vs. Sequence Clustering. We use Blobs with $k = 10$ on stream sizes $n = (5000, \ldots 50,000)$, and Sine-curve with $k = 3$ on stream sizes $n = (2000, \ldots 30,000)$. For both, we set $p = 5$, $\lambda = 0.1$, $trials = 10$. The mean and standard deviation of the F1 scores and runtimes are given in Fig. 3. The benchmark (BanditPAM) achieves a mean F1 of 0.95 and 1.0 for Blobs and Sine-curve, respectively.

SECLEDS outperforms both CluStream and StreamKM++ on the Blobs dataset, and additionally outperforms Minibatch k-means on the Sine-curve dataset. Minibatch k-means performs exceptionally well on point clustering, but loses its edge on sequence clustering. This is because the centroids are computed by collapsing temporally-linked dimensions into single values that do not adequately represent the sequences. An improvement in F1 score is observed for CluStream and StreamKM++ on the higher dimensional Sine-curve dataset because of fewer clusters ($k = 10$ vs. $k = 3$).

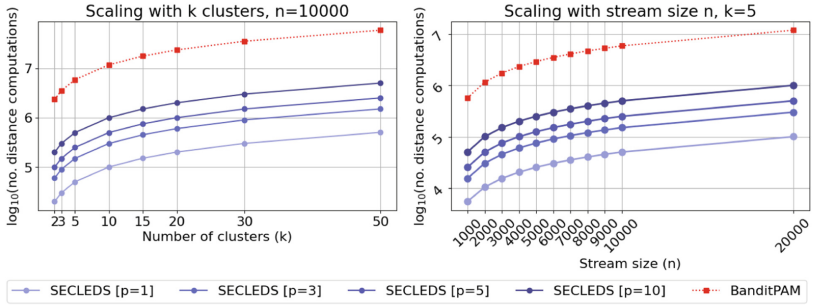
We also compare the effect of euclidean and dynamic time warping distance on the Sine-curve dataset. Although, they both produce equivalent results, it must be noted that euclidean distance only works with fixed-length sequences. An example of SECLEDS-dtw on clustering bivariate sequences $d = 2, w =$(min:15, max:121) from *UJI Pen Characters* [8] is given in the appendix.

Initialization Quality. Stream clustering algorithms are greatly impacted by the quality of cluster initialization. To test this, we compare SECLEDS against SECLEDS-rand (initialized with randomly selected medoids from the initial batch $\mathbb{B}$). Evidently, the clusters take a long time to converge, regardless of the stream size. The cumulative F1 score over time for these configurations is given in the appendix, showing that although the impact of poor initialization is reduced over time, SECLEDS-rand does not completely recover from it. Thus, the distance-based non-uniform sampling strategy proves to be extremely helpful in initializing good clusters.

Clustering with Concept Drift. We use Sine-curve-drifted with $k = 3$, $p = 5$, $\lambda = 0.1$, $trials = 10$ on stream sizes $n = (2000, \ldots 30,000)$. SECLEDS outperforms *all baselines* by 138.7%, and outperforms the *best-performing streaming baseline* by 143.7%, on average. BanditPAM no longer serves as a benchmark because it only has a static view of the data, *i.e.,* it does not distinguish between class distributions at $T = t_x$ and $T = t_{x+y}$. Both SECLEDS and CluStream maintain their F1 scores with concept drift, but SECLEDS is 161.8% better than CluStream. StreamKM++ and Minibatch k-means observe a significant reduction in their performance. We hypothesize that it might be due to the lack



(a) $p = \{3, 5\}$ provide the best trade-off between runtime and F1 for Sine-curve.



(b) SECLEDS requires 83.7% fewer distance computations compared to BanditPAM.

Fig. 4. Scaling with n, k, and p: (a) Empirical results; (b) Theoretical estimate.

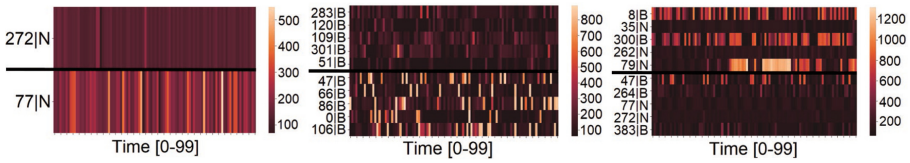
of exponential decay in k-means, which limits the movement of the centroids towards newer data. This experiment provides strong evidence for SECLEDS' ability to handle concept drift with only k-fixed clusters.

Runtime Analysis. StreamKM++ and Minibatch k-means are among the fastest clustering algorithms on all datasets, which is expected since they are based on k-means. CluStream does not scale well for high-dimensional datasets, and is much slower than SECLEDS on sequence clustering. As the stream size n grows, SECLEDS also becomes faster than the high-performance implementation of BanditPAM on both point and sequence clustering. Interestingly, the runtimes of BanditPAM, CluStream and StreamKM++ seem to be affected by concept drift: given the same dataset and constant parameters, their runtimes increase approximately twofold when there is drift in the data. We hypothesize that this is a side effect of the sampling strategy used to speed up these algorithms.

Scaling with n***,*** k***, and*** p***.*** We use Sine-curve with $k = 4$, $p = \{1, 3, 5, 10\}$, $\lambda = 0.1$, $trials = 10$ on stream sizes $n = (2,000, \ldots 30,000)$. The mean and standard deviation of the F1 and runtime of SECLEDS is reported in Fig. 4a.

Table 2. Clustering real network traffic: Compared to BanditPAM, SECLEDS requires fewer distance computations, is faster, and has a better cluster quality. SECLEDS-dtw is slower but produces better clusters than SECLEDS. Overall, ∼5.5 h of network traffic is clustered in under 27 min (Bold = best scores).

	Stream config.	# Distances ($k = 2$)	Run time ($k = 2$)	**F1** ($k = 2$)	F1 ($k = 5$)
BanditPAM	Time-ordered	10.3×10^6	978.03 s	0.64	0.38
	Cross-validated		984.8 s	0.64	0.38
SECLEDS	Time-ordered	$\mathbf{2.1 \times 10^6}$	**629.39 s**	**0.85**	0.82
	Cross-validated		**631.84 s**	0.79	0.76
SECLEDS-dtw	Time-ordered	$\mathbf{2.1 \times 10^6}$	1623.05 s	**0.85**	**0.88**
	Cross-validated		1626.89 s	**0.81**	**0.80**



(a) BanditPAM (b) SECLEDS-dtw (c) SECLEDS

Fig. 5. Visualizing the medoids of BanditPAM, SECLEDS & SECLEDS-dtw on $k = 2$, $p = 5$. Each row is a medoid. The label denotes curve identifier and y_i.

A single medoid per cluster, which is standard for PAM-based algorithms, does poorly in a streaming setting. Intuitively, more medoids help to improve the stability of the clusters, but the relationship is not linear. If `p` is set too low, the medoids keep jumping to various regions in the dataset, and if it is set too high, the medoids slow down the evolution of the clusters, having an equally detrimental effect on the performance. The optimal value of `p` with respect to performance and runtime is dataset-dependent. For Sine-curve, $p = \{3, 5\}$ are good alternatives. Additionally, although SECLEDS has multiple medoids per cluster, it performs significantly fewer distance computations compared to the (almost linear) BanditPAM. Figure 4b shows this for increasing stream size n, number of clusters k, and number of medoids p, with BanditPAM as reference.

6.1 Use Case: Intelligent Network Traffic Sampling via SECLEDS

A typical enterprise network has a bandwidth of 25 Mbps[2], which produces about 17,000 packets *per second*, consuming 2 terabytes of storage space *each day*! To conserve space, the packets are aggregated into network flows (netflows) at the router level, and only a fraction of them are stored for analysis *i.e.,* 1

[2] https://mosaicnetworx.com/it-challenges/bits-bytes-understanding-enterprise-network-speeds/.

in N netflows are stored. Naturally, randomly sampled network traffic does not preserve the temporal patterns of the data, thus limiting the efficacy of traffic profiling and behavior analytics.

We propose to cluster sequences of netflows using SECLEDS, and to periodically store a medoid snapshot of each cluster, since they are representative of the network traffic seen so far. This way, each snapshot stores an overview of temporally-linked netflows. The number of clusters k can be chosen depending on the required granularity of behaviors captured by the clusters. It can also be approximated from an initial batch using, *e.g.,* [3]. The number of medoids p can be configured according to available storage space, network bandwidth, and the intervals at which to store the medoids.

We demonstrate this use case by generating a stream from the CTU13-9 netflows. The construction and feature engineering processes are given in the appendix. In short, the ground truth provides two classes, *i.e.,* $y_i \in \{botnet, normal\}$. Univariate sequences of average bytes per netflow are used to separate the two classes. We use two configurations for the stream: i) *Time-ordered:* the sequences arrive in order of their timestamps; ii) *Cross-validated:* we shuffle the stream. We run SECLEDS and SECLEDS-dtw with $k = 2$, $p = 5$, $trials = 5$, and compare the performance against BanditPAM. The results are given in Table 2.

SECLEDS is faster and produces better medoids compared to BanditPAM. Figure 5 visualizes the final medoids produced by all three algorithms in the form of temporal heatmaps. Temporal heatmaps have previously been used to visualize temporal similarities in [20]. Each row shows a sequence (medoid), and the colors indicate the magnitude of the curve at each time step. Both medoids of BanditPAM are from the normal class. Although the medoids of SECLEDS-dtw are all from the botnet class, it is evident that they capture distinct behaviors of the malicious hosts. SECLEDS finds medoids from both classes, but the clusters are impure, *i.e.,* more medoids on average are from different classes. As such, the cluster quality of SECLEDS-dtw is significantly better than that of SECLEDS.

The results indicate that there are many smaller classes in the network stream, reflecting the various behaviors of benign and infected hosts. When k is set to a larger number, the clustering algorithms find smaller, purer data regions, *e.g.,* for $k = 5$, SECLEDS-dtw produces 4 pure clusters (2 normal and 2 botnet), while SECLEDS only produces 1 pure (normal) cluster, see appendix for their temporal heatmaps. Table 2 shows the F1 scores for $k = 5$. Note that although the clustering results for $k = 5$ are better than $k = 2$, the former obtains a lower F1 score as a side-effect of the metric: it penalizes higher number of clusters when less class labels are available by lowering the recall. As such, we recommend to over-estimate k in order to sample many regions from the network traffic.

Finally, SECLEDS is faster than SECLEDS-dtw, as expected. SECLEDS clusters the entire stream in 3.1% of the traffic collection time, and SECLEDS-dtw in 8% of the collection time. This experiment provides evidence that SECLEDS can handle much larger network bandwidths.

Network Bandwidth Support. SECLEDS-dtw can handle networks with a bandwidth of up to 1.08 Gbps, which is more than sufficient for small to medium enterprises. The experiments in Table 2 show that SECLEDS spends $\frac{1626.89}{213386} = 0.0076$ seconds on average to cluster a single sequence of length $w = 100$. Thus, SECLEDS can cluster 131.58 sequences per second. Assuming that the sequence windows w are non-overlapping over the traffic stream, SECLEDS can process 13,158 individual netflows per second. The CTU13-9 dataset is composed of 115,415,321 packets aggregated into 2,087,509 netflows. Assuming uniform distribution, each netflow contains about 55.2 packets. SECLEDS can, thus, process 726,315.79 packets per second. Given that each network packet is about 1500 bytes, this makes a total of 1.089 Gigabytes per second. Similarly, SECLEDS can handle network bandwidths of up to 2.79 Gbps.

7 Conclusions

We propose SECLEDS, a streaming version of k-medoids with constant memory footprint. SECLEDS uses a combination of multiple medoids per cluster and a medoid voting scheme to create k-clusters that evolve with evolving data streams. Testing on several real and synthetic datasets and comparing against state-of-the-art baselines, we demonstrate that i) SECLEDS achieves competitive F1 score compared to the benchmark (BanditPAM) on streams without concept drift; ii) SECLEDS outperforms all baselines by 138.7% on streams with concept drift; iii) SECLEDS reduces the number of required distance computations by 83.7% compared to the benchmark, making it faster than BanditPAM and CluStream for several clustering tasks, iv) SECLEDS can support high-bandwidth network streams of up to 1.08 Gbps using the expensive dynamic time warping distance. These results reinforce the importance of designing lightweight medoid-based stream clustering algorithms.

Acknowledgements. We thank Ruben te Wierik, Silviu Fucarev, and Rami Al-Obaidi for their contributions to the SECLEDS algorithm.

References

1. Ackermann, M.R., Märtens, M., Raupach, C., Swierkot, K., Lammersen, C., Sohler, C.: Streamkm++ a clustering algorithm for data streams. JEA **17**, 2–1 (2012)
2. Aggarwal, C.C., Philip, S.Y., Han, J., Wang, J.: A framework for clustering evolving data streams. In: VLDB, pp. 81–92. Elsevier (2003)
3. de Andrade Silva, J., Hruschka, E.R.: Extending k-means-based algorithms for evolving data streams with variable number of clusters. In: ICMLA, vol. 2, pp. 14–19. IEEE (2011)
4. Barros, R.S.M., Santos, S.G.T.C.: A large-scale comparison of concept drift detectors. Inf. Sci. **451**, 348–370 (2018)
5. Boeva, V., Nordahl, C.: Modeling evolving user behavior via sequential clustering. In: Cellier, P., Driessens, K. (eds.) ECML PKDD 2019. CCIS, vol. 1168, pp. 12–20. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-43887-6_2

6. Cao, F., Estert, M., Qian, W., Zhou, A.: Density-based clustering over an evolving data stream with noise. In: SDM, pp. 328–339. SIAM (2006)
7. Cook, D.J., Krishnan, N.C., Rashidi, P.: Activity discovery and activity recognition: a new partnership. IEEE Trans. Cybern. **43**(3), 820–828 (2013)
8. Dua, D., Graff, C.: UCI machine learning repository (2017)
9. Fahy, C., Yang, S.: Finding and tracking multi-density clusters in online dynamic data streams. IEEE Trans. Big Data **8**, 178–192 (2019)
10. Garcia, S., Grill, M., Stiborek, J., Zunino, A.: An empirical comparison of botnet detection methods. Comput. Secur. **45**, 100–123 (2014)
11. Glover, F.: Future paths for integer programming and links to artificial intelligence. Comput. Oper. Res. **13**(5), 533–549 (1986)
12. Guijo-Rubio, D., Durán-Rosal, A.M., Gutiérrez, P.A., Troncoso, A., Hervás-Martínez, C.: Time-series clustering based on the characterization of segment typologies. IEEE Trans. Cybern. **51**(11), 5409–5422 (2020)
13. Guo, J., Liu, G., Zuo, Y., Wu, J.: Learning sequential behavior representations for fraud detection. In: ICDM, pp. 127–136. IEEE (2018)
14. Hyde, R., Angelov, P., MacKenzie, A.R.: Fully online clustering of evolving data streams into arbitrarily shaped clusters. Inf. Sci. **382**, 96–114 (2017)
15. Islam, M.K., Ahmed, M.M., Zamli, K.Z.: A buffer-based online clustering for evolving data stream. Inf. Sci. **489**, 113–135 (2019)
16. Lloyd, S.: Least squares quantization in PCM. IEEE Trans. Inf. Theory **28**(2), 129–137 (1982)
17. Lu, J., Liu, A., Dong, F., Gu, F., Gama, J., Zhang, G.: Learning under concept drift: a review. TKDE **31**(12), 2346–2363 (2018)
18. Lu, N., Zhang, G., Lu, J.: Concept drift detection via competence models. Artif. Intell. **209**, 11–28 (2014)
19. Manning, C., Raghavan, P., Schütze, H.: Introduction to information retrieval. Nat. Lang. Eng. **16**(1), 100–103 (2010)
20. Nadeem, A., Hammerschmidt, C., Gañán, C.H., Verwer, S.: Beyond labeling: using clustering to build network behavioral profiles of malware families. In: Stamp, M., Alazab, M., Shalaginov, A. (eds.) Malware Analysis Using Artificial Intelligence and Deep Learning, pp. 381–409. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-62582-5_15
21. Nadeem, A., Verwer, S., Moskal, S., Yang, S.J.: Alert-driven attack graph generation using s-PDFA. IEEE Trans. Dependable Sec. Comput. **19**(2), 731–746 (2021)
22. Pedregosa, F., et al.: Scikit-learn: Machine learning in python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
23. Schubert, E., Rousseeuw, P.J.: Faster k-medoids clustering: improving the PAM, CLARA, and CLARANS algorithms. In: Amato, G., Gennaro, C., Oria, V., Radovanović, M. (eds.) SISAP 2019. LNCS, vol. 11807, pp. 171–187. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-32047-8_16
24. Sculley, D.: Web-scale k-means clustering. In: WWW, pp. 1177–1178 (2010)
25. Silva, J.A., Faria, E.R., Barros, R.C., Hruschka, E.R., de Carvalho, A.C., Gama, J.: Data stream clustering: a survey. CSUR **46**(1), 1–31 (2013)
26. Tiwari, M., Zhang, M.J., Mayclin, J., Thrun, S., Piech, C., Shomorony, I.: Banditpam: almost linear time k-medoids clustering via multi-armed bandits. NeurIPS **33**, 10211–10222 (2020)
27. Ushakov, A.V., Vasilyev, I.: Near-optimal large-scale k-medoids clustering. Inf. Sci. **545**, 344–362 (2021)

28. Wang, T., Li, Q., Bucci, D.J., Liang, Y., Chen, B., Varshney, P.K.: K-medoids clustering of data sequences with composite distributions. IEEE Trans. Signal Process. **67**(8), 2093–2106 (2019)
29. Wang, X., Mueen, A., Ding, H., Trajcevski, G., Scheuermann, P., Keogh, E.: Experimental comparison of representation methods and distance measures for time series data. Data Min. Knowl. Disc. **26**(2), 275–309 (2013)
30. Wang, Y., Chen, L., Mei, J.P.: Incremental fuzzy clustering with multiple medoids for large data. IEEE Trans. Fuzzy Syst. **22**(6), 1557–1568 (2014)
31. Zhang, T., Ramakrishnan, R., Livny, M.: Birch: a new data clustering algorithm and its applications. Data Min. Knowl. Disc. **1**(2), 141–182 (1997)
32. Žliobaitė, I., Pechenizkiy, M., Gama, J.: An overview of concept drift applications. In: Japkowicz, N., Stefanowski, J. (eds.) Big Data Analysis: New Algorithms for a New Society. SBD, vol. 16, pp. 91–114. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-26989-4_4
33. Zubaroğlu, A., Atalay, V.: Data stream clustering: a review. Artif. Intell. Rev. **54**(2), 1201–1236 (2021)