



Detecting Arbitrarily Oriented Subspace Clusters in Data Streams Using Hough Transform

Felix Borutta^(✉), Daniyal Kazempour, Felix Mathy,
Peer Kröger, and Thomas Seidl

Ludwig-Maximilians-Universität München, Munich, Germany
{borutta,kazempour,kroeger,seidl}@dbs.ifi.lmu.de

Abstract. When facing high-dimensional data streams, clustering algorithms quickly reach the boundaries of their usefulness as most of these methods are not designed to deal with the curse of dimensionality. Due to inherent sparsity in high-dimensional data, distances between objects tend to become meaningless since the distances between any two objects measured in the full dimensional space tend to become the same for all pairs of objects. In this work, we present a novel oriented subspace clustering algorithm that is able to deal with such issues and detects arbitrarily oriented subspace clusters in high-dimensional data streams. Data streams generally implicate the challenge that the data cannot be stored entirely and hence there is a general demand for suitable data handling strategies for clustering algorithms such that the data can be processed within a single scan. We therefore propose the CASHSTREAM algorithm that unites state-of-the-art stream processing techniques and additionally relies on the Hough transform to detect arbitrarily oriented subspace clusters. Our experiments compare CASHSTREAM to its static counterpart and show that the amount of consumed memory is significantly decreased while there is no loss in terms of runtime.

Keywords: Oriented subspace clustering · Stream clustering

1 Introduction

Data clustering, i.e., finding groups of similar objects, is an established and widely used technique for unsupervised problems and/or for explorative data analysis. However, when facing high-dimensional data, particularly clustering algorithms quickly reach the boundaries of their usefulness as most of them are not designed to deal with the problems known by the “curse of dimensionality”. Due to inherent sparsity in high-dimensional data, distances between any two objects measured in the full dimensional space tend to become the same for all pairs of objects and, thus, can no longer be used to distinguish similar from dissimilar objects. Furthermore, clusters often appear within different

Electronic supplementary material The online version of this chapter (https://doi.org/10.1007/978-3-030-47426-3_28) contains supplementary material, which is available to authorized users.

lower dimensional subspaces. Therefore, it is usually not useful to search for clusters in the full dimensional data space or apply dimensionality reduction which would only result in one subspace rather than several different ones. To overcome those issues, several subspace clustering algorithms have been developed in the past that simultaneously search for meaningful subspaces and for clusters (within these subspaces). Some of these algorithms, e.g. [4, 15, 16], assume attribute independence and restrict themselves to the detection of axis-parallel subspace clusters for performance reason. More general, so-called correlation clustering algorithms, e.g. [1, 2, 7, 8], allow arbitrarily oriented subspaces that represent a (usually linear) combination of features, i.e., explicitly allow correlation among features.

Another, yet less considered challenge is subspace clustering in data streams. Nowadays, as data is produced with high velocity, streaming algorithms become more and more important. This also holds for areas where high-dimensional data is produced rapidly, e.g., in industry where large numbers of machine sensors record huge amounts of data within short time periods. In these scenarios, the data can usually no longer be stored entirely and hence there is a general need for suitable data handling strategies for clustering algorithms such that the data can be processed within a single scan. In this work, we tackle this problem and present a novel oriented subspace clustering algorithm that is able to detect arbitrarily oriented subspace clusters in data streams. This method not only reduces the amount of required memory for processing the data significantly, but also compresses entire groups of data that are similar wrt to various combinations of features. The key idea of the proposed method is to load chunks of data into memory, deriving so-called *Concepts* as summary structures and applying a decay mechanism to downgrade the relevance of stale data. Our experimental evaluation demonstrates the usefulness of the presented method and shows that the used heap space is drastically reduced without losses in terms of runtime and accuracy.

2 Related Work

Correlation Clustering. Static algorithms for oriented subspace clustering can be categorized into PCA-based and Hough-based approaches. The PCA-based approaches [2, 5, 7] rely on decomposing neighborhood sets into Eigensystems that are used to define the corresponding subspaces. The usage of neighborhood sets makes them prone to outliers and noise. In contrast, approaches based on Hough transformations [1, 14] rely on parameter space transformations, making them generally more robust. All these methods have been designed for static data and are not applicable in streaming environments.

Stream Clustering. Previously published work on stream clustering can generally be distinguished by the way the algorithms process the incoming data. A large group of algorithms rely on (*clustering*) *feature vector* (CF) data structures that have originally been proposed for the *BIRCH* algorithm [22]. The idea is to represent a set of data objects by only a few key statistics that sufficiently

describe the aggregated data. This approach has been adapted for many other stream clustering approaches, e.g., [3,9,10]. Another compression technique that is widely employed for stream clustering is to only keep track of the cluster representatives. The basic idea is to represent entire chunks of data solely in form of cluster representatives, e.g., cluster centroids, [12,17,23]. Further, but less related, techniques to deal with the challenge of summarizing data streams can be found in [21].

Subspace Clustering in Data Streams. The first method able to cluster high-dimensional data streams properly was *HPStream* [4], a k -means based axis-parallel subspace clustering method that uses an adopted form of CF vectors to represent relevant cluster statistics. *IncPreDeCon* [15] is an incremental, axis-parallel subspace clustering algorithm based on a density-based clustering model that supports incremental updates but lacks supporting any form of aging and hence cannot deal with streaming data directly. *PreDeConStream* [13] and *HDDStream* [16] present density-based (axis-parallel) subspace clustering algorithms that both aggregate incoming data objects within different microcluster structures and retrieve the final clustering by following (slightly different) variants of the density-based clustering scheme proposed in [6]. The *SiblingTree* method [18] is a grid-based axis-parallel subspace clustering approach aiming at detecting all low-dimensional clusters in all subspaces. All these previously mentioned methods are limited to find axis-parallel subspaces. The recently presented *CorrStream* algorithm [8] is a PCA-based approach for arbitrarily-oriented subspace clustering on data streams. As a PCA-based method, it determines subspace clusters derived from neighborhood sets, and hence is prone to outliers. In contrast, our method relies on Hough transformation and hence is able to filter outlier.

3 Correlation Clustering Using Hough Transform

The Hough transformation originally has been introduced for detecting linear segments in image data [19]. The basic idea is to map every object in data space to its corresponding object function in Hough space, and subsequently identify intersections of a specific amount of object functions. If such an intersection exists, the corresponding data objects are located on a line segment in data space. This duality of the Hough transform is shown in Fig. 1. The *CASH* algorithm [1] borrows this idea of parameter space transformation for the sake of oriented subspace clustering. Precisely, they transform objects from data space to Hough space and scan the Hough space for dense areas, i.e., areas where many functions intersect, by subdividing the space into grid cells in a top-down fashion. For a given cell c , if the

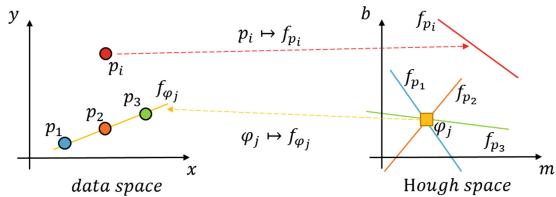


Fig. 1. Left: data space, right: Hough space

The *CASH* algorithm [1] borrows this idea of parameter space transformation for the sake of oriented subspace clustering. Precisely, they transform objects from data space to Hough space and scan the Hough space for dense areas, i.e., areas where many functions intersect, by subdividing the space into grid cells in a top-down fashion. For a given cell c , if the

Input: Data Stream \mathcal{S} , Batch size b
Output: Clustering

```

1:  $Clustering \leftarrow \emptyset$ 
2:  $batch \leftarrow$  empty collection of size  $b$ 
3: for incoming data object  $o$  from  $\mathcal{S}$  do
4:   if  $batch$  is not full then
5:     add  $o$  to  $batch$ 
6:   end if
7:   if  $batch$  is full then
8:      $currentConcepts = CASH(batch)$ 
9:      $Clustering.add(currentConcepts)$ 
10:     $unifyConcepts(Clustering, ...)$ 
11:    // see Section 4.4
12:     $batch \leftarrow$  empty collection of size  $b$ 
13:   end if
14: end for
    
```

Algorithm 1. CashStream

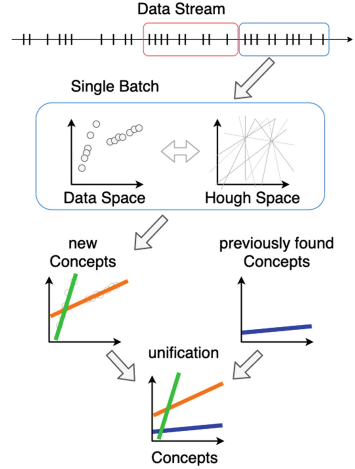


Fig. 2. Workflow

number of object functions intersecting c is greater or equal than a pre-defined $minPts$ parameter, c is split into halves according to a predefined order on the axes. The division terminates if a resulting cell is either considered sparse, i.e., the number of object functions intersecting this cell is less than $minPts$, or a maximum number of splits $maxSplit$ is reached. A grid cell c that is dense after $maxSplit$ divisions represents a cluster: the points corresponding to the functions intersecting c form a cluster within a arbitrarily oriented $(d-1)$ -dimensional subspace. However, the cluster (or some of the contained objects) might form an even lower dimensional cluster. Therefore, the object functions that form the $(d-1)$ -dimensional cluster are transformed back into the data space and projected onto the orthonormal basis that can be derived from c . To detect subspace clusters of lower dimensions, the $CASH$ algorithm is performed on the resulting $(d-1)$ -dimensional dataset recursively until no more cluster can be found.

4 CashStream

4.1 Data Processing: Batch Processing

Regarding the facts that data cannot be kept in memory entirely and stale data shall be downgraded within stream applications, the $CASH$ algorithm cannot be adjusted straightforwardly. To tackle these challenges, we propose to process incoming data in batches, similar to [12], i.e., loading chunks of data into memory and eventually computing cluster representatives which are kept in memory while the actual data objects are discarded. This data processing scheme has several advantages as it (1) enables the adaptation to concept shifts since processing data batch-wise allows to identify dense grid cells¹, potentially with novel subspaces, during the division steps, (2) caps the amount of consumed memory and (3) even

¹ Note that this is not possible with real-time stream processing.

allows the flexibility to adjust to changing data dimensionality as there is no need for defining a static grid. Precisely, our algorithm basically performs an adapted variant of *CASH* on single data chunks and keeps cluster representatives, that we will refer to as *Concepts*, in memory. Since the *Concepts* must be maintained efficiently, they are designed to be additive, such that two similar *Concepts* can conveniently be unified into a single *Concept* (see Fig. 2). Algorithm 1 outlines the main procedure of *CASHSTREAM*. After defining the *Concept* data structure, we define the similarity between *Concepts* and describe the unification step as well as the aging procedure in the following.

4.2 Cluster Representatives: Concepts

As a suitable summary structure for data objects that are assigned to a cluster, we define a *Concept* as follows.

Definition 1. *A Concept is a data structure used as abstraction of a cluster resulting from CASH. In a data space $\mathcal{D} \in \mathbb{R}^d$, a Concept of dimensionality $l < d$ captures an l -dimensional hyperplane in parameter space \mathcal{P} with aggregated information of the data objects it contained as a result of CASH. A Concept consists of the following attributes:*

- a set E containing $d - l$ equations in Hessian normal form,
- mean μ of all data objects that are assigned to the cluster,
- number of data objects N that are assigned to the cluster,
- the timestamp t of the last update, and
- reference P to parent *Concept* of dimensionality $l + 1$, if $l < d - 1$.

The $d - l$ equations in Hessian normal form are the hyperplane equations that define the l -dimensional subspace. These are obviously an essential part of the *Concept* as they describe the subspace, are used for the unification with other *Concepts*, and also are part of the final result of *CASHSTREAM*. The mean μ is the centroid of the data objects that are assigned to the corresponding cluster and is used for checking whether the *Concept* can be merged with another one. N denotes the number of data objects that are assigned to the cluster. This value and the timestamp t of the last update of this *Concept* are used to calculate an importance score for the *Concept*. The importance scores are used to weight the *Concepts* for the unification of two similar *Concepts*, since a recent *Concept* that represents a large number of data objects should contribute more than a stale *Concept* that does not represent as many objects. Finally, a *Concept* also includes a reference to a parent *Concept*, i.e., a *Concept* representing a higher-dimensional subspace in which the child *Concept* is embedded. This enables *CASHSTREAM* to retrieve a cluster hierarchy.

On Representing Subspaces in Hessian Normal Form. The Hessian normal form (HNF) [20] has proven to be a well-suited representation for linear correlation cluster models as it contains a normal vector which describes the orientation of the corresponding hyperplane, respectively subspace. This is essential

for the unification step as we use the orientations of two subspaces to determine their similarity. By using the HNF, we can formally describe a $(d-1)$ -dimensional hyperplane \mathcal{H} as

$$\mathbf{x} \cdot \mathbf{n} + b \leq \epsilon,$$

with \cdot indicating the scalar product, $\mathbf{x} \in \mathbb{R}^d$ denoting a data point lying on the hyperplane, $\mathbf{n} \in \mathbb{R}^d$ denoting the unit normal vector and b being the minimum distance between the hyperplane and the origin. Since subspace clusters typically are not perfectly correlated, we allow a certain amount of deviation ϵ and consider every data point \mathbf{x} that solves this equation to lie on \mathcal{H} . Note that the ϵ parameter is implicitly defined by setting the *maxSplit* parameter, i.e., the parameter that basically defines the size of a grid cell on the lowest split level.

A *Concept* contains $d-l$ of such hyperplane equations as it requires $d-l$ HNF equations for describing a l -dimensional subspace. Intuitively, this can be understood as follows: if $d-l$ $(d-1)$ -dimensional hyperplanes intersect in a d -dimensional space (with $l < d$), the intersection is a l -dimensional hyperplane. Mathematically, this can be seen as solving a simple linear system

$$Ax = b,$$

with A denoting an $m \times d$ matrix, where m is the number of normal vectors. If $d > m$, the linear system is under determined and hence the solution set describes a $(d-m)$ -dimensional subspace.

As described in Sect. 3, CASHSTREAM likewise projects the data objects of an i -dimensional cluster onto the corresponding $(i-1)$ -dimensional subspace to find even lower dimensional clusters. In particular, it also produces an i -dimensional normal vector \mathbf{n}_i to define an i -dimensional basis B_i from which the $(i-1)$ -dimensional subspace is derived as $B_i \setminus \mathbf{n}_i \in \mathbb{R}^{i-1}$ in this step. By doing this iteratively until no lower dimensional subspace can be found, the *CASH* procedure retrieves an ordered set of $d-l$ HNF equations for an l -dimensional subspace, i.e.,

$$\begin{aligned} \mathbf{n}_d \cdot x + r_0 &= 0 \\ \mathbf{n}_{d-1} \cdot (B_d \setminus \mathbf{n}_d \cdot x) + r_1 &= 0 \\ \mathbf{n}_{d-2} \cdot (B_{d-1} \setminus \mathbf{n}_{d-1} \cdot (B_d \setminus \mathbf{n}_d \cdot x)) + r_2 &= 0 \\ &\dots \end{aligned}$$

with $\mathbf{n}_{d-i} \in \mathbb{R}^{d-i}$, $0 \leq i < l$, denoting the $(d-i)$ -dimensional normal vector that defines the $(d-i)$ -dimensional basis B_{d-i} , x being a data point associated with the i -dimensional subspace cluster and r_i being the distances between the subspace hyperplane and the origin. $B_{d-i} \setminus \mathbf{n}_{d-i}$ is a $(d-i-1) \times (d-i)$ projection matrix that is used to project $(d-i)$ -dimensional data objects onto the $(d-i-1)$ -dimensional subspace. However, for measuring the similarity between two *Concepts* (cf. Sect. 4.3), the normal vectors have to be d -dimensional. We therefore reconstruct d -dimensional normal vectors from lower-dimensional normal vectors as follows. Let $\mathbf{n}_{d-i} \in \mathbb{R}^{d-i}$, with $0 < i < l$, be the $(d-i)$ -dimensional normal

vector defining the subspace whose basis is denoted as $B_{d-i-1} = B_{d-i} \setminus \mathbf{n}_{d-i}$, then the reconstructed d -dimensional normal vector $\mathbf{n}'_d \in \mathbb{R}^d$ is

$$\mathbf{n}'_d = (((\mathbf{n}_{d-i} \cdot B_{d-i+1} \setminus \mathbf{n}_{d-i+1}) \cdot B_{d-i+2} \setminus \mathbf{n}_{d-i+2}) \cdot \dots) \cdot B_d \setminus \mathbf{n}_d.$$

Employing this reconstruction strategy to all $(d-i)$ -dimensional normal vectors with $0 < i < l$ in addition with the d -dimensional normal vector \mathbf{n}_d finally results in the desired set of $d-l$ non-parallel, and hence linearly independent [11], d -dimensional normal vectors that define the $d-l$ hyperplane equations.

4.3 Similarity Between Concepts

Theoretically, there is an infinite number of equation sets describing a single subspace cluster, e.g., a 1-dimensional subspace cluster can be modeled by the intersection of two 2D hyperplanes, the orientation of which is not necessarily important. In terms of *Concept* similarity, this means that two *Concepts* shall be considered similar as long as the intersections of their subspace equations describe approximately the same subspace, regardless the orientations of their subspace equations when considering them individually. Given this observation and the fact that each subspace hyperplane is defined by its normal vectors, we formalize the distance measure based on the following idea: Understanding an intersecting set of hyperplanes as the set of their respective normal vectors, every other normal vector contained in a second set of equations representing the same linear subspace is linearly dependent to the first set. However, since we aim at quantifying the linear dependence of these vectors rather than just determining whether they are linearly dependent or not, we propose the following similarity measure. Given a set of linearly independent normal vectors $V = \{\mathbf{n}_1, \dots, \mathbf{n}_k\}$, we quantify the linear dependence of another vector \mathbf{m} wrt V by calculating the singular values $SV(A)$ of matrix $A = (\mathbf{n}_1, \dots, \mathbf{n}_k, \mathbf{m})$ and dividing the smallest value by the largest one. The closer the resulting value

$$L_{dep} \left(\underbrace{\mathbf{v}_1, \dots, \mathbf{v}_k}_A, \mathbf{m} \right) = \frac{\min(SV(A))}{\max(SV(A))}$$

is to zero, the closer the vectors of the matrix are to being linearly dependent due to adding \mathbf{m} . Given two *Concepts* C_1 and C_2 with their sets of normal vectors N_1 and N_2 being of cardinality k , and each normal vector representing a $(d-k)$ -dimensional subspace, we define the *Singular Value Distance* as follows:

$$SV_{dist}(C_1, C_2) = \max_{\mathbf{n} \in N_2} (L_{dep}(N_1, \mathbf{n})).$$

Note that this distance measure only accounts for the orientation of the correlation clusters described by the *Concepts*. However, two *Concepts* that describe different, parallel subspaces would have a singular value distance equal to zero. To avoid an unification of such *Concepts* we introduce the following secondary measure accounting for the actual distance in an Euclidean sense, i.e.,

$$d_{perp}(p, E) = |n_1 p_1 + \dots + n_d p_d - r|,$$

with p denoting any data point of a *Concept* C_1 , E denoting the HNF equation of a *Concept* C_2 and n being the corresponding normal vector. As the actual data points that defined the subspace are not available due to aggregating the necessary information, we use the centroid of the *Concept* as representative. Thus, we compute the *Equation Shift Distance* between two *Concepts* C_1 and C_2 as

$$d_{shift}(C_1, C_2) = \max_{i=1, \dots, k} d_{perp}(\mu_2, E_{1,i}),$$

with $E_{1,i}$ being the hyperplane equations of C_1 and μ_2 being the mean of all data points forming the subspace captured in C_2 .

4.4 Aging and Unification

Aging. Informally, the unification of two *Concepts* is the process of merging two subspace cluster representatives. However, when unifying two *Concepts* it is important to consider the importance of the *Concepts*, as for instance a very recent *Concept* is typically more important than a stale *Concept*, or a *Concept* that represents lots of data objects is more important than a *Concept* that represents only a few. Therefore, we introduce an *importance score* for each *Concept* that we use as weighting factor when merging two *Concepts*. Formally, we define the importance score of a *Concept* C as

$$\mathcal{I}(C) = e^{-\lambda \Delta t} \cdot N_C,$$

with λ being the decay parameter, Δt being the temporal difference between the current timestamp and the timestamp given in C , and N_C being the number of data objects that have been assigned to C . The first part of this equation, i.e., $e^{-\lambda \Delta t}$, is referred to as temporal part and contains the damping factor $\lambda > 0$. A high value of λ means low importance of old data and vice versa. The temporal part is also used to discard very old *Concepts* that are considered irrelevant for an up-to-date subspace clustering model. We therefore introduce a threshold θ that basically models a sliding window approach as a *Concept* whose temporal part falls below the threshold θ is discarded.

Unification. After extracting the new *Concepts* of a batch and recalculating the importance score of all *Concepts* in memory, we perform an unification step for the new *Concepts* and the previously extracted ones. Beginning at dimensionality $d - 1$, we compare all *Concepts* pairwise in terms of similarity and unify two *Concepts* if they are similar enough wrt some similarity threshold. The unification is continued in descending order regarding dimensionality. If two *Concepts* C_1 and C_2 of the same dimensionality can be unified, the following operations are performed to create the resulting *Concept* C^* :

- For each pair of equations $E_{1,i}$ and $E_{2,i}$ with $0 < i < d - l$, we define a new equation E_i^* by using the weighted mean of the normal vectors and the weighted mean of the distances to the origin of the two equations, i.e.,

$$E_i^* = \frac{\mathcal{I}(C_1) \cdot n_{E_{1,i}} + \mathcal{I}(C_2) \cdot n_{E_{2,i}}}{2} \cdot x + \frac{\mathcal{I}(C_1) \cdot r_{E_{1,i}} + \mathcal{I}(C_2) \cdot r_{E_{2,i}}}{2}.$$

This creates a new and possibly slightly shifted set of hyperplane equations.

- The mean representative for C^* is calculated by weighting the respective means from C_1 and C_2 with their importance, i.e.,

$$\mu_{C^*} = \frac{\mathcal{I}(C_1) \cdot \mu_{C_1} + \mathcal{I}(C_2) \cdot \mu_{C_2}}{2}.$$

- The number of data objects represented by C^* is the sum of data objects represented by C_1 and C_2 , i.e., $N_{C^*} = N_{C_1} + N_{C_2}$.
- The timestamp of C^* is set to the current timestamp, i.e., the timestamp of the younger *Concept* C_1 , such that $t_{C^*} = t_{C_1}$.
- The reference to the parent *Concept* of C^* will be set to the parent *Concept* of C_1 . Pointers of *Concepts* having C_1 or C_2 as parent are set to C^* .

As *Concepts* do not have to be identical wrt normal vectors and origin distances in order to trigger the unification, there will be some shifts of the yet found subspace clusters. In some applications it might be useful to record these shifts, e.g., to detect abnormal behaviors. CASHSTREAM enables the tracking of concept shift, since eventual drifts would result in rotations or parallel shifts of one or several plane equations describing the *Concept*. Hence, one simply has to record changes that may result from an unification of an old and a new *Concept* to get a history of changes in the underlying data distribution. However, this comes to the costs of requiring additional memory space.

5 Experiments

We evaluate CASHSTREAM by comparing the proposed streaming algorithm against the static counterpart *CASH* wrt the performance indicators accuracy, throughput and memory consumption. Those measures are important metrics for streaming methods as these methods typically aim at trading some accuracy for a drastically decreased memory consumption, or runtime.

Datasets. We use synthetic and real world datasets throughout this section. The synthetic dataset is a 4-dimensional set of points, containing two 2-dimensional planes of 1000 data points each, and 1000 random noise points. The planes both are jittered, making the data not perfectly correlated within their corresponding subspaces (as it appears in real world applications). The real-world dataset is a slightly manipulated version of the *wages dataset*. The original dataset has also been used in [1], and consists of 534 records each having four different features, i.e., age, years of education, years of experience and salary. However, we enlarge the dataset by copying and shuffling the records such that we have 40000 data points and finally can use the data to simulate a data stream appropriately.

Parameter Settings. We perform grid searches over various parameter settings and report the results for the best settings. Precisely, we range the parameters over the following sets: damping factor $\lambda \in \{.2, .5, .8\}$, temporal threshold $\theta \in \{.5, .8, 1\}$, singular value distance threshold $\tau_{SV\,dist} \in \{.005, .01, .02, .03\}$, and

equation shift distance threshold $\tau_{shift} \in \{.05, .1, .15\}$ ². The timestamp of a batch is set according to its number, i.e., the i -th batch gets timestamp i . The $minPts$ parameter that must be set for *CASH* is set proportionally to the batch size, i.e., $minPts = \tilde{m} \cdot s$, with \tilde{m} being the $minPts$ fraction and s being the batch size. The other *CASH* specific parameter $maxSplits$ is set according to the dataset at hand and reported for each experiment individually.

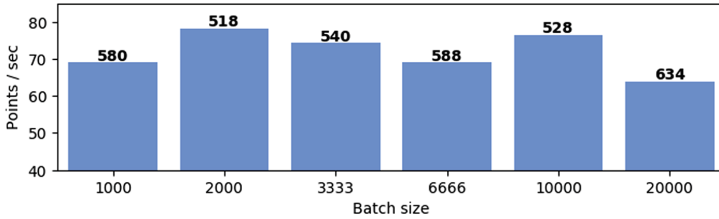


Fig. 3. Throughput for various batch sizes on the wages dataset; values above the bars are the absolute runtimes in sec; $maxSplits = 10$, $\tilde{m} = 0.2$.

Clustering Quality. For measuring the clustering quality of *CASHSTREAM*, we compare the results to a clustering on the same dataset for several different settings of the batch size parameter, including the batch size for which a single batch contains the entire dataset, which is equivalent to the static *CASH*. In terms of evaluation metrics, we employ the Adjusted Rand Index (*ARI*) and the Adjusted Mutual Information (*AMI*) scores.

Table 1. Results on the synthetic dataset. k is the number of batches, $maxSplits = 9$, $\tilde{m} = 0.3$.

Batch size	k	<i>ARI</i>	<i>AMI</i>
3000	1	0.951	0.922
1500	2	0.943	0.907
1000	3	0.924	0.881
750	4	0.875	0.829

Note that due to the lack of ground truth in the real-world dataset, we restrict ourselves to a synthetic dataset for evaluating the clustering quality. The calculated *ARI* and *AMI* for this dataset can be seen in Table 1. In general, it can be observed that the clustering quality slightly drops when choosing a batch size below 1000. This might indicate that the sub-sample might not reflect the data distribution sufficiently when choosing the batch size too small, which can be especially problematic in

scenarios where correlations are imperfect. Another reason for the decreasing clustering accuracy can be the presence of temporal effects (i.e., slight drifts in the data distribution, increasing amount of noise, etc.).

Runtime/Throughput. We investigate the actual throughput in terms of data points per second. In general, our evaluation of the throughput can be understood as a runtime comparison between the batched algorithm and the static *CASH*. For the throughput experiment, we used the enlarged real-world dataset to demonstrate the scalability of the batched streaming approach. In Fig. 3, we

² Note that those parameters are application dependent and thus not investigated in further detail.

report the throughput in data points per second and the total runtime in seconds. Each of the reported values is the mean value over three runs. For all those runs, we compared the resulting clustering models (by means of comparing the detected subspaces) with the expected clustering model and selected the parameter setting according to the best result. This experiment shows that the stream processing procedure has no loss in runtime compared to the static variant. In particular, it can be seen that the unification of *Concepts* barely has any effect on the runtime performance. We also observe that the batch size barely affects this performance measure.

Memory. As memory consumption is a critical metric for streaming applications, we show the monitored RAM usage of the batched approach and compare it to the static *CASH*. Precisely, we report the heap space usage profiles for both approaches as the memory usage at runtime is the decisive performance metric. The shown graphs were created using Java VisualVM 1.4.2, which is included in the Java JDK. To simulate a light-weight system, we cap the maximal available heap space to 2 GB (Fig. 4).

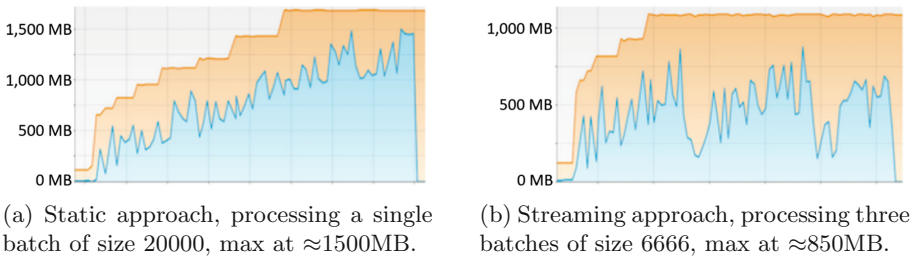


Fig. 4. Heap space usage profiles for the wages dataset ($maxSplits = 8$, $\tilde{m} = 0.2$).

For this experiment, we again use the enlarged wages dataset. This time the dataset consists of 20000 data points (augmented the same way as previously). Figure 4a shows the heap usage profile when using a single batch that contains all data points, resp. the static version, and Fig. 4b shows the profile when computing the same experiment with three batches.

For the static approach simulated in the full-sized 20000 points batch, the heap space rises steadily to a maximum level of around 1500 MB. When subdividing the points into 3 batches of 6666 points, we observe two crucial details: Firstly, the peak heap space usage is approx. 850 MB, which is significantly lower than in the static approach. Secondly, the three sequentially processed batches can clearly be identified as three peaks in the heap space profile.

6 Conclusion

In this work, we presented the novel subspace clustering algorithm *CASHSTREAM* that is able to deal with high-dimensional streaming data efficiently. Precisely,

CASHSTREAM relies on the subspace clustering paradigm that was introduced for the static *CASH* algorithm, i.e., using Hough transformations to identify interesting linear subspaces. However, in contrast to *CASH*, the proposed algorithm uses a batch processing scheme, identifies interesting subspaces within the data batches, and subsequently compresses important information within *Concept* data structures. Our experiments showed that CASHSTREAM is fairly robust against different choices for the batch size and simultaneously reduces the memory consumption significantly compared to the static *CASH* algorithm (less than 50% on the real-world dataset). At the same time the loss in terms of clustering quality is negligible.

Acknowledgement. This work has been funded by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IS18036A. The authors of this work take full responsibilities for its content.

References

1. Achtert, E., Böhm, C., David, J., Kröger, P., Zimek, A.: Global correlation clustering based on the Hough transform. *Stat. Anal. Data Min.* **1**(3), 111–127 (2008)
2. Achtert, E., Böhm, C., Kriegel, H.P., Kröger, P., Zimek, A.: On exploring complex relationships of correlation clusters. In: *Proceedings of SSDBM*, p. 7 (2007)
3. Aggarwal, C.C., Han, J., Wang, J., Yu, P.S.: A framework for clustering evolving data streams. In: *Proceedings of VLDB*, pp. 81–92 (2003)
4. Aggarwal, C.C., Han, J., Wang, J., Yu, P.S.: A framework for projected clustering of high dimensional data streams. In: *Proceedings of VLDB*, pp. 852–863 (2004)
5. Aggarwal, C.C., Yu, P.S.: Finding generalized projected clusters in high dimensional spaces, vol. 29 (2000)
6. Böhm, C., Kailing, K., Kriegel, H.P., Kröger, P.: Density connected clustering with local subspace preferences (2004)
7. Böhm, C., Kailing, K., Kröger, P., Zimek, A.: Computing clusters of correlation connected objects. In: *Proceedings of SIGMOD*, pp. 455–466 (2004)
8. Borutta, F., Kröger, P., Hubauer, T.: A generic summary structure for arbitrarily oriented subspace clustering in data streams. In: Amato, G., Gennaro, C., Oria, V., Radovanović, M. (eds.) *SISAP 2019. LNCS*, vol. 11807, pp. 203–211. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-32047-8_18
9. Bradley, P.S., Fayyad, U.M., Reina, C., et al.: Scaling clustering algorithms to large databases. In: *Proceedings of KDD*, vol. 98, pp. 9–15 (1998)
10. Cao, F., Ester, M., Qian, W., Zhou, A.: Density-based clustering over an evolving data stream with noise. In: *Proceedings of SDM*, vol. 6, pp. 328–339 (2006)
11. Corwin, L.: *Multivariable Calculus*. Routledge, London (2017)
12. Guha, S., Mishra, N., Motwani, R., o’Callaghan, L.: Clustering data streams. In: *Proceedings of FOCS*, pp. 359–366 (2000)
13. Hassani, M., Spaus, P., Gaber, M.M., Seidl, T.: Density-based projected clustering of data streams. In: Hüllermeier, E., Link, S., Fober, T., Seeger, B. (eds.) *SUM 2012. LNCS (LNAI)*, vol. 7520, pp. 311–324. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33362-0_24
14. Kazempour, D., Mauder, M., Kröger, P., Seidl, T.: Detecting global hyperparaboloid correlated clusters: a Hough-transform based multicore algorithm. *Distrib. Parallel Databases* **39**, 37–72 (2018). <https://doi.org/10.1007/s10619-018-7246-0>

15. Kriegel, H.P., Kröger, P., Ntoutsi, I., Zimek, A.: Towards subspace clustering on dynamic data: an incremental version of PreDeCon. In: Proceedings of International Workshop on Novel Data Stream Pattern Mining Techniques, pp. 31–38 (2010)
16. Ntoutsi, I., Zimek, A., Palpanas, T., Kröger, P., Kriegel, H.P.: Density-based projected clustering over high dimensional data streams. In: Proceedings of SDM, pp. 987–998 (2012)
17. O’Callaghan, L., Mishra, N., Meyerson, A., Guha, S., Motwani, R.: Streaming-data algorithms for high-quality clustering. In: Proceedings of ICDE, pp. 685–694 (2002)
18. Park, N.H., Lee, W.S.: Grid-based subspace clustering over data streams. In: Proceedings of CIKM, pp. 801–810 (2007)
19. Rosenfeld, A.: Picture processing by computer, vol. 1, pp. 147–176. ACM (1969)
20. Scheid, H., Schwarz, W.: Elemente der linearen Algebra und der Analysis (2009)
21. Silva, J.A., Faria, E.R., Barros, R.C., Hruschka, E.R., De Carvalho, A.C., Gama, J.: Data stream clustering: a survey. *ACM Comput. Surv.* **46**(1), 13 (2013)
22. Zhang, T., Ramakrishnan, R., Livny, M.: BIRCH: an efficient data clustering method for very large databases. *ACM Sigmod Rec.* **25**, 103–114 (1996)
23. Zhou, A., Cao, F., Qian, W., Jin, C.: Tracking clusters in evolving data streams over sliding windows. *Knowl. Inf. Syst.* **15**(2), 181–214 (2008). <https://doi.org/10.1007/s10115-007-0070-x>