International Journal of Data Science and Analytics (2022) 14:243–259
https://doi.org/10.1007/s41060-022-00327-y

REGULAR PAPER

RASCL: a randomised approach to subspace clusters

Sandy Moens[1,2] · **Boris Cule**[3,4] · **Bart Goethals**[2,5]

Received: 3 May 2021 / Accepted: 15 April 2022 / Published online: 11 May 2022
© The Author(s) 2022

Abstract
Subspace clustering aims to discover clusters in projections of highly dimensional numerical data. In this paper, we focus on discovering small collections of highly interesting subspace clusters that do not try to cluster all data points, leaving noisy data points unclustered. To this end, we propose a randomised method that first converts the highly dimensional database to a binarised one using projected samples of the original database. Subsequently, this database is mined for frequent itemsets, which we show can be translated back to subspace clusters. In this way, we are able to explore multiple subspaces of different sizes at the same time. In our extensive experimental analysis, we show on synthetic as well as real-world data that our method is capable of discovering highly interesting subspace clusters efficiently.

Keywords Subspace clusters · High-dimensional data · Sampling · Maximal itemsets

1 Introduction

Clustering is an important field within data mining research. The main task of clustering is to group similar objects together, while keeping sufficiently different objects apart [1]. However, due to the well-known *curse of dimensionality* [2], traditional clustering methods struggle when encountering high-dimensional data. In short, with high-dimensional data, the distances between pairs of objects, measured over all dimensions, become increasingly similar. As a result, no proper clusters can be formed, as all objects end up almost equally distant from each other. In other words, while clustering attempts to find localised neighbourhoods of objects, there can be no talk of such neighbourhoods in high-dimensional space. In a time when companies and organisations collect more and more data about its customers, suppliers, operational processes, etc., avoiding the curse of dimensionality is becoming more and more important.

For example, a company may keep track of over a hundred attributes to describe their customers, and, while many customers may share some of these attributes, there will always be enough attributes that make them sufficiently different from each other to be considered similar. However, it is crucial that the company groups its customers into segments in order to, for example, apply targeted marketing campaigns, or conduct meaningful surveys.

Subspace clustering attempts to solve this problem by trying to discover clusters of objects that are similar in a limited number of dimensions [3]. However, given the exponential complexity of the search space, identifying the relevant set of dimensions is computationally demanding: with m-dimensional data, there are $2^m - 1$ possible sets of dimensions within which clusters could be found, which is why existing subspace clustering methods suffer from long run-times [4]. Furthermore, some existing approaches produce full clusterings, thereby ensuring that each data object is assigned to exactly one cluster. This is not always desirable, for two reasons. First, the data may contain a lot of noise that should ideally not be assigned to any cluster. Second, there is no reason why a particular object should not be assigned to multiple

✉ Sandy Moens
sandy.moens@gmail.com

✉ Boris Cule
b.cule@tilburguniversity.edu

✉ Bart Goethals
bart.goethals@uantwerpen.be

1 ENSEK Benelux, Beringen, Belgium

2 Department of Computer Science, Universiteit Antwerpen, Antwerpen, Belgium

3 Department of Cognitive Science and Artificial Intelligence, Tilburg University, Tilburg, Netherlands

4 Department of Accountancy and Finance, Universiteit Antwerpen, Antwerpen, Belgium

5 Faculty of Information Technology, Monash University, Melbourne, Australia

clusters, especially if the sets of dimensions that define these clusters are entirely different.

For example, a customer could be part of a group targeted by one advertising campaign based on gender, income and marital status, and of another group targeted by an entirely different campaign based on location, being a parent, and having an interest in sports.

In this paper, we take a similar approach to the problem as the CartiClus[5] method: we first convert a numeric database to a transactional one and subsequently use frequent pattern mining to extract subspace clusters. Our method can efficiently produce highly interesting subspace clusters, along with the dimensions that define them. Moreover, we allow objects to be part of multiple clusters, and we leave objects that are not similar to any other objects in any set of dimensions unclustered. We avoid the computational complexity of existing subspace clustering methods by deploying a randomised algorithm. In the first step, we take a large number of samples from the original data, such that each sample consists of a number of objects in a fixed (random) set of dimensions. (Other dimensions are discarded.) In each sample, we then cluster the objects and subsequently assign all objects in the original data to the nearest cluster centroid. This produces a set of objects per centroid, which we interpret as a transaction. By merging the transactions produced for all different samples, we obtain a transaction database. This process is called binarisation of the data. Note that by using a large number of samples, this method will produce relatively many highly similar transactions. We then randomly sample maximal frequent itemsets from this database to obtain potential subspace clusters. Finally, we identify the relevant dimensions for each discovered cluster.

We perform an extensive experimental evaluation on a variety of datasets. First of all, we compare two variants of our algorithm to see whether additional computational effort results in better clustering performance. We evaluate the performance on two aspects—first, *object quality*, that measures whether objects that belong together are, in fact, placed into the same clusters, and, second, *dimension quality*, which measures to which extent the relevant dimensions are correctly identified for the discovered clusters. Furthermore, we perform experiments to evaluate how various parameter settings affect the performance of our algorithm, and provide the reader with guidance on how to set sensible parameter values without any knowledge of clusters that may or may not be present in the data. Finally, we compare our algorithm to CartiClus and ProClus, two state-of-the-art techniques for subspace clustering. We conclude that, while these methods produce comparable results with optimal parameter settings, our algorithm is much less susceptible to producing poor results if the parameter values are changed. In an unsupervised learning setting, where no ground truth is available to the end user, this can be crucial. Without accidentally stumbling upon the correct parameter settings, CartiClus and ProClus will produce dramatically poorer results, while our method remains relatively robust with a variety of settings.

A preliminary version of this paper was published as *A Sampling-based Approach for Discovering Subspace Clusters* at the 2019 Discovery Science Conference [6]. Here, we extend our work with new material, both in terms of the theoretical analysis and the experimental evaluation. We now elaborate on the description of our own method by including the pseudocode, as well as by adding examples to illustrate the discussed concepts. Additionally, Sect. 3.3, discussing how our method could be generalised to using any clustering and classification algorithm, is entirely new. Similarly, the experimental section of our paper has been significantly expanded and now contains more than 20 new sets of experiments, comparing our algorithm to existing methods on a variety of evaluation metrics, as well as a thorough discussion of the achieved results.

The main contributions of this paper can be summarised as follows:

– we propose a randomised sampling algorithm that efficiently identifies localised clusters and their relevant dimensions,
– we simultaneously explore multiple subspaces, thus significantly reducing the required effort,
– we allow data objects to be part of multiple clusters, and we leave noise objects unclustered,
– we perform a theoretical evaluation to show the efficiency of our method and an extensive experimental evaluation to show the quality of the output.

The remainder of the paper is organised as follows. In Sect. 2, we introduce the necessary notations and describe the problem. Section 3 provides a thorough description of our algorithm, which we experimentally evaluate in Sect. 4. We discuss the related work in Sect. 5, before concluding the paper in Sect. 6.

2 Background

In this section, we introduce the necessary definitions and notations from the fields of subspace clustering and frequent itemset mining. We also discuss and formally define the quality measures used to evaluate our experimental results.

2.1 Subspace clustering

Let $\mathcal{D} = \{D_1, \ldots, D_m\}$ be a set of m dimensions. Each dimension D_i comes equipped with a domain $dom(D_i)$. An m-dimensional data point $p = (d_1, \ldots, d_m)$ is a tuple of values over $\mathcal{D}$, such that $d_i \in dom(D_i)$ for each $i = \{1, \ldots, m\}$. The input database $\mathcal{P} = (p_1, \ldots, p_q)$ contains a collection of q such m-dimensional data points. Furthermore, each dimension D_i comes equipped with a distance function $\delta_{D_i} : dom(D_i) \times dom(D_i) \rightarrow \mathbb{R}^+$. Additionally, we assume that for any subset of dimensions $\mathtt{D} = \{\mathtt{D}_1, \ldots, \mathtt{D}_l\}$, with $1 \leq l \leq m$ and $\mathtt{D} \subseteq \mathcal{D}$ there exists a distance function $\delta_{\mathtt{D}} : (dom(\mathtt{D}_1) \times \ldots \times dom(\mathtt{D}_l)) \times (dom(\mathtt{D}_1) \times \ldots \times dom(\mathtt{D}_l)) \rightarrow \mathbb{R}^+$. All used distance functions must satisfy the usual conditions (non-negativity, identity, symmetry, and the triangle inequality). In our examples we use $\mathbb{R}$ as the domain for all the dimensions and we use the Euclidean distance as a distance function. Given a subset of dimensions $\mathtt{D} \subseteq \mathcal{D}$, we denote by $p^{\mathtt{D}}$ a data point, and by $\mathtt{P}^{\mathtt{D}}$ a set of data points, projected onto the given dimensions. We define a *subspace cluster* as follows.

Definition 1 (Subspace cluster) A subspace cluster is a tuple containing a subset of data points and a subset of dimensions, formally

$$S = (\mathtt{P}, \mathtt{D}), \text{ with } \mathtt{P} \subseteq \mathcal{P} \text{ and } \mathtt{D} \subseteq \mathcal{D}.$$

This definition forms a logical extension of a traditional cluster by incorporating the relevant dimensions in which the data points form a cluster. Even more, this definition leans itself towards our itemset-based algorithm described in the next section. However, for some of the measures described in the next section, the following, alternative, set based definition of tuples is more natural and easier to use [7,8]. We denote this set based representation as:

$$S^{\mathfrak{T}} = \{(p_i, D_j) \,|p_i \in \mathtt{P} \wedge D_j \in \mathtt{D} \wedge (\mathtt{P}, \mathtt{D}) \in S\}.$$

Furthermore, let $\mathfrak{T}$ be a mapping from a tuple-based representation of a subspace cluster to a set-based representation, i.e., $\mathfrak{T}(S) = S^{\mathfrak{T}}$.

In this work, we are interested in clusters for which the data points $\mathtt{P}$ lie in close proximity to one another in $\mathtt{D}$. Given the definition of a subspace cluster, the problem we tackle is to discover a concise collection of subspace clusters that have a high precision on both the discovered data points and on the set of dimensions.

2.2 Subspace clustering measures

For the sake of completeness, we add the definitions of the quality measures used throughout the experiments. We refer to the original paper [8] for an in depth study of these measures.

There are 3 flavours of quality measures depending on the type of information of a subspace cluster they operate on: measures taking into account only object (data point) information are sub-scripted with OBJ, measures taking into account only dimension information are sub-scripted with DIM, and measures taking into account information of both objects and dimensions are sub-scripted with SC.

Given $S_R = (\mathtt{P}_R, \mathtt{D}_R)$ a random subspace cluster and $S_G = (\mathtt{P}_G, \mathtt{D}_G)$ the ground truth subspace cluster, the following measures are defined over the objects:

$$\begin{aligned}\text{RECALL}_{\text{OBJ}}(S_R, S_G) &= \frac{|\mathtt{P}_R \cap \mathtt{P}_G|}{|\mathtt{P}_G|} \\ &= \text{PRECISION}_{\text{OBJ}}(S_G, S_R).\end{aligned}$$

Given the same two subspace clusters the following measures are defined on the dimensions:

$$\begin{aligned}\text{RECALL}_{\text{DIM}}(S_R, S_G) &= \frac{|\mathtt{D}_R \cap \mathtt{D}_G|}{|\mathtt{D}_G|} \\ &= \text{PRECISION}_{\text{DIM}}(S_G, S_R).\end{aligned}$$

Finally, given the same two subspace clusters the following measures are more easily defined on the set based representation of the subspace cluster $S^{\mathfrak{T}}$. We use the mapping $\mathfrak{T}$ to switch between the representations:

$$\begin{aligned}\text{RECALL}_{\text{SC}}(S_R, S_G) &= \frac{|\mathfrak{T}(S_R) \cap \mathfrak{T}(S_G)|}{|\mathfrak{T}(S_G)|} \\ &= \text{PRECISION}_{\text{SC}}(S_G, S_R).\end{aligned}$$

The F1 measure is the harmonic mean between precision and recall and is defined as:

$$\begin{aligned}&\text{F1}_*(S_R, S_G) \\ &= \frac{2 \times \text{RECALL}_*(S_R, S_G) \times \text{PRECISION}_*(S_R, S_G)}{\text{RECALL}_*(S_R, S_G) + \text{PRECISION}_*(S_R, S_G)}\end{aligned}$$

with $* \in \{\text{OBJ}, \text{DIM}, \text{SC}\}$.

Given a set of clusters $\boldsymbol{\mathcal{S}}_R$ and a set of ground truth clusters $\boldsymbol{\mathcal{S}}_G$ the PRECISION_*, RECALL_* and F1_* can be defined on set level by assigning each cluster from $\boldsymbol{\mathcal{S}}_R$ to the closest cluster $\boldsymbol{\mathcal{S}}_G$ given the quality measure. A set quality measure $\text{Q}^S \in \{\text{PRECISION}_*^S, \text{RECALL}_*^S, \text{F1}_*^S\}$ on set level is formally defined as

$$\text{Q}^S(\boldsymbol{\mathcal{S}}_R, \boldsymbol{\mathcal{S}}_G) = \frac{1}{|\boldsymbol{\mathcal{S}}_G|} \sum_{S_R \in \boldsymbol{\mathcal{S}}_R} \underset{S_G \in \boldsymbol{\mathcal{S}}_G}{\text{argmax}}\, \text{Q}(S_R, S_G)$$

with Q the non-set counterpart.

Finally, the subspace clustering measure M^{E4SC} [8] on set level is formally defined as

$$\text{M}^{E4SC}(\boldsymbol{\mathcal{S}}_R, \boldsymbol{\mathcal{S}}_G) = \frac{2 \times \text{F1}_{\text{SC}}^S(\boldsymbol{\mathcal{S}}_R, \boldsymbol{\mathcal{S}}_G) \times \text{F1}_{\text{SC}}^S(\boldsymbol{\mathcal{S}}_G, \boldsymbol{\mathcal{S}}_R)}{\text{F1}_{\text{SC}}^S(\boldsymbol{\mathcal{S}}_R, \boldsymbol{\mathcal{S}}_G) + \text{F1}_{\text{SC}}^S(\boldsymbol{\mathcal{S}}_G, \boldsymbol{\mathcal{S}}_R)}.$$

2.3 Frequent itemset mining

Let $\mathcal{I} = (i_1, ...i_n)$ be a finite set of n items. A *transaction* t is a subset of items. We denote by $\mathcal{T} = (t_1, ..., t_o)$ a database of o transactions. An *itemset* $\mathtt{I}$ is also a subset of items. A transaction t is said to support an itemset $\mathtt{I}$ if $\mathtt{I} \subseteq t$. The set of all transactions in $\mathcal{T}$ that support an itemset is called the *cover* of that itemset, i.e., $cov(\mathtt{I}) = \{t \mid t \in \mathcal{T} \wedge \mathtt{I} \subseteq t\}$. The *support* of an itemset is the size of its cover, i.e., $sup(\mathtt{I}) = |cov(\mathtt{I})|$. Frequent itemsets and maximal frequent itemsets are defined as follows.

Definition 2 (Frequent itemset) Given a minimal support threshold $\sigma \geq 0$, an itemset $\mathtt{I}$ is frequent if its support is larger than or equal to σ, i.e., $sup(\mathtt{I}) \geq \sigma$.

Definition 3 (Maximal frequent itemset) A frequent itemset $\mathtt{I}$ is called maximal if there exists no strict superset of $\mathtt{I}$ that is also frequent with respect to σ.

The anti-monotonic property of the support of itemsets guarantees that all subsets of a frequent itemset are also frequent.

3 Randomised subspace clusters

Existing methods for discovering subspace clusters from numeric data often focus on the complete raw dataset to compute subspace clusters using a bottom-up [4,9,10] or a top-down approach [11]. In this paper we introduce RASCL, which takes a different approach to the problem by using randomised subsets of the data (both in the data points and in the dimensions) as a starting point for detecting subspace clusters. The discovered clusters are then checked for occurrence in multiple subsamples of the data. If a cluster occurs frequently enough in the set of samples we output it as a subspace cluster. Our algorithm relies on two simple premises: 1) higher-dimensional subspace clusters also form subspace clusters in lower dimensions, which is also the basis for other bottom-up approaches to subspace clustering [9]; 2) if we take enough random samples and use them to detect clusters, a lot of similar subclusters of the same true cluster will be found in different projections. Moreover, by repeating such a randomised procedure many times we end up with a stable solution.

In a nutshell, our proposed algorithm consists of three consecutive steps:

1) convert a numerical database to a transactional one using K-means clustering (Section 3.1),
2) extract interesting subspace clusters from the transactional database using frequent itemset mining (Section 3.2),
3) heuristically select the most interesting subspace clusters (Section 3.4).

3.1 Randomised data transformation

3.1.1 Data binarisation

To binarise a numeric database $\mathcal{P}$ into a transaction database $\mathcal{T}$ we use the indices of data points as the items for $\mathcal{T}$, resulting in $|\mathcal{P}|$ items. In addition, we obtain a mapping between data points and items. Ideally, a transaction contains data points that are close to each other in some set of dimensions. In that case, an itemset (essentially a set of data points) that occurs in a large fraction of transactions can be seen as a subspace cluster over some set of dimensions.

We define a randomised process for constructing a single transaction database. We repeat this process n times and concatenate all transactions into a single database $\mathcal{T}^*$. We first sample a small subset of data points $\mathtt{P}$ and a small subset of dimensions $\mathtt{D}$. (The sampling strategy is explained below.) The data points are projected onto the subset of dimensions and used as input for the K-means clustering algorithm. The resulting cluster centroids are used to partition the original data points, assigning each data point to the closest centroid. As such, each centroid represents one transaction in the transactional database and its items are the data points assigned to it. Formally, for a set of centroids $C^{\mathtt{D}}$ found by K-means, the closest centroid for a projected data point $p^{\mathtt{D}}$ is given by

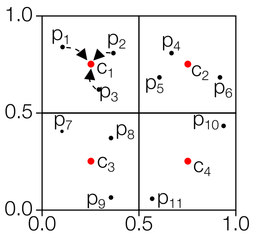
$$c^{p^{\mathtt{D}}} = \operatorname*{argmin}_{c^{\mathtt{D}} \in C^{\mathtt{D}}} (\delta_{\mathtt{D}}(p^{\mathtt{D}}, c^{\mathtt{D}})).$$

In the unlikely event that multiple centroids are the closest, we take the first one that was encountered.

Example 1 An example conversion from a numerical database to a binarised one is shown in Fig. 1. The figure shows a toy database of 11 data points in a 2D space. A red circle represents a synthetic cluster centroid and the surrounding square visually shows data points closest to that centroid. When constructing the binarised database, the index of a data point is added to the transaction of the nearest cluster centroid. The resulting transaction database is shown in Fig. 1b. For example, for p_1, p_2 and p_3 the closest centroid is c_1 resulting in the first transaction.

3.1.2 Generating data samples

As mentioned previously, our binarisation strategy requires a sample of data points and a sample of dimensions. The main question now is how we can bias the sampling procedure to obtain samples that will have a higher potential to contain cluster structures.



(a)

centroid id	items
c_1	1, 2, 3
c_2	4, 5, 6
c_3	7, 8, 9
c_4	10, 11

(b)

Fig. 1 **a** A fictitious example dataset with 2 dimensions, 11 data points (black dots) and 4 centroids (red circles). **b** Binarised dataset in short format for the toy dataset

For the data points, we can sample k data points uniformly at random. By repeating this a large number of times, we expect each cluster to be represented by a sufficient number of data points in a high enough number of samples.

For the dimensions, a naive solution would be to sample uniformly at random a subset of dimensions of size x, with $1 \le x \le |\mathcal{D}|$. However, since the number of combinations larger than 2 can blow up, a random sample of dimensions will likely be too large to contain a meaningful cluster (e.g., if there are 100 dimensions, there are many more subsets of size 50 than of size 2, and a subspace cluster is, on the other hand, more likely to be found in low-dimensional space). The probability that such a uniformly sampled set contains a subspace cluster will decrease rapidly when the size of $\mathcal{D}$ increases. On the other hand, sampling just one dimension may result in discovering cluster structures that do not span multiple dimensions. Therefore, at this stage, we sample a set of two dimensions, and, if a cluster spans over more than two dimensions, the relevant set of dimensions will be identified at a later stage (see Sect. 3.2). We apply weighted sampling to boost the probability of sampling dimensions that contain cluster structures. Similar to Moise et al. [4], we assume that uniformly distributed dimensions do not contain any cluster structure. As such, to detect non-uniformity of a dimension we create a histogram using the Freedman–Diaconis' rule [12] to compute an appropriate number of bins for the data. This rule is robust to outliers and does not assume data to be normally distributed. Let us denote by B^D the bins for a given dimension using the Freedman–Diaconis' rule and let $|b|$ denote the number of data points falling in bin b. The unnormalised sampling potential $\mathcal{W}$ of a dimension is given by

$$\mathcal{W}(D) = \sqrt{\frac{|\{b \mid b \in B^D \wedge |b| \le \frac{|\mathcal{P}|}{|B^D|}\}|}{|B^D|}}. \quad (1)$$

The logic behind this formula is that we wish to prioritise those dimensions in which the points are not uniformly distributed, as they are more likely to contain clusters. Note that if the points are uniformly distributed, the expected number of points in each bin would be $\frac{|\mathcal{P}|}{|B^D|}$. However, if a large number of points was clustered in a single bin, with the remaining points uniformly distributed across the remaining bins, then all those remaining bins would contain fewer than $\frac{|\mathcal{P}|}{|B^D|}$ points. We therefore compute how many bins contain fewer than the number of expected data points under uniform data distribution. The larger the number of such bins in a given dimension D, the higher the value of $\mathcal{W}(D)$. The resulting distribution thus favours dimensions with a larger clustering potential.

Algorithm 1: BinariseDataset($\mathcal{P}$, k, K)

```
Input: $\mathcal{P}$ database of data points, $k$ number of data points to sample, $K$ number of clusters
Result: Transaction dataset $\mathcal{T}$
1 $\mathcal{T} \longleftarrow []$;
2 $\mathrm{P} \longleftarrow k$ random i.i.d. samples from $\mathcal{P}$;
3 $\mathrm{D} \longleftarrow$ 2 different dimensions sampled from $\mathcal{D}$ with sampling potential cfr. Eq. 1;
4 $C^{\mathrm{D}} \longleftarrow K$ centroids using $K$-means given projected data points $\mathrm{P}^{\mathrm{D}}$;
5 for $c^{\mathrm{D}} \in C^{\mathrm{D}}$ do
6     $\mathcal{T} \longleftarrow \mathcal{T} + [\{p | p \in \mathcal{P}, c^{p^{\mathrm{D}}} = \operatorname{argmin}_{c^{\mathrm{D}} \in C^{\mathrm{D}}}(\delta_{\mathrm{D}}(p^{\mathrm{D}}, c^{\mathrm{D}}))\}]$;
7 end
8 return $\mathcal{T}$;
```

Note that randomly sampling sets of dimensions also makes our algorithm robust to missing values. Since we explore various sets of dimensions, missing values have very limited impact on our method. Concretely, a missing value in a particular dimension would only prevent a point from being included in a cluster that is defined by that dimension. Such a missing value would play no part at all in evaluating whether the point belongs to a cluster that is not defined by that dimension.

3.1.3 Pseudocode

The procedure for binarisation is given in Algorithm 1. It takes as input a database of data points, a number k of data points to sample and parameter K used for clustering. Line 1 initialises the list of transactions, line 2 samples the data points and line 3 samples the dimensions. Line 4 runs K-means on the projected data points and stores the projected centroids. Lines 5 to 7 create single transactions per cluster centroid, by adding all data point indices to whichever cluster centroid is closest. The binarised database is returned on line 8.

3.1.4 Time complexity

The worst case complexity of our binarisation method is mostly dependent on K-means. However, we use only a small subset of data points, typically $|\mathtt{P}| \ll |\mathcal{P}|$, to compute cluster centroids. For this small subset the complexity for clustering is $\mathcal{O}(n \times (|\mathtt{P}| \times |\mathtt{D}| \times K \times i))$ with n the number of database samples and i the number of iterations. The generation of samples for data points can be done in $\mathcal{O}(|\mathcal{P}|)$ and for dimensions can be done in $\mathcal{O}(|\mathcal{D}|)$. The assignment of data points to cluster centroids is done in a single sweep, i.e., $\mathcal{O}(K \times |\mathcal{P}|)$. The total time complexity for generating samples and binarising the database is $\mathcal{O}(K \times |\mathcal{P}| + |\mathcal{D}| + n \times (|\mathtt{P}| \times |\mathtt{D}| \times K \times i))$.

3.2 Extracting subspace clusters

We previously constructed a binarised database $\mathcal{T}^*$ by concatenating n binarised ones built using random samples of data points and dimensions. The premise for each database is that its transactions represent cluster centroids and their items are indices of data points in their close proximity for the set of dimensions. Since we generated n samples, we know that each index occurs n times within $\mathcal{T}^*$. If then a set of items occurs often together in the database, i.e., it is a frequent itemset with high support, then we know that in many sets of dimensions the same set of data points occur in close proximity, which is exactly the objective for a subspace cluster. This means essentially that every frequent itemset $\mathtt{I}$ for which $sup(\mathtt{I}) \geq \sigma$, with $0 < \sigma \leq n$, represents a subspace cluster in a currently unknown set of dimensions. However, the number of frequent itemsets is typically huge, largely because all subsets of frequent itemsets are frequent. To alleviate this problem we use maximal itemsets and, more particularly, our algorithm samples μ maximal frequent itemsets from the binarised database. The resulting itemsets are the data points for subspace clusters.

An effective method for sampling maximal frequent itemsets was introduced by Moens and Goethals [13]. It iteratively extends an itemset with new items, until the set is found to be maximal given a threshold τ and a monotonic quality measure (e.g., support). In each step a probability distribution is computed over the remaining items (i.e., augmentations that result in a score $\geq \tau$) given a quality function (e.g., the support of the itemset augmented with the item). After sampling a single item from the distribution, the itemset is updated and remaining items that result in a score $\leq \tau$ are discarded. Next, the distribution is recomputed and the process is repeated until the list of remaining items is empty. The resulting itemset is maximal by construction.

After extracting a collection of data points, the next step is to discover the dimensions in which the data points form a cluster. In contrast to some existing methods [4,11], we do not require to go back to the data itself to check each dimension individually, since our binarisation process preserved some essential information that can guide us here. That is, our algorithm previously sampled collections of dimensions which can be reused to determine a valid subset of dimensions. We denote by $dims(t)$ a map that for a transaction returns its *linked dimensions*, i.e., the dimensions that were used for its construction in the binarisation process. For a maximal itemset $\mathtt{I}$ we can use the transactions in its cover to determine its *relevant dimensions*, i.e., the set containing all linked dimensions for transactions in $cov(\mathtt{I})$. Formally, $dims(\mathtt{I}) = \{d | d \in \mathcal{D} \land d \in dims(t) \land t \in cov(\mathtt{I})\}$. An itemset $\mathtt{I}$, mapped to the data points $\mathtt{P}$, forms together with its relevant dimensions the subspace cluster $S = (\mathtt{P}, dims(\mathtt{I}))$.

Algorithm 2: FindSubspaceClusters($\mathcal{T}$, σ, μ)

```
Input: $\mathcal{T}$ transactional database, $\sigma$ minimum support, $\mu$
       number of subspace clusters to find
Result: List of $\mu$ subspace clusters
1 $C \leftarrow []$;
2 for $i \in [1, ..., \mu]$ do
3     $\mathtt{I} \leftarrow$ random maximal itemset from $\mathcal{T}$
        with minimum support $\sigma$ [13];
4     $\mathtt{P} \leftarrow$ data points determined by $\mathtt{I}$;
5     $\mathtt{D} \leftarrow dims(\mathtt{I})$;
6     $C \leftarrow C + (\mathtt{P}, \mathtt{D})$;
7 end
8 return $C$;
```

3.2.1 Pseudocode

The pseudocode for extracting subspace clusters is given in Algorithm 2. Line 1 initialises the list of subspace clusters. Lines 2 to 7 extract μ subspace clusters by first sampling a maximal itemset from $\mathcal{T}$ (line 3) and mapping the itemset to the real data points (line 4). Then the relevant dimensions are computed in line 5 and the subspace cluster is added to the list of clusters in line 6. Finally, line 8 returns all extracted subspace clusters.

3.3 General approach to binarisation

Above, we described the binarisation stage of our algorithm. In short, we transform the multi-dimensional dataset into a transaction database by first selecting a number of random points from the data, then using K-means to cluster those points, and, finally, assigning all the other original points to the nearest cluster centroid. This final step could also be seen as a 1-nearest neighbour classification of those remaining points. More generally, our algorithm is an instance of a procedure where clustering is performed on a small subset of points, and those clusters are then used as class labels to classify the remaining points.

In our case, we chose K-means clustering and 1-nearest neighbour classification as they are the simplest and the most efficient clustering and classification algorithms, respectively. However, in principle, there is no reason not to use any clustering algorithm for the first step or any classifica-

Table 1 An overview of the main characteristics of the synthetic datasets that have been used throughout the experiments

	#rows	#dimensions	#clusters	#objects/cluster (avg)	#dimensions/cluster (avg)
$\text{dbsizescale}_{s1500}$	1,595	20	10	166.3	14.0
$\text{dbsizescale}_{s2500}$	2,658	20	10	276.5	14.0
$\text{dbsizescale}_{s3500}$	3,722	20	10	385.8	14.0
$\text{dbsizescale}_{s4500}$	4,785	20	10	496.2	14.0
$\text{dbsizescale}_{s5500}$	5,848	20	10	608.5	14.0
dimscale_{d05}	1,595	5	10	182.6	3.5
dimscale_{d10}	1,595	10	10	181.5	6.7
dimscale_{d25}	1,595	25	10	180.9	16.9
dimscale_{d50}	1,595	50	10	181.6	33.5
dimscale_{d75}	1,595	75	10	181.9	50.4
noisescale_{n10}	1,611	20	10	166.5	14.6
noisescale_{n30}	2,071	20	10	166.1	14.6
noisescale_{n50}	2,900	20	10	166.3	14.6
noisescale_{n70}	4,833	20	10	166.8	14.6

tion algorithm for the second step. Obviously, an advantage of K-means is that it produces natural centroids, with which the coordinates of the remaining points can easily be compared. Clusters produced by other clustering algorithms may require different characterisation techniques, just as other classification algorithms may rely on different ways to assign class labels to the remaining points. It is beyond the scope of this paper to analyse how combinations of various clustering and classification algorithms would perform in our problem setting, but this does remain a potential avenue of future research.

Additionally, in the interest of efficiency, it is even possible to skip the clustering step altogether. In the extreme case, the chosen values for k (the number of randomly selected points) and K (the desired number of clusters) could be the same. In this case, each of the points would represent an initial centroid for K-means, and the clustering algorithm would end after one iteration, resulting in each cluster containing exactly one point—namely, its original centroid. Naturally, in this case, it would make no sense to even call the clustering algorithm. In short, this method boils down to simply selecting K random points and then building a transaction database by assigning all other points to the nearest point. In Sect. 4, we experimentally compare this approach to using K-means on k random points and refer to this version of the algorithm as RASCL$_R$.

3.4 Selecting the best subspace clusters

After discovering a large number of subspace clusters (depending on parameter μ), we finally select a small collection of r clusters that can be deemed the most interesting subspace clusters. The number of data points that is present in the subspace cluster is an indication that the same set of data points are often related even in different subsets of dimensions. That is, even though the different samples have different dimension sets, the data points in the resulting binarised database occur in transactions for similar cluster centroids. In our method we will employ this heuristic (i.e., the larger the cluster, the better) for sorting discovered subspace clusters. In Sect. 4.3, we experimentally demonstrate that this sorting strategy works well. Finally, to reduce redundancy in the cluster results, we sequentially evaluate each cluster and select those clusters that have less than 25% cluster overlap with previously selected ones. Note that when sorting the clusters using the number of objects, this results in smaller clusters as r increases. Finally, we exclude very small clusters with fewer than 10 data points.

4 Experiments

We implemented RASCL in Python, but depend on a Java library for sampling maximal frequent itemsets. Our source code and all accompanying experiments have been made available online[1]. In our experiments, we use synthetic benchmark data provided by Günnemann et al. in their overview paper on subspace clustering [8]. This data is used to evaluate the performance of subspace clustering techniques with respect to their invariance to size, dimensionality and noise. The characteristics of the datasets are shown in Table 1. To evaluate the performance of the various techniques we measure PRECISION$_{\text{OBJ}}$ and RECALL$_{\text{OBJ}}$ scores on object (data point) level, as well as their harmonic mean F1$_{\text{OBJ}}$. These measures evaluate the correctness of the objects

[1] Source code and experiments are available via https://gitlab.com/adrem/rascl

defined by the subspace clusters with respect to some ground truth. Additionally we use their dimensionality aware counterparts which are indicated by the subscripts DIM (for scores about the dimensions) and SC (for scores about the combination of objects and dimensions) [8]. Finally, we also use M^{E4SC} [8], a measure to assess the quality of clusterings. Note that unless stated otherwise, we assign just one discovered cluster to each ground truth cluster.

We compare two variants of our algorithm: RASCL sets $k > K$ and RASCL_R sets $k = K$ which essentially skips the clustering step (i.e., we randomly pick k points and allocate the remaining points to the nearest point). For each dataset we use the ground truth and select the ground truth cluster with the largest overlap (highest $\text{PRECISION}_\text{SC}$) with the cluster being evaluated to compute its quality. We run each experiment 10 times and report the average results for the first r subspace clusters. Note that based on the overlap parameter, less than r clusters may be reported. Finally, unless stated otherwise, we fix the following parameters: $n = 1000$, $k = 100$, $K = 20$, $\sigma = 200$, $\mu = 100$ and $r = 10$. As a general rule, we provide the following guidance: n should be set high enough to obtain a representative sample, k should be sufficiently larger than K for the clustering to make sense, r should be set to the desired number of discovered clusters, μ should be high enough so no information is lost due to randomisation. K and σ are more difficult to set, but we show that the performance of RASCL is not overly sensitive to changes in their values (as long as σ is low enough to find some itemsets, otherwise no clusters at all can be found).

4.1 Cluster quality

We first compare our methods to CARTICLUS [5] and PROCLUS [11]. We used different setups of RASCL and RASCL_R by varying K and σ (indicated by the superscript). For CARTICLUS we use the parameter settings as selected by the authors of the original paper [5] as basis for this experiment. For PROCLUS we set parameters following the ground truth.

The object quality results in terms of $\text{PRECISION}_\text{OBJ}$ and RECALL_OBJ are shown in Fig. 2. We can observe that all algorithms perform very well with respect to $\text{PRECISION}_\text{OBJ}$ except for PROCLUS. Moreover, using $K = 20$ and $\sigma = 200$ our algorithms slightly outperform CARTICLUS. Turning to RECALL_OBJ, we see that $\text{RASCL}_\text{R}^{K10\sigma100}$, $\text{RASCL}^{K10\sigma100}$ and $\text{RASCL}^{K20\sigma200}$ outperform the competitors on RECALL_OBJ, while $\text{RASCL}_\text{R}^{K20\sigma200}$ often fails to deliver good results. This is due to the randomness introduced by our algorithm: using random centroids leads to fewer completely similar transactions and more partially similar transactions. Combined with a high support this results in small subclusters of the true ground truth clusters, leading to very high $\text{PRECISION}_\text{OBJ}$ with a low RECALL_OBJ.

The results for the dimension quality are shown in Fig. 3. We see that our algorithms generally outperform the competitors by quite a margin and we see that our simple solution of using linked dimensions (Sect. 3.2) works really well. This holds when the sampled cluster truly defines a subspace cluster. For smaller subspace clusters with lower $\text{PRECISION}_\text{OBJ}$, the dimension quality decreases as a result. Comparing ($K = 10$, $\sigma = 100$) to ($K = 20$, $\sigma = 200$), the latter produces better results, mostly because there are more linked dimensions tied to the cover of the maximal itemset, substantially boosting RECALL_DIM.

4.2 Runtime experiment

We tested the time required to run various algorithms on each one of the synthetic datasets. Our aim here is not to show that our algorithm is faster than existing methods such as PROCLUS, but rather that our method can handle datasets with different, growing characteristics well.

We used the optimal parameter settings for all different methods tested and ran 10 trial runs to account for fluctuations in runtimes. The averaged timing results are shown in Fig. 4. In Fig. 5 we show the same runtimes for growing datasets characteristics but relative to the runtime of the baseline settings, i.e., for sizescale compared to the dataset with 1500 data points, for dimscale compared to the dataset with 5 dimensions and for noisescale compared to the dataset with 10 noise dimensions.

A first thing to notice is that PROCLUS runs very fast for each of these datasets. RASCL_R and RASCL behave very similarly in general, but RASCL_R requires significantly less time because it skips the clustering step to create cluster centroids. CARTICLUS runs fast for smaller sized datasets, but tends to blow up quickly once the characteristics of the data increase.

In addition to the full runtimes, we also show the breakdown into smaller steps within the RASCL and RASCL_R algorithms in Tables 2 and 3, respectively. First, the reading of the data is a small cost. The generation of samples is fast and we see no impact of the increasing database size for the dbsizescale and noisescale datasets. For the dimscale datasets we see the time increasing because the number of dimensions is increasing and we compute the sampling potential using a density based approach in Eq. 1. For RASCL the generation of transactions depends largely on the number of records in the dataset as we use K-means to select centroids. In contrast, we do not use this step in RASCL_R, but rather use voronoi allocation using randomly selected centroids, which decreases the runtimes of the generation of transactions a lot. Sampling of clusters is tied to the number of items in the transaction dataset, which ultimately depends on the number of data objects in the original dataset. Therefore, we see the time

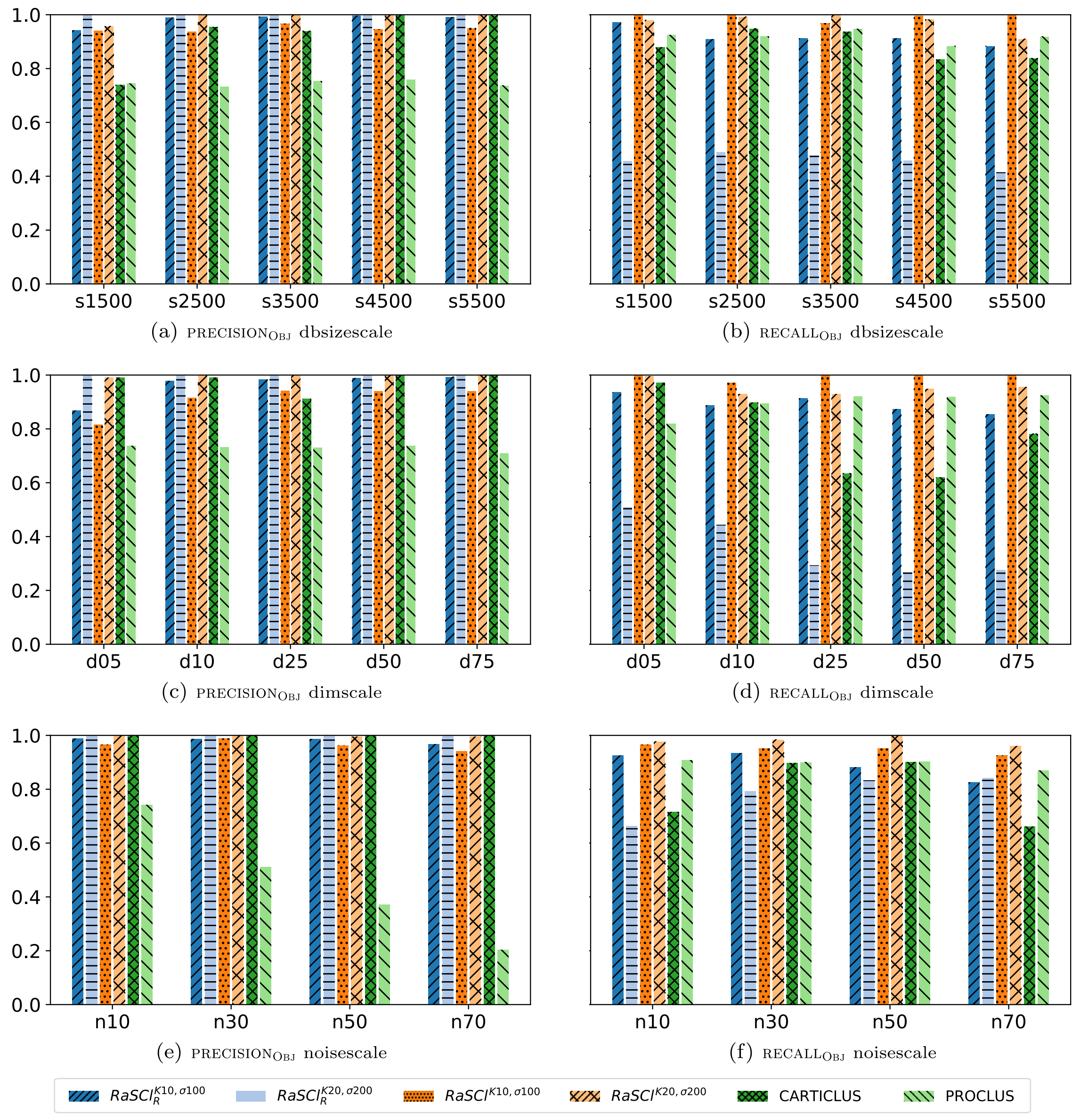


Fig. 2 Object quality scores for synthetic datasets grouped by the type of datasets. The graphs on the left show the precision scores on object level, the graphs on the right show the recall scores on object level. For RASCL and RASCL$_R$ we use default parameters and fixed parameters indicated in superscript. For CARTICLUS we used the settings from the original paper and for PROCLUS parameters are set following the ground truth

increase with the size of the data for dbsizescale and noisescale. In addition, comparing RASCL and RASCL$_R$, we see that the former runs longer, which can be assigned to the fact that it finds longer/better itemsets because the formed clusters are inherently better than in the fully randomised approach. Finally, the post-processing (final cluster selection) is a minor task in the algorithms.

4.3 Sorting strategy

In Sect. 3.4, we introduced a heuristic for sorting the sampled clusters. To evaluate our heuristic, we sampled a collection of clusters using default parameters and setting $\mu = 10000$ for RASCL. The results are shown in Fig. 6: the upper chart shows a histogram of the length of sampled clusters, the lower chart shows the relationship between the object length and the F1$_{SC}$-score for dbsizescale$_{s5500}$. (Other datasets produce similar figures.) The colours indicate the cluster to

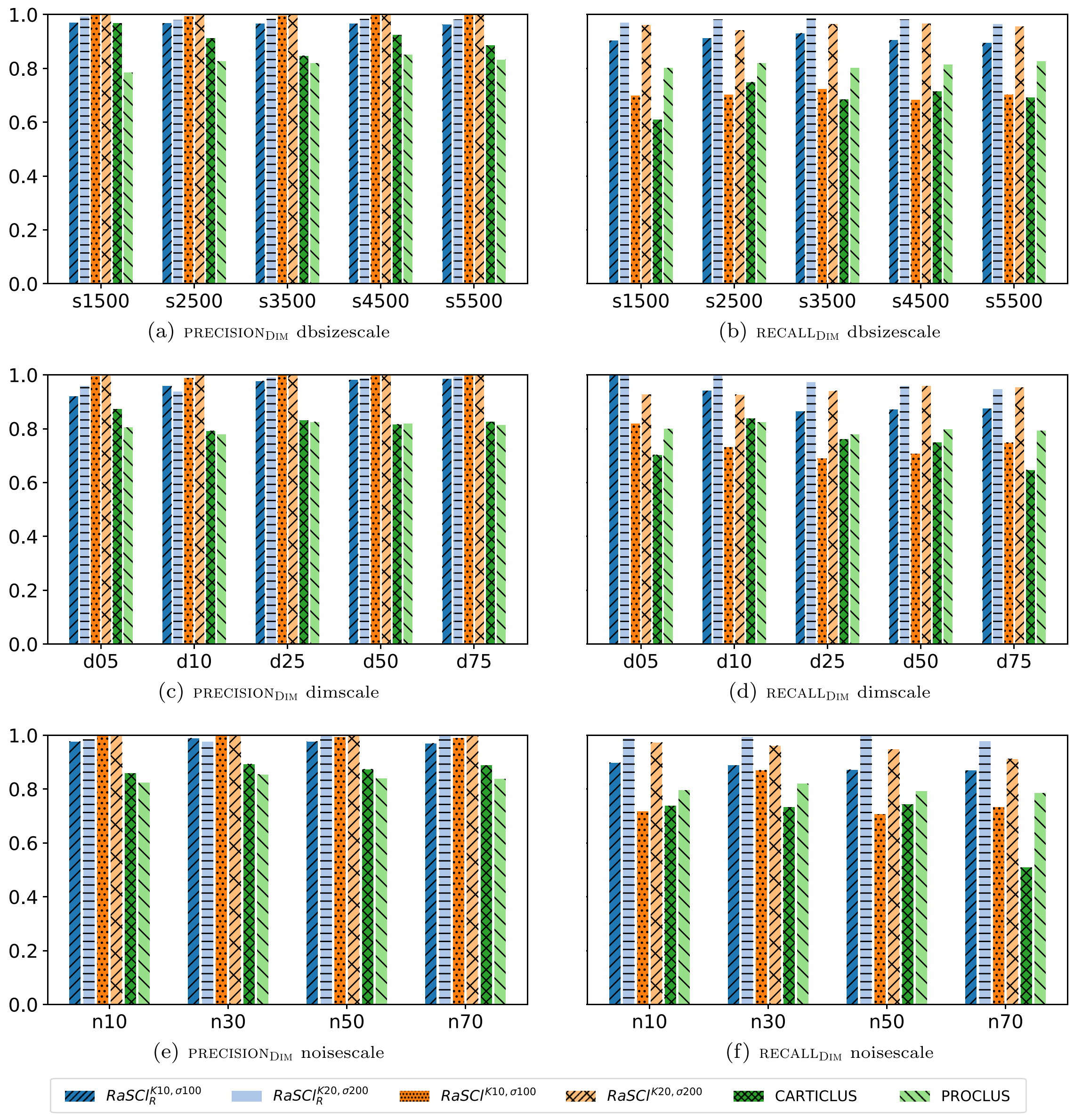


Fig. 3 Dimension quality scores for synthetic datasets grouped by the type of datasets. The graphs on the left show the precision scores on dimension level, the graphs on the right show the recall scores on dimension level. For RASCL and RASCL$_R$ we use default parameters and fixed parameters indicated in superscript. For CARTICLUS we used the settings from the original paper and for PROCLUS parameters are set following the ground truth

which the subspace cluster is assigned when computing F1$_{SC}$. To the left we show the results using all sampled clusters and to the right we show the results when retaining only non-overlapping cluster allocations. Our algorithm mostly produces large clusters, i.e., our randomised process generates many highly similar transactions, resulting in large maximal itemsets with high F1$_{SC}$-scores: good object quality results in good dimension quality. An interesting observation is the nod at object size 572 in Fig. 6a, which is an artefact of the data creation process including overlapping clusters: subclusters of overlapping clusters are assigned to the wrong ground truth cluster resulting in lower cluster scores. More precisely, the data contains a cluster C_1 of object size 572 that is fully contained in another cluster C_2 of size 1101, but with a slightly modified dimension set. What we observe for sampled clusters with an object size larger than 572 is that our algorithm finds a subset of objects that is a subset of C_2. However, our dimension strategy finds the dimension set for cluster C_1, because there is not enough evidence that some dimensions are irrelevant for a C_2 subspace cluster. Conse-

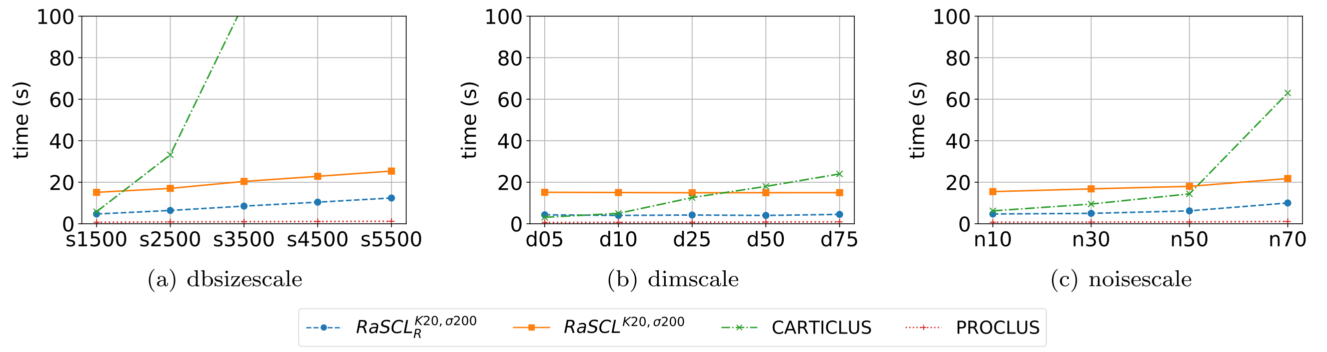


Fig. 4 Timing experiment showing the runtimes of various algorithms on synthetic datasets

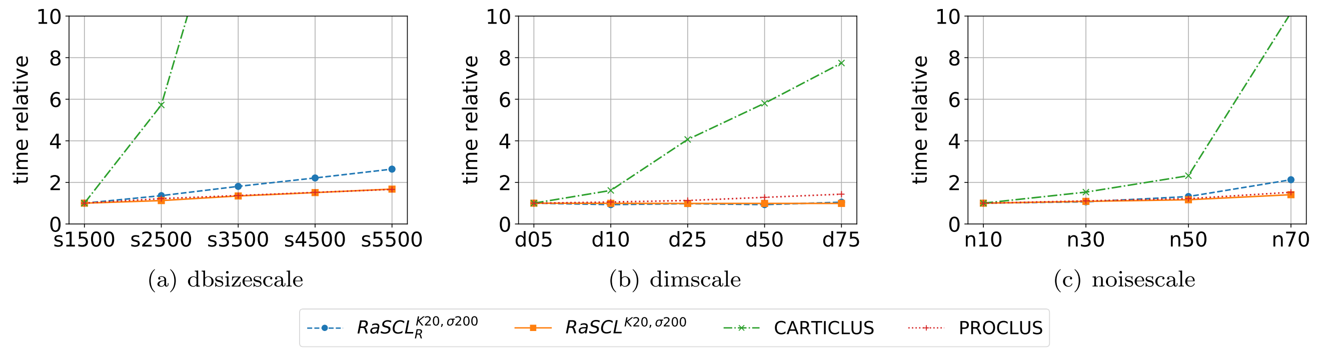


Fig. 5 Relative timing experiment showing the runtimes on synthetic datasets. Timings are relative to the runtime of the smallest dataset of that group. We show how the algorithms behave with the increasing dataset certain characteristics

quently, the PRECISION$_{\text{OBJ}}$ of the cluster goes down, which is reflected in the F1$_{\text{SC}}$-score. Different quality scores show similar effects. Indeed, retaining only non-overlapping cluster allocations, our algorithm mostly produces full clusters. Note that the low score for the orange cluster is due to randomisation in the transaction dataset creation and other runs could produce much better results for this cluster (see also Fig. 6d).

4.4 Parameter sensitivity

We test the influence of our most important parameters K and σ on the dimscale$_{\text{d05}}$, dimscale$_{\text{d25}}$ and dimscale$_{\text{d75}}$ datasets

Table 2 Detailed runtime breakdown for RASCL$^{K20,\sigma200}$

	Read data	Generate samples	Generate transactions	Sample clusters	Post-process	Total time
dbsizescale$_{\text{s1500}}$	0.02	0.19	11.71	2.89	0.02	14.83
dbsizescale$_{\text{s2500}}$	0.02	0.18	12.93	4.03	0.03	17.19
dbsizescale$_{\text{s3500}}$	0.02	0.17	14.30	5.82	0.02	20.33
dbsizescale$_{\text{s4500}}$	0.03	0.18	15.47	7.32	0.03	23.02
dbsizescale$_{\text{s5500}}$	0.03	0.17	16.42	8.39	0.02	25.04
dimscale$_{\text{d05}}$	0.01	0.17	12.47	2.80	0.01	15.46
dimscale$_{\text{d10}}$	0.04	0.17	11.79	2.70	0.01	14.71
dimscale$_{\text{d25}}$	0.02	0.20	11.51	2.93	0.02	14.68
dimscale$_{\text{d50}}$	0.03	0.22	11.90	2.91	0.02	15.08
dimscale$_{\text{d75}}$	0.04	0.25	11.63	2.83	0.02	14.76
noisescale$_{\text{n10}}$	0.02	0.19	12.40	2.80	0.03	15.43
noisescale$_{\text{n30}}$	0.02	0.19	11.71	2.97	0.03	14.92
noisescale$_{\text{n50}}$	0.02	0.17	13.79	3.58	0.02	17.58
noisescale$_{\text{n70}}$	0.03	0.19	15.75	5.02	0.02	21.01

Table 3 Detailed runtime breakdown for $\text{RASCL}_{R}^{K20,\sigma 200}$

	Read data	Generate samples	Generate transactions	Sample clusters	Post-process	Total time
dbsizescale$_{s1500}$	0.02	0.15	2.40	2.36	0.01	4.94
dbsizescale$_{s2500}$	0.02	0.13	3.10	3.52	0.02	6.79
dbsizescale$_{s3500}$	0.02	0.12	3.91	4.79	0.01	8.85
dbsizescale$_{s4500}$	0.03	0.12	4.69	6.04	0.02	10.90
dbsizescale$_{s5500}$	0.03	0.12	5.61	7.15	0.02	12.93
dimscale$_{d05}$	0.01	0.12	2.33	2.28	0.01	4.76
dimscale$_{d10}$	0.02	0.13	2.32	2.25	0.01	4.73
dimscale$_{d25}$	0.02	0.16	2.35	2.21	0.01	4.75
dimscale$_{d50}$	0.03	0.19	2.35	2.28	0.02	4.86
dimscale$_{d75}$	0.04	0.21	2.30	2.18	0.02	4.75
noisescale$_{n10}$	0.02	0.13	2.30	2.51	0.01	4.97
noisescale$_{n30}$	0.02	0.13	2.67	2.69	0.03	5.54
noisescale$_{n50}$	0.02	0.12	3.38	3.36	0.02	6.89
noisescale$_{n70}$	0.03	0.12	5.03	5.17	0.02	10.38

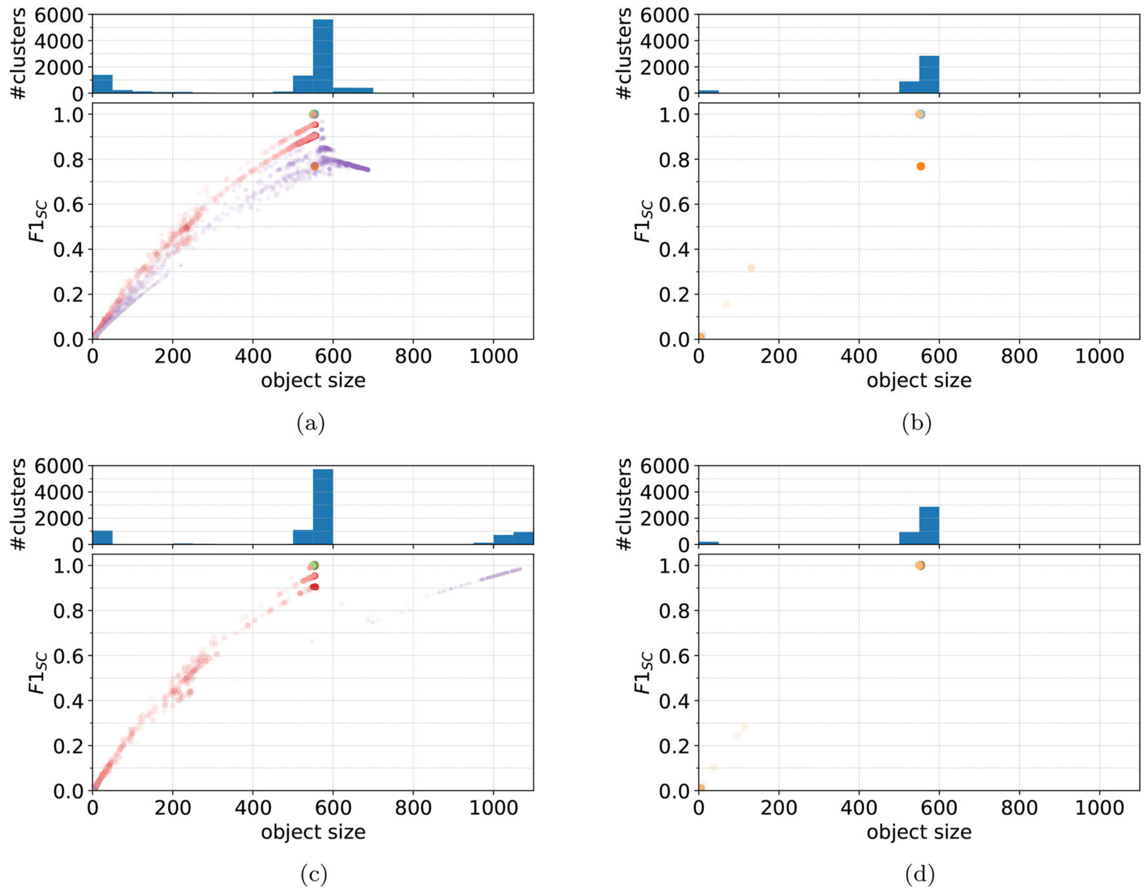


Fig. 6 The relation between the object size and the $F1_{SC}$ quality score for dbsizescale$_{s5500}$. The top charts show histograms of the length of sampled clusters. The lower charts show the object size in comparison to the $F1_{SC}$ quality score. Figure **a** and **c** show all discovered subspace clusters for 2 different runs, Fig. **b** and **d** show non-overlapping clusters for the same runs

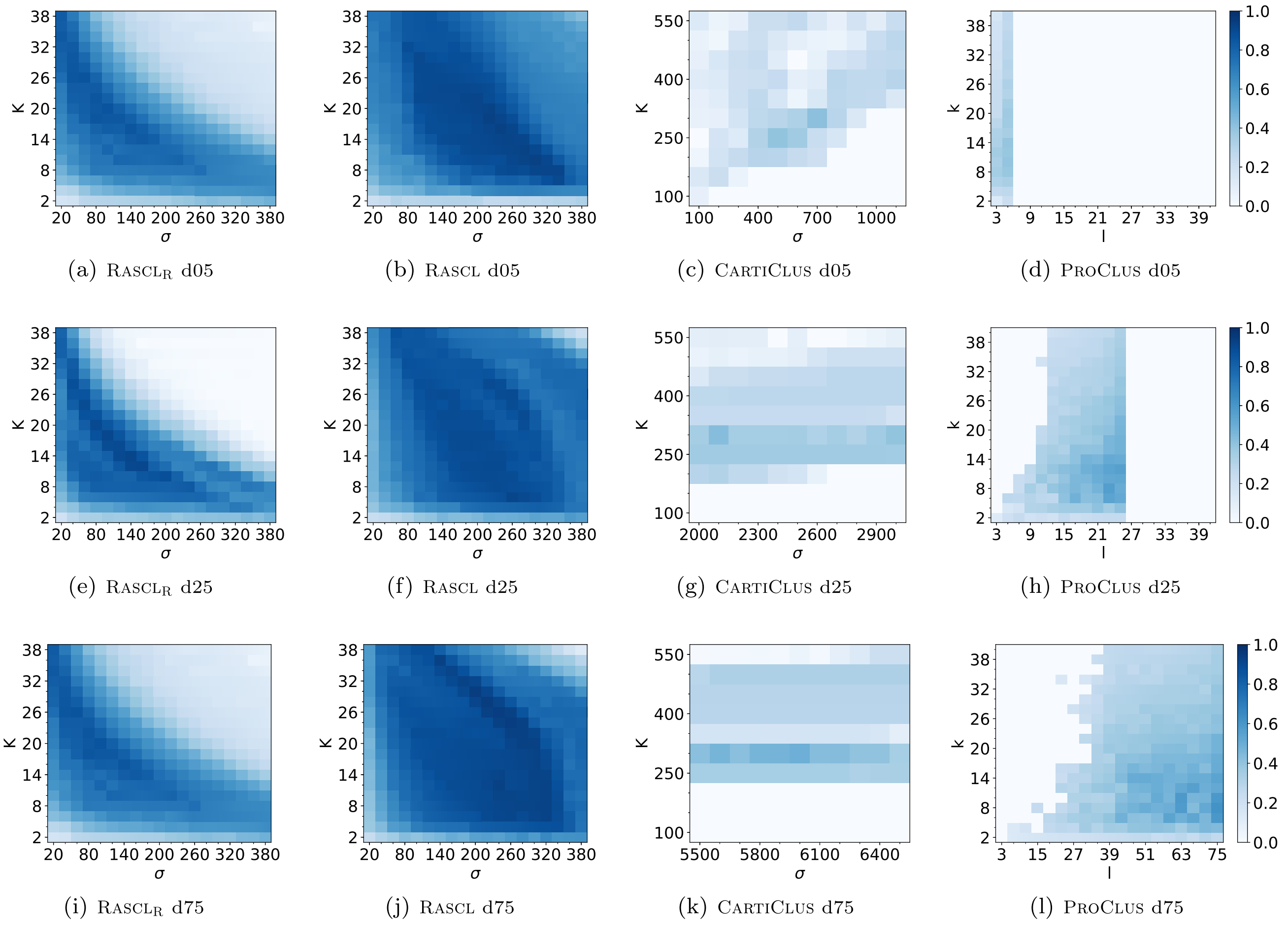


Fig. 7 Parameter grid search quality results for various methods on several dimscale datasets

and show the M^{E4SC} scores. Note that even though our algorithm is not meant for producing full clusterings, it produces very high-quality results on this metric. Figure 7 shows the scores for the first 10 subspace clusters using various parameter settings for the algorithms. For our algorithm we use a window around the default parameters. For CARTICLUS we use a grid around the optimal parameters and for PROCLUS we define sensible grids. We see that RASCL is not overly susceptible to parameter changes and that, in general, the default parameters produce good and stable results. In contrast, RASCL$_R$ can still produce very good results, but the quality diminishes quickly when the parameters are not too far from the optimal parameters. Increasing K or σ results in subclusters of the true clusters, thus decreasing the overall score. We see that finding good settings for CARTICLUS and PROCLUS is much harder. For PROCLUS l cannot exceed the number of dimensions in the data, resulting in lots of 0 scores in the figures. The experiments on other datasets produced similar results.

We also show the impact of varying k and n, whilst keeping the other parameters fixed at the default settings ($K = 20$, $\sigma = 200$ and $\mu = 100$). For each setting, we create 10 sets of clusters and compute the M^{E4SC} score. We then average the quality scores for each parameter setting and plot the resulting scores using a boxplot. The results on synthetic datasets are shown in Fig. 8. To show the stabilisation of the quality score we show three different setups. First, we varied k starting from 800 up to 2000 (incl.) with steps of 100, while varying n between 80 and 400 (incl.) with steps of 20 in Fig. 8a. Second, we summarise the scores when k varies between 800 and 1200 (incl.) and n varies between 80 and 120 (incl.) in Fig. 8b. Finally, we also show the quality when k is fixed to 1000 and n is fixed to 100 in Fig. 8c. These results show the stability of the scores while varying the two parameters. In addition, this shows how the fixed setting ($k = 1000$, $n = 100$) consistently achieves one of the highest scores for each dataset.

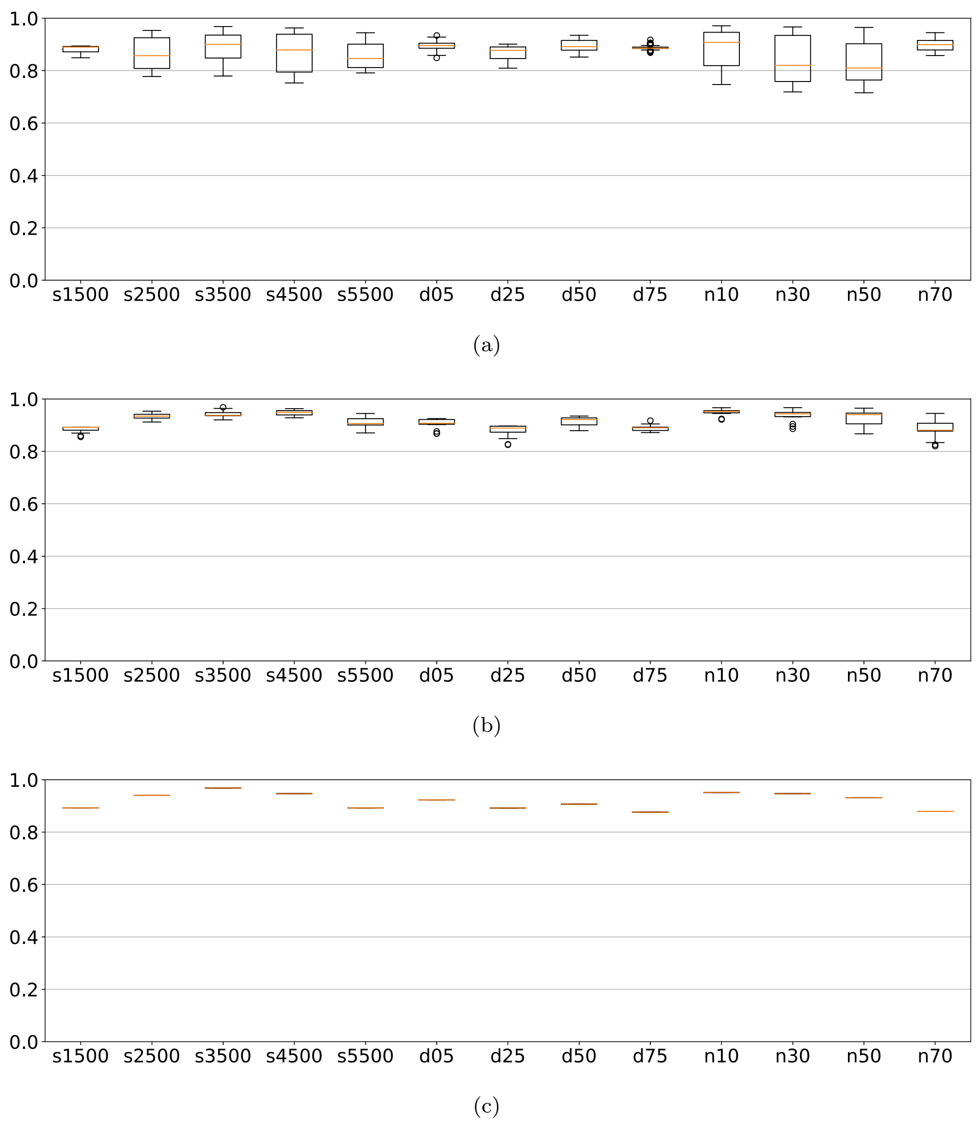


Fig. 8 Boxplots of tenfold averaged M^{E4SC} scores for RASCL on various datasets with fixed parameters $K = 20$, $\sigma = 200$ and $\mu = 100$ and varying parameters for k and n. In **a** k is varied between 60 and 400 (incl.) with increments of 20 and n is varied between 600 and 2000 with increments of 100. In **b** k is varied between 80 and 120 (incl.) with increments of 20 and n is varied between 800 and 1200 (incl.) with increments of 100. In **c** k is fixed to 1000 and n is fixed to 100

4.5 Real-world data

We tested our method on the pendigits dataset, a classification dataset found in the UCI machine learning repository[2]. Using RASCL with $n = 1000$, $k = 100$, $K = 10$, $\sigma = 100$, $\mu = 100$ and $r = 10$ we discover multiple subspace clusters for each class. A general trend we found was that the discovered clusters have a very high PRECISION$_{\text{OBJ}}$ of approx. 91%, but they have rather low RECALL$_{\text{OBJ}}$ averaging around 20%. We evaluate the largest subspace cluster in Fig. 9. The silhouette plot shows high similarity for points in the cluster (red) and a much lower score for points outside the cluster (blue). A similar trend is found in the scatter plots, which are obtained using *t-SNE* transformation [14] based on the relevant dimensions. The left scatter plot shows all data (blue) together with the subspace cluster (red), while the right scatter plot shows only data points not in the cluster. This shows that using our method we do not miss many data points that are within the region of the subspace cluster according to this transformation. The plot on the right is the Andrews plot [15], which can be seen as a smoothed parallel coordinates plot, showing cluster structures more clearly. It shows that the data points in the cluster are closely related in all dimensions. Similar plots are found for the remaining clusters.

5 Related work

Traditional clustering methods are known to be struggling in high-dimensional space. Clustering techniques that measure distances between data objects over all dimensions (i.e., the full data space) are hampered by irrelevant dimensions and produce meaningless results. There exist two

[2] https://archive.ics.uci.edu/ml/datasets/Pen-Based+Recognition+of+Handwritten+Digits

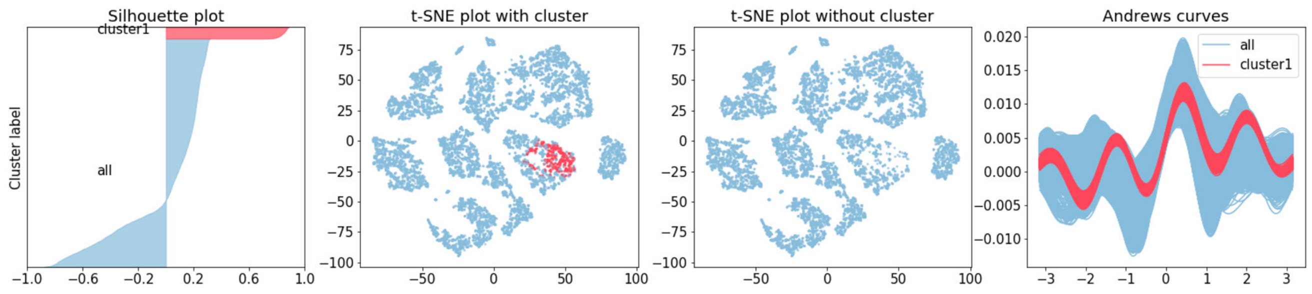


Fig. 9 Subspace cluster 1 for the pendigits dataset using RASCL

well-known techniques for projecting a high-dimensional database to lower-dimensional projections—namely, dimensionality reduction [16] and unsupervised feature selection [17]. However, such projections, inevitably, suffer from information loss. In addition, these methods produce single views of the data, allowing for cluster detection only within a single projection, while other projections, that may yet reveal interesting clusters, are left unexplored.

Subspace clustering, on the other hand, attempts to find clusters in subsets of dimensions. However, some traditional clustering models, unsuited to this setting, have been adapted for this purpose. For example, the well-known K-means algorithm [18] has been adapted for subspace clustering. The resulting PROCLUS [11] algorithm was one of the first methods for finding projected clusters and clusterings. The analogy to our work is the initialisation phase, where a two-step randomised procedure is used to obtain an approximation to a *piercing set*, i.e., a set of points each from a different cluster, which are refined to full clusters.

DOC [19] is an algorithm that finds subspace clusters using a Monte Carlo method to sample a random point from a cluster as well as a *discriminating* set of points. It then extends the random point to a full subspace cluster using a bounding box around that point. Finally, it selects the subspace cluster with the highest quality. The latter is a trade-off between the number of data points and the number of dimensions. Its extension MINECLUS [20] uses the same medoid points for expanding the cluster, but it drops the randomised procedure. Similar to our approach, it also converts the data to a binarised dataset. As such, finding the best projected cluster for random medoid p is transformed to the problem of finding the best itemset given a goodness function μ. Other clustering algorithms, such as DBSCAN [21], have also been adapted for the subspace clustering task [22]. Since then, more general techniques have been proposed for searching the subspace [23,24], where the discovery of clusters is left to specialised algorithms, followed by further efforts to reduce the time complexity of such methods [25]. However, as mentioned in Sect. 1, all of the above methods are computationally very expensive as they search in an exponential set of subspaces.

FIRES [10] is a generic framework for finding subspace clusters, employing existing clustering techniques to compute a set of base clusters in single dimensions. These base clusters are then merged based on their similarity, and the resulting clusters are then pruned and refined to optimise accuracy. The CARTICLUS algorithm [5], like our method, creates a binarised dataset. However, in CARTICLUS, the dimensions are defined during the construction of transactions (or *carts*), such that all carts rely on the same dimension sets. Finally, the carts are mined for frequent itemsets which are then translated back to subspace clusters. Bi-clustering [26] also simultaneously clusters rows and columns of numeric matrices. However, bi-clusters allow for more general clusters as they, for instance, group rows with constant values for a set of columns or group columns that decrease similarly over a set of rows. Typically, such methods are used for analysis of biological data such as gene expression data.

Recently, as in many other fields, there has been increased usage of deep learning methods for discovering subspace clusters. Examples of deep learning subspace clustering algorithms include StructAE, which uses deep neural networks [27], SSC, a graph structured autoencoder [28], and SDEC that performs semi-supervised deep embedded clustering [29]. Kelkar et al. provide a more complete overview of the wider subspace clustering field in a recent survey [30].

Our method addresses important problems that existing methods suffer from. First of all, by binarising the data in a random fashion, we can efficiently explore many different subspaces, of different dimensions, in one go. We are therefore neither limited to a single projection nor, like CARTICLUS, to fixed sets of dimensions defined during the binarisation step. This flexibility increases the likelihood of quickly discovering the correct set of dimensions for each cluster. Moreover, by deploying a randomised itemset search, we also avoid the computational complexity of traditional frequent itemset mining algorithms [31]. Finally, unlike the deep learning algorithms, our method is entirely transparent in terms of how points are allocated to clusters. If a point belongs to a cluster, it is fairly straightforward to explain why, knowing how our algorithm works. Concretely, points form a cluster together because they have repeatedly been found to be similar to each other in a particular set of dimensions during our random sampling procedure used for building a binarised transaction database.

6 Conclusion

In this paper, we present a novel method for discovering interesting clusters in high-dimensional data. In the first stage of our technique, we start by converting the original data into a transaction database by selecting a relatively small number of random data objects, projecting them to a small number of random dimensions, then clustering those objects, and, finally, building transactions by assigning all data objects to their closest cluster centroids. We repeat the above procedure a large number of times and merge all the resulting transactions into a single large transaction database. The main idea behind this approach is that objects that form subspace clusters will appear together in many transactions. In the second stage, we sample maximal itemsets randomly from the transaction database, and consider each such itemset to be a potentially interesting cluster of objects. Finally, in the third stage, for each discovered cluster, we identify a relevant set of dimensions, thus defining a subspace cluster.

A major advantage of our method is that, by using the two randomised procedures, we avoid both the combinatorial explosion of possible dimension sets, and the computational cost of frequent itemset mining. In addition, we do not attempt to produce full clusterings, and we allow data objects to be part of multiple clusters (typically in different sets of dimensions), while noise objects will not be part of any cluster at all. Experimentally, we evaluate two variants of our algorithm and compare our technique to existing state-of-the-art methods. We demonstrate that our method produces quality clusters and is not overly sensitive to changes in the parameter settings, which is crucial for an unsupervised learning task where getting the parameters right can be very challenging.

In Sect. 3.3, we described how the first stage of our method can be generalised to a clustering step followed by a classification step, and even how the clustering step can be skipped altogether. In our implementation, we have chosen the K-means algorithm for the clustering step, and the 1-nearest neighbour algorithm for the classification step. In future work, we intend to analyse how combinations of different clustering and classification algorithms would perform in this setting. In particular, we are interested in examining if investing additional computational effort for these two tasks would ultimately lead to better overall results, or is keeping it simple (as we do now) sufficient to obtain quality output.

Funding No funding was received to assist with the preparation of this manuscript.

Declarations

Conflict of interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. Jain, A.K., Murty, M.N., Flynn, P.J.: Data clustering: a review. ACM Comput. Surv. (CSUR) **31**(3), 264–323 (1999)
2. Bellman, R.E.: Adaptive Control Processes: A Guided Tour. Princeton University Press, Princeton (2015)
3. Parsons, L., Haque, E., Liu, H.: Subspace clustering for high dimensional data: a review. ACM SIGKDD Explor. Newslett. **6**(1), 90–105 (2004)
4. Moise, G., Sander, J., Ester, M.: P3c: A robust projected clustering algorithm. In *Sixth international conference on data mining (ICDM'06)*. IEEE, 2006, pp. 414–425
5. Aksehirli, E., Goethals, B., Muller, E., Vreeken, J.: Cartification: a neighborhood preserving transformation for mining high dimensional data, In *2013 IEEE 13th international conference on data mining (ICDM)*, IEEE, (2013), pp. 937–942
6. Moens, S., Cule, B., Goethals, B.: A sampling-based approach for discovering subspace clusters. In: Discovery science - 22nd international conference, DS. Springer **2019**, 61–71 (2019)
7. Patrikainen, A., Meila, M.: Comparing subspace clusterings. IEEE Trans. Knowl. Data Eng. **18**(7), 902–916 (2006)
8. Günnemann, S., Färber, I., Müller, E., Assent, I., Seidl, T.: External evaluation measures for subspace clustering, In: *Proc of the 20th ACM international conference on information and knowledge management*. ACM, 2011, pp. 1363–1372
9. Agrawal, R., Gehrke, J., Gunopulos, D., Raghavan, P.: *Automatic subspace clustering of high dimensional data for data mining applications*. ACM, (1998), vol. 27, no. 2
10. Kriegel, H.-P., Kroger, P., Renz, M., Wurst, S.: A generic framework for efficient subspace clustering of high-dimensional data, In: *Fifth IEEE international conference on data mining (ICDM'05)*. IEEE, 2005, pp. 8–pp
11. Aggarwal, C.C., Wolf, J.L., Yu, P.S., Procopiuc, C., Park: J.S.: Fast algorithms for projected clustering, In: *ACM SIGMoD Record*, vol. 28, no. 2. ACM, (1999), pp. 61–72
12. Freedman, D., Diaconis, P.: On the histogram as a density estimator: L 2 theory. Probab. Theory Relat. Fields **57**(4), 453–476 (1981)
13. Moens, S., Goethals, B.: Randomly sampling maximal itemsets. In: *KDD workshop on interactive data exploration and analytics*. ACM, 2013, pp. 79–86
14. Maaten, L.V.D., Hinton, G.: Visualizing data using t-sne. J. Mach. Learn. Res. **9**, 2579–2605 (2008)
15. Andrews, D.F.: Plots of high-dimensional data, *Biometrics*, pp. 125–136, (1972)

16. Lee, J.A., Verleysen, M.: Nonlinear Dimensionality Reduction. Springer, Berlin (2007)
17. Dy, J.G., Brodley, C.E.: Feature selection for unsupervised learning. J. Mach. Learn. Res. **5**, 845–889 (2004)
18. MacQueen, J., et al.: Some methods for classification and analysis of multivariate observations, In: *proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, no. 14. Oakland, CA, USA, 1967, pp. 281–297
19. Procopiuc, C.M., Jones, M., Agarwal, P.K., Murali, T.: A monte carlo algorithm for fast projective clustering, In: *proceedings of the 2002 ACM SIGMOD international conference on management of data*. ACM, 2002, pp. 418–427
20. Yiu, M.L., Mamoulis, N., Frequent-pattern based iterative projected clustering. In: *Third IEEE international conference on data mining, ICDM 2003*, IEEE, pp. 689–692 (2003)
21. Ester, M., Kriegel, H.-P., Sander, J., Xu, X., et al.: A density-based algorithm for discovering clusters in large spatial databases with noise. Kdd **96**(34), 226–231 (1996)
22. Kailing, K., Kriegel, H.-P., Kröger, P.: Density-connected subspace clustering for high-dimensional data, In: *proceedings of the 2004 SIAM international conference on data mining*. SIAM, 2004, pp. 246–256
23. Cheng, C.-H., Fu, A.W., Zhang, Y.: Entropy-based subspace clustering for mining numerical data, In: *proceedings of the fifth ACM SIGKDD international conference on knowledge discovery and data mining*. ACM, (1999), pp. 84–93
24. Nguyen, H.V., Müller, E., Vreeken, J., Keller, F., Böhm, K.: Cmi: An information-theoretic contrast measure for enhancing subspace cluster and outlier detection, In: *SIAM international conference on data mining*. SIAM, (2013), pp. 198–206
25. Pourkamali-Anaraki, F., Folberth, J., Becker, S.: Efficient solvers for sparse subspace clustering. Signal Process. **172**, 107548 (2020)
26. Madeira, S.C., Oliveira, A.L.: Biclustering algorithms for biological data analysis: a survey. IEEE/ACM Trans. Comput. Biol. Bioinf. (TCBB) **1**(1), 24–45 (2004)
27. Peng, X., Feng, J., Xiao, S., Yau, W.-Y., Zhou, J.T., Yang, S.: Structured autoencoders for subspace clustering. IEEE Trans. Image Process. **27**(10), 5076–5086 (2018)
28. Majumdar, A.: Graph structured autoencoder. Neural Netw. **106**, 271–280 (2018)
29. Ren, Y., Hu, K., Dai, X., Pan, L., Hoi, S.C., Xu, Z.: Semi-supervised deep embedded clustering. Neurocomputing **325**, 121–130 (2019)
30. Kelkar, B.A., Rodd, S.F.: Subspace clustering'a survey, In: *Data Management, Analytics and Innovation*. Springer, (2019), pp. 209–220
31. Agrawal, R., Srikant, R., *et al.*: Fast algorithms for mining association rules. In: *proceedings 20th international conference very large data bases, VLDB*, vol. 1215, (1994), pp. 487–499

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.