Kaur and Datta *Journal of Big Data* (2015) 2:17
DOI 10.1186/s40537-015-0027-y

Journal of Big Data
a SpringerOpen Journal

RESEARCH Open Access

A novel algorithm for fast and scalable subspace clustering of high-dimensional data

Amardeep Kaur* and Amitava Datta

*Correspondence:
amardeep.kaur@research.uwa.edu.au
School of Computer Science and Software Engineering, University of Western Australia, Stirling Highway, 6009, Perth, Australia

Abstract

Rapid growth of high dimensional datasets in recent years has created an emergent need to extract the knowledge underlying them. Clustering is the process of automatically finding groups of similar data points in the space of the dimensions or attributes of a dataset. Finding clusters in the high dimensional datasets is an important and challenging data mining problem. Data group together differently under different subsets of dimensions, called subspaces. Quite often a dataset can be better understood by clustering it in its subspaces, a process called subspace clustering. But the exponential growth in the number of these subspaces with the dimensionality of data makes the whole process of subspace clustering computationally very expensive. There is a growing demand for efficient and scalable subspace clustering solutions in many Big data application domains like biology, computer vision, astronomy and social networking. Apriori based hierarchical clustering is a promising approach to find all possible higher dimensional subspace clusters from the lower dimensional clusters using a bottom-up process. However, the performance of the existing algorithms based on this approach deteriorates drastically with the increase in the number of dimensions. Most of these algorithms require multiple database scans and generate a large number of redundant subspace clusters, either implicitly or explicitly, during the clustering process. In this paper, we present SUBSCALE, a novel clustering algorithm to find non-trivial subspace clusters with minimal cost and it requires only k database scans for a k-dimensional data set. Our algorithm scales very well with the dimensionality of the dataset and is highly parallelizable. We present the details of the SUBSCALE algorithm and its evaluation in this paper.

Keywords: Data mining; High dimensional data; Subspace clustering; Scalable data mining

Introduction

With recent advancements in information technology, voluminous data are being captured in almost every conceivable area, ranging from astronomy to biological sciences. Thousands of microarray data repositories have been created for gene expression investigation [1]; sophisticated cameras are becoming ubiquitous, generating a huge amount of visual data for surveillance; the Square Kilometre Array Telescope is being built for astrophysics research and is expected to generate several petabytes of astronoimcal data every year [2]. All of these datasets (also called Big data) have a large number of dimensions (attributes) and pose significant research challenges for the data mining community [3, 4].

Each data point is a vector of measurements in a multidimensional space. Clustering is an unsupervised data mining process of grouping similar data points into clusters (dense regions) without any prior knowledge of the underlying data distribution. A variety of distance measures can be used to quantify the similarity or density of the data points. For the last two decades, there has been significant research work in cluster analysis of the data [5–7]. Traditional clustering algorithms were designed to generate clusters in the full-dimensional space by measuring the proximity between the data points using all of the dimensions of a dataset [8, 9]. But as the dimensionality of data increases, some of the dimensions become irrelevant to some of the clusters. These classical clustering algorithms are ineffective as well as inefficient for the high-dimensional Big data.

The high-dimensional data suffers from the *Curse of Dimensionality* [10] which has two main implications: (1) As the dimensionality of data grows, the relative contrast among similar and dissimilar points becomes less [11]. (2) Data tend to group together differently under the different sets of dimensions [12]. Therefore, it becomes imperative to find the clusters in the relevant subsets of dimensions of the data (called *subspaces*). The subspace clustering is the process of finding clusters in the subspaces of the dataset. In this paper, we focus our interest on finding all the clusters in all possible subspaces. The subspace clustering is not 'just clustering' because we do not have any advance knowledge about the number of subspaces which contain clusters, the dimensionality of these subspaces and the number of clusters hidden in each subspace. As a k-dimensional data can have upto $2^k - 1$ possible axis-parallel subspaces, the exponential number of these subspaces poses unique computational challenges for mining Big data with large number of dimensions.

Motivating examples

The area of subspace clustering is of critical importance with enormous applications for Big data mining:

- In biology, high throughput gene expression data obtained from microarray chips forms a matrix [13]. Each cell in this matrix contains the expression level of a gene (row) under an experimental condition (column). The genes which co-express together under the subsets of experimental conditions are likely to be functionally similar [14]. One of the interesting characteristics of this data is that both genes (rows) and experimental conditions (columns) can be clustered for meaningful biological inferences. One of the assumptions in molecular biology is that only a subset of genes are expressed under a subset of experimental conditions, for a particular cellular process [15]. Also, a gene or an experimental condition can participate in more than one cellular processes allowing the existence of overlapping gene-clusters in different subspaces. Cheng and Church [16] were the first to introduce biclustering which is just an extension of subspace clustering, for microarray datasets. Since then many subspace clustering algorithms have been designed for the understanding of cellular processes [17] and gene regulatory networks [18], assisting in disease diagnosis [19] and thus, better medical treatments. Eren et al. [20] have recently compared the performance of related subspace clustering algorithms in microarray data and because of the combinatorial nature of the solution space, subspace clustering is still a challenge in this domain.

- Many computer vision problems are associated with matching images, scenes, or motion dynamics in video sequences. Image data is very high dimensional, e.g, a low-end 3.1 megapixel camera can capture an 2048×1536 image of 3145728 dimensions. It has been shown that the solutions to these high-dimensional computer vision problems lie in finding the structures of interest in the lower-dimensional subspaces [21–23]. As a result subspace clustering is very important in many computer vision and image processing problems, e.g, recognition of faces and moving objects. Face recognition is a challenging area as the images of the same object may look entirely different under different illumination conditions and different images can look same under different illumination settings. However, Basri et al. [21] have proved that all possible illumination conditions can be well approximated by a 9-dimensional linear subspace, which has further directed the use of subspace clustering in this area [24, 25]. Motion segmentation involves segregating each of the moving objects in a video sequence and is very important for robotics, video surveillance, action recognition etc. Assuming each moving object has its own trajectory in the video, the motion segmentation problem reduces to clustering the trajectories of each of the object [26], another subspace clustering problem.
- In online social networks, the detection of communities having similar interests can aid both sociologists and target marketers [26]. Günnemann et al. [27] have applied subspace clustering on social network graphs for community detection.
- In radio astronomy, clusters of galaxies can help cosmologists trace the mass distribution of the universe and further understand the origin of universe theories [28].
- Another important area of subspace clustering is web text mining through document clustering. There are billions of digital documents available today and each document is a collection of many words or phrases, making it a high-dimensional application domain. Document clustering is very important these days for the efficient indexing, storage and retrieval of digital content. Documents can group together differently under different sets of words. An iterative subspace clustering algorithm for text mining has been proposed by Li et al. [29].

In all of the applications discussed above, meaningful knowledge is hidden in the underlying subspaces of the data, which can only be explored through the subspace clustering techniques. The number of dimensions participating in a particular subspace S defines its cardinality denoted by $|S|$. If $|S_1| < |S_2|$, it means that S_1 is a lower dimensional or lower subspace and S_2 is a higher dimensional or higher subspace.

There are two main flavours of subspace clustering available in the literature: top down and bottom up. In top down approaches, the user defines the number of clusters and the number of relevant subspaces [30, 31]. It is not possible to automatically find all possible clusters in all the subspaces using this method.

The other paradigm for subspace clustering is the bottom-up hierarchical method based on the Apriori principle [32] and has the capability of reporting all hidden clusters in their relevant subspaces [33–35]. The Apriori principle is based on the downward closure property of the dense regions. According to Apriori principle, the projections of the close points in higher dimensional subspaces are also close to each other. This

knowledge helps to prune the non-dense neighbourhoods of the data points in the lower dimensional subspaces as they will never lead to the dense neighbourhoods in the higher dimensional subspaces. Thus, only the dense set of points (clusters) starting from the 1-dimensional subspaces, are chosen as candidates to be combined together iteratively for computing the higher dimensional clusters. As in Fig. 1, the 1-dimensional clusters from subspaces ($\{1\}, \{3\}$ and $\{4\}$) are combined to find the clusters in the 2-dimensional subspaces ($\{1,3\}, \{3,4\}$ and $\{1,4\}$) and then finally in the 3-dimensional space $\{1,3,4\}$. However, many combinations of the lower dimensional clusters might not result in the higher dimensional clusters during the step-by-step bottom-up clustering process. Although these algorithms look promising for finding all possible subspace clusters, they fail to scale beyond few tens of dimensions due to the exponential search space [5, 12, 36]. Also, the speed as well as the quality of clustering is of major concern for the high dimensional data clustering [4].

There are two main reasons for the inefficiency in the current bottom-up subspace clustering algorithms: detection of redundant trivial clusters and an excessive number of database scans. Revisiting Apriori principle, if C is a cluster in a d-dimensional space then it is also redundantly present in all of the $2^d - 1$ projections. And if this cluster C does not exists in any of the (d+1)-dimensional higher subspaces, then it is called a *maximal subspace cluster*. Ideally, the non-maximal clusters should not be generated because they are trivial but most of the algorithms implicitly or explicitly compute them. The second problem arises during the iterative bottom-up process of combining lower dimensional candidate clusters. Multiple database scans ($O(|\mathbf{DB}|^2)$) are required for determining the occupancy of each and every data point while merging these candidate clusters where, $|\mathbf{DB}|$ is the size of the dataset. Though sophisticated indexing structures have been used for these look-ups, still it remains one of the main inefficiencies in the subspace clustering algorithms so far.

In this paper, we present a novel subspace clustering algorithm that aims to remove both of these inefficiencies and has a high degree of parallelism. Our algorithm SUBSCALE

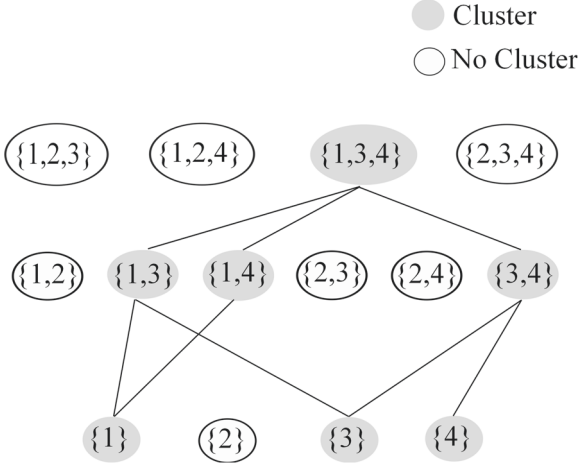

Fig. 1 Bottom-up clustering. Typical iterative bottom up generation of clusters based on the Apriori-principle. Dense points in 1-dimensional subspace are combined to compute two-dimensional clusters which are then combined to compute three dimensional clusters and so on

creates a possibility to walk through the exponential search space of the Big data and thus, directly compute the *maximal subspace clusters*. We propose a new idea of assigning signatures to the 1-dimensional clusters such that their collisions will help to identify the maximal subspace clusters without generating any intermediate trivial clusters. SUBSCALE algorithm requires only k database scans to process a k-dimensional dataset and is far more scalable with the dimensions as compared to the existing state-of-the-art algorithms. We have experimented with upto 6144 dimensional datasets and the results are promising.

This paper is an extended version of the work published in the ICDM 2014 workshop proceedings [37]. The SUBSCALE algorithm has been reworked to accommodate the scalability irrespective of the main memory constraint, which was still a bottleneck in the previously published version. In this paper, we have experimented with the higher dimensional datasets and analyzed in detail its performance as well as the factors influencing the quality of clusters. We also discuss various issues governing the optimal values of the input parameters and the flexibility available in the SUBSCALE algorithm to adapt these parameters accordingly.

This paper is organized as follows. The current literature related to our approach is discussed in the next Section Background and literature review. We discuss SUBSCALE algorithm in detail in the Section Research design and methodology and then in the Section Results and discussion, we analyse its performance. Finally, the paper is summarized in the Conclusions Section.

Background and literature review

The traditional clustering algorithms use the whole data space to find full-dimensional clusters. DBSCAN [9] is a well known full-dimensional clustering algorithm and according to it, a point is dense if it has τ or more points in its ϵ-neighborhood and a cluster is defined as a set of such dense points. However, the curse of dimensionality implies that the data loses its contrast in the higher-dimensional space [10, 11]. Therefore, these full-dimensional clustering algorithms are not able to detect any meaningful clusters with the increase in dimensionality of the data.

Another technique to deal with the high dimensionality is to reduce the number of dimensions by removing the irrelevant (or less relevant) dimensions, e.g. Principal Component Analysis (PCA) transforms the original high dimensional space into a low dimensional space [38]. Since PCA preserves the original variance of the full-dimensional data during this transformation, therefore, if no cluster structure was detected in the original dimensions, no new clusters in the transformed dimensions will be found. The transformed dimensions lack the intuitive meaning as it is difficult to interpret the clusters found in the new dimensions in relation to the original data space. Also, dimensionality reduction is not always possible [33].

The significance of the local relevance among the data with respect to the subset of dimensions, has lead to the advent of subspace clustering algorithms [12, 39]. We have identified the following desirable properties which should be satisfied by a subspace clustering algorithm for a k-dimensional dataset of n points:

1. Since different points group together differently with respect to the different sets of attributes, a subspace clustering algorithm should find all possible clusters in which

a point participates, e.g. if a cluster C_1 in subspace $\{1, 3, 4\}$ contains points $\{P_3, P_6, P_7, P_8\}$ and another cluster C_2 in subspace $\{1, 3, 6\}$ contains points $\{P_1, P_3, P_4, P_6\}$, both of the clusters C_1 and C_2 should be detected. Note that, both points P_3 and P_6 are participating together in two different clusters in different subspaces.

2. The subspace clustering algorithm should give only non-redundant information i.e. if all the points in a cluster C_1 are also present in a cluster C_2 and the subspace in which C_1 exists is a subset of the subspace in which the cluster C_2 exists, then the cluster C_1 should not be included in the result, as the cluster C_1 does not give any additional information and is a trivial cluster. A strong conformity to this criterion would be that such redundant lower dimensional clusters are not generated at all, as their generation and pruning later on, leads to the higher computational cost. In other words, the subspace clustering algorithm should output only the *maximal subspace clusters*. As discussed earlier, a cluster is in a *maximal subspace* if there is no other cluster which conveys the same grouping information between the points as already given by this cluster.

There are two main approaches in the literature to deal with subspace clustering: top down and bottom up. Top down approaches like PROCLUS [30] and FINDIT [31] use projected clustering for high-dimensional data. An initial approximation of the number of the clusters and the relevant subspaces are given by the user. An objective function is chosen to iteratively regenerate the optimal quality clusters. Each data point is allocated to at most one cluster and the clusters are disjoint. Thus, these algorithms fail to detect all maximal clusters and do not conform to the 2^{nd} criterion of desirable properties described above. Neither do these algorithms satisfy the 1^{st} criterion, as only a user defined number of clusters is detected.

The bottom up approach is based on the downward closure property of the Apriori principle which was originally used for frequent item-set mining [32]. According to this principle, if a set of points from a k-dimensional space when projected onto a lower $(k-1)$ dimensional subspace is not dense, then this set cannot be dense in the k-dimensional space. Thus, it is important to process only the dense set of points from the lower dimensional spaces (starting from 1-dimensional) in a bottom up manner to compute the higher dimensional subspace clusters (Fig. 1). These algorithms can find all possible clusters in all the subspaces but they fail to scale with the dimensions [33–35, 40, 41].

Agrawal et al. [33] were the first to introduce the grid-density based approach in their famous CLIQUE algorithm to find the subspace clusters in the data. Each single dimension of the data space is partitioned into equal-sized ξ units using a fixed size grid. A unit is considered dense if the number of points in it exceeds the density support threshold, τ. The candidate k-dimensional units are determined by joining all $(k-1)$-dimensional dense units which share the first $(k-2)$ dimensions. MAFIA [42] and ENCLUS [40] proposed modifications over CLIQUE by using adaptive grid as well as more quality criteria for qualifying clusters. However, in all of these grid based approaches, the generation of $(k-1)$ dimensional units before k-dimensional units, leads to inefficiency as lots of redundant clusters are generated along the way. These approaches satisfy the 1^{st} criterion of the desired subspace clustering algorithm and can find all of the arbitrary shaped clusters, but they fail to satisfy the 2^{nd} criterion as they still generate many trivial clusters. Also,

their efficiency highly depends on the granularity and the positioning of the grid. Their scalability with respect to the higher dimensions is a common issue.

Similar to the previous bottom-up clustering approaches, SUBCLU [34] computes the higher dimensional clusters using the lower-dimensional clusters, iteratively starting with 1-dimensional clusters. The only difference is that it applies full-dimensional clustering algorithm DBSCAN to each subspace while computing the clusters. SUBCLU too generates all lower-dimensional trivial clusters and fails to satisfy the 2^{nd} criterion of the desired subspace clustering algorithm. Kriegel et al. proposed FIRES [35] which first computes 1-dimensional histograms called base clusters using any chosen clustering technique. These base clusters are then merged based on the number of the intersecting points to find the maximal subspace cluster approximations, but this algorithm requires multiple database scans.

Assent et al. proposed INSCY [41] for subspace clustering which is an extension of SUBCLU. They use a special index structure called a SCY-tree which can be traversed in the depth first order to generate high dimensional clusters. Their algorithm compares each data point of the base cluster and enumerates them in order to merge the base clusters for generating the higher-dimensional clusters. The search for the maximal subspace clusters in INSCY is quite exhaustive as they implicitly generate all intermediate trivial clusters during the bottom up clustering process and hence, INSCY too can not strongly conform to the 2^{nd} criterion of the desired subspace clustering algorithm. The complexity of INSCY is $O(2^k|\mathbf{DB}|^2)$, where k is the dimensionality of the maximal subspace cluster and $|\mathbf{DB}|$ denotes the size of the dataset.

The need for enumerating points in $O(2^k)$ subspaces using the multi-dimensional index structure introduces computational cost as well as inefficiency. All of these subspace clustering algorithms discussed so far require multiple database scans.

In the next Section, we propose SUBSCALE algorithm which is much more scalable with the size and dimensions of the data. It does not require multiple database scans and do not generate trivial clusters during the process. It directly computes the maximal subspace clusters from 1-dimensional dense points and is highly parallelizable.

Research design and methodology

In this section, we discuss SUBSCALE algorithm in detail, starting with the definitions and theoretical foundations.

Continuing with the monotonicity of the Apriori principle, a set of dense points in an k-dimensional space S is dense in all the lower dimensional projections of this space [33]. In other words, if we have the dense sets of points in each of the 1-dimensional projections of the attribute-set of a given data, then the sufficiently common points among these 1-dimensional sets will lead us to the dense points in the higher dimensional subspaces.

Based on this premise, we develop our algorithm to efficiently find the maximal clusters in all possible subspaces of a high-dimensional dataset. Table 1 contains the notations which will be used throughout this paper. For the ease of readability, we use the term '1-D' to represent any 1-dimensional point or group of points.

Definitions and problem

We assume $\mathbf{DB} = \{P_1, P_2, \ldots, P_n\}$, is a database of n points in a k-dimensional space, where each point is a k-dimensional vector, s.t. $P_i = \left\{P_i^1, P_i^2, \ldots, P_i^k\right\}$. A **subspace** S is a

Table 1 Notations used in this paper

Notation	Meaning
DB	The database of points
$\|T\|$	The cardinality of a set T
n	The total number of points, $n = \|\mathbf{DB}\|$
D	The set of attributes, $D = \{d_1, d_2, \ldots, d_k\}$
k	The total number of dimensions, $k = \|D\|$
d_i	The i^{th} dimension, $d_i \in D$
P_i	The i^{th} point, $P_i \in \mathbf{DB}$
S	A subspace, $S \subset D$
P_i^S	The i^{th} point projected on a subspace S
N^S	The neighbourhood of a point in a subspace S
$dist()$	The distance function to find neighbourhood
C	A cluster
CS	A core set of density connected points within ϵ distance
$\mathcal{U}$	A dense unit, $\|\mathcal{U}\| = \tau + 1$
1-D	One dimensional
$\mathcal{H}$	Signature of a dense unit
L	Large Integer
K	A set of random large integers, $\|K\| = n$
hTable	A hash table

subset of the original attribute-set D, such that, an m-dimensional subspace is denoted as $S = \{d_1, d_2, \ldots, d_m\}$ where, $d_i \in D$ and $1 \leq m \leq k$. A subspace S' is the projection of a higher-dimensional subspace S, if $S' \subset S$. The *dimensionality* of a subspace refers to the total number of dimensions participating to build this subspace.

In this paper, we adopt the definition of density from DBSCAN [9] which is based on two user defined parameters ϵ and τ, such that, a point is dense if it has at least τ points within ϵ distance. Formally, let the **neighbourhood** $N^S(P_i)$ of a point P_i in a particular subspace S be a set of all the points P_j such that $dist(P_i^S, P_j^S) < \epsilon$, $P_i \neq P_j$. $dist()$ is a similarity function based on the distance between the values of the points. A point P_i is **dense** in a subspace S, if $\left|N^S(P_i)\right| \geq \tau$ i.e. it has at least τ neighbours within ϵ distance. These dense points can be easily connected to form a subspace cluster.

Definition 1 (Density-Reachability [9])**.**
*If a point P_j is in the neighbourhood of a dense point P_i then, P_j is said to be directly-density-reachable from point P_i. The direct density reachability is not symmetric if both points are not dense. Two points P_i, P_j are said to be **density-reachable** from each other, if there is a chain of directly-density-reachable points between them, i.e., $\forall P_r, P_{r+1} \in \{P_i, P_1, \ldots, P_m, P_j\}$, P_{r+1} is directly density reachable from P_r.*

Definition 2 (Density-Connected [9])**.**
*Two points P_i, P_j are said to be **density-connected** with each other, if there is a point P_r such that both P_i and P_j are density-reachable from P_r. Both density reachability and connectivity is defined with respect to same ϵ and τ parameters.*

Definition 3 (Maximal Subspace Cluster)**.**
Referring to the previous definitions, a ***subspace cluster****,* $C = (P, S)$ *is a group of dense points P in a subspace S, such that* $\forall P_i, P_j \in P$*,* P_i *and* P_j *are density connected with each other in the subspace S with respect to* ϵ *and* τ *and there is no other point* $P_r \in P$*, such that* P_r *is density-reachable from some* $P_q \notin P$ *in the subspace S. A cluster* $C_i = (P, S)$ *is called a* ***maximal subspace cluster*** *if there is no other cluster* $C_j = (P, S')$ *such that* $S' \supset S$*.*

The maximality of a particular subspace is always relative to a cluster. A subspace which is maximal for a certain group of points might not be maximal for another group of points. It is to note that some of the related literature treat the maximality of the clusters in terms of the inclusion of all possible density-connected points (in a given subspace) into one cluster. We call it an inclusive property of clustering algorithm. Our 'maximal subspace clusters' are both inclusive (with respect to the points in a given subspace) and maximal (with respect to the lower dimensional projections). Next, we present the underlying idea behind our approach.

Basic idea

We observe in Fig. 2 that the two-dimensional $Cluster_5$ is an intersection of its 1-D projections of points in $Cluster_1$ and $Cluster_2$. Also, we note that the projections of the points $\{P_7, P_8, P_9, P_{10}\}$ on d_2-axis form a 1-D cluster ($Cluster_3$), but there is no 1-D cluster in the dimension d_1 which has the same points in it, which justify the absence of a two-dimensional cluster containing these points in the subspace $\{d_1, d_2\}$.

Given an m-dimensional cluster $C = (P, S)$ where, $S = \{d_1, d_2, \ldots d_m\}$, the projections of the points in P are dense points in each of the single dimension, $\{d_1\}, \{d_2\}, \ldots, \{d_m\}$. It implies that if a point is not dense in a 1-dimensional space then it will not participate in the cluster formation in the higher subspaces containing that dimension. Thus, we can combine the 1-D dense points in m different dimensions to find the density-connected

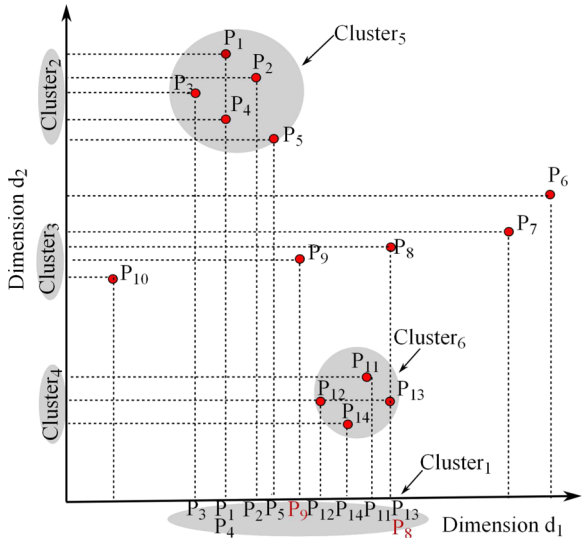

Fig. 2 Projections of dense points. Basic idea behind SUBSCALE. The projections of the points $\{P_7, P_8, P_9, P_{10}\}$ on d_2-axis form a 1-D cluster ($Cluster_3$), but no 1-D cluster in dimension d_1 having the same points as $Cluster_3$, thus, absence of corresponding 2-dimensional cluster containing these points in the subspace $\{d_1, d_2\}$

sets of points in the maximal subspace S. Recall that a point is *dense* if it has at least τ neighbours in ϵ neighbourhood with respect to a distance function *dist()*. We choose L_1 metric as the distance function to find the dense points in 1-dimensional subspaces.

Observation 1. *If at least $\tau + 1$ density-connected points from a dimension d_i also exist as density-connected points in the single dimensions $d_j, \ldots, d_r$, then these points will form a set of dense points in the maximal subspace, $S = \{d_i, d_j, \ldots d_r\}$.*

Following Definition 3, each point in a subspace cluster will belong to the neighbourhood of at least one dense point. Therefore, the smallest possible projection of a cluster from the higher dimensional subspace is of cardinality $\tau + 1$ and let us call it a **dense unit**, $\mathcal{U}$. If $\mathcal{U}_1^{d_i}$ and $\mathcal{U}_2^{d_j}$ are two 1-D dense units from the single-dimensions d_i and d_j respectively, we say that these are the **same** dense units if they contain the same points i.e. $\mathcal{U}_1^{d_i} = \mathcal{U}_2^{d_j}$, if $\forall P_i \left[P_i \in \mathcal{U}_1^{d_i} \leftrightarrow P_i \in \mathcal{U}_2^{d_j} \right]$.

Observation 2. *Following the observation 1, if the same dense unit $\mathcal{U}$ exists across m single dimensions, then $\mathcal{U}$ exists in a maximal subspace spanned by these m dimensions.*

In order to check if two dense units are same, we propose a novel idea of assigning signatures to each of these 1-D dense units. The rationale behind this is to avoid comparing the individual points among all dense units in order to decide whether they contain exactly same points or not. We can hash the signatures of these 1-D dense units from all k dimensions and the resulting collisions will lead us to the maximal subspace dense units (Observation 2). Our proposal for assigning signatures to the dense units is inspired by the work in number theory by Erdös and Lehner [43] which we explain in detail below.

Assigning signatures

If $L \geq 1$ is a positive integer, then a set $\{a_1, a_2, \ldots, a_\delta\}$ is called its partition, such that $L = \sum_{i=1}^{\delta} a_i$ for some $\delta \geq 1$ and $a_i > 0$ is called a summand. Also, let $p_\delta(L)$ be the total number of such partitions, when each partition has at most δ summands.

Erdös and Lehner studied these integer partitions by probabilistic methods and gave an asymptotic formula for $\delta = o(L^{\frac{1}{3}})$,

$$p_\delta(L) \sim \frac{\binom{L-1}{\delta-1}}{\delta!} \tag{1}$$

Observation 3. *Assume K be a set of random large integers, $\delta \ll |K| \ll p_\delta(L)$. Let $\mathcal{U}_1$ and $\mathcal{U}_2$ be the two sets of integers drawn from K s.t. $|\mathcal{U}_1| = |\mathcal{U}_2| = \delta$ and $\delta = o(L^{\frac{1}{3}})$. Let us denote the sums of the integers in these two sets as $sum(\mathcal{U}_1)$ and $sum(\mathcal{U}_2)$ respectively. We observe that if $sum(\mathcal{U}_1) = sum(\mathcal{U}_2) = L$, then $\mathcal{U}_1$ and $\mathcal{U}_2$ are **same** with an extremely high probability, if L is very large.*

Proof. From Eq. 1, for a very large positive integer L, if we take relatively very small partition size δ, then the number of unique fixed-sized partitions will be astronomically large. And the probability of getting a particular partition set of size δ is:

$$\frac{\binom{L-1}{\delta-1}}{\delta!} \Big/ \binom{L}{\delta} = \frac{(L-1)!\,\delta!\,(L-\delta)!}{(\delta-1)!\,(L-\delta)!\,\delta!\,L!} = \frac{1}{L(\delta-1)!} \tag{2}$$

It means the probability of randomly choosing the same partition again is extremely low. And this probability can be made very small by choosing a large value of L and relatively very small δ. Since L is the sum of the labels of δ points in a dense unit $\mathcal{U}$, L can be made very large if we choose very large integers as the individual labels. Thus, with $\delta = \tau + 1$, the two dense units $\mathcal{U}_1$ and $\mathcal{U}_2$ will contain the same points with very high probability, if $sum(\mathcal{U}_1) = sum(\mathcal{U}_2)$, provided this sum is very large. □

We randomly generate a set K of n large integers and use a one-to-one mapping $M : \mathbf{DB} \mapsto K$ to assign a unique *label* to each point in the database. The **signature** $\mathcal{H}$ of a dense unit $\mathcal{U}$ is given by the sum of the labels of the points in it. Thus, relying on observation 3, we can match these 1-D signatures across different dimensions without checking for the individual points contained in these dense units, e.g., if $\mathcal{U}_1^{d_1}, \mathcal{U}_2^{d_2}, \ldots \mathcal{U}_m^{d_m}$ are m dense units in m different single dimensions, with their points already mapped to the large integers, we can hash their signature-sums to a hashtable. If all the sums collide then these dense units are same (with very high probability) and exist in the subspace $\{d_1, d_2, \ldots, d_m\}$. Thus, the final collisions after hashing all dense units in all dimensions generate dense units in the relevant maximal subspaces. We can combine these dense units to get final clusters in their respective subspaces.

We ran a few numerical experiments to validate observation 3 and the results are shown in Fig. 3. A trial consists of randomly drawing a set of 4 labels from a given range of integers, e.g., while using 6-digit integers as labels, we have a range between 100,000 and 999,999 to choose from. All labels in a given set in a given trial uses the same integer range. Each time we fill a set with random integers, we store its sum in a common hash table (separate for each integer range). A collision occurs when the sum of a randomly drawn set is found to be same as an already existing sum in the hash table. We note that the number of collisions is indeed very small when large integers are used as labels, e.g,

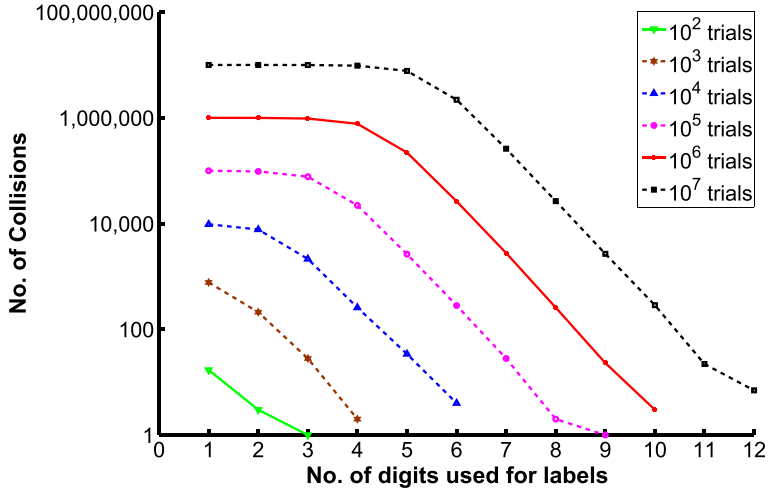

Fig. 3 Numerical experiments for probability of collisions. A trial includes drawing a label set of 4 random integers with the same number of digits e.g., {333, 444, 555, 666} is a sample set where no. of digits is 3. The probability of drawing same set of integers reduces drastically with the use of larger integers as labels in the set, e.g., ≈ 10 collisions for 100 million trials when the label is a 12-digit integer

there are about 10 collisions (a negligible number) for 100 million trials when the label is a 12-digit integer.

Also, we observed the probability of collisions by experimenting with different cardinalities of such partition sets drawn at random from a given integer range. We note in Fig. 4 that the probability of collisions decreases further with the higher values of set sizes. The gap between the number of collisions widens for the larger integer ranges.

Interleaved dense units

Although matching 1-D dense units across 1-dimensional subspaces is a promising approach to directly compute the maximal subspace clusters, it is however difficult to identify these 1-D dense units. The reason is that 1-dimensional subspaces may contain interleaved projections of more than one cluster from the higher-dimensional spaces, e.g., in Fig. 2, $Cluster_1$ contains projections from both $Cluster_5$ and $Cluster_6$. The only way to find all possible dense units from $Cluster_1$ is to generate all possible $\binom{|Cluster_1|}{\tau+1}$ combinations where, $()$ denotes a standard combinatorial expression.

Let CS be a **core-set** of such interleaved 1-D dense points such that, each point in this core set is within ϵ distance of each other such that, $|CS| > \tau$ where $|\ |$ denotes cardinality of a set. A core-set of the points in a subspace S can be denoted as CS^S with S as the superscript. In our algorithm, we first find these core-sets in each dimension and then generate all combinations of size $\tau + 1$ as potential dense units.

As can be seen from Fig. 2, many such combinations in the d_1 dimension will not result in dense units in the subspace $\{d_1, d_2\}$. Moreover, it is possible that none of the combinations will convert to any higher dimensional dense unit. The construction of 1-dimensional dense units is the most expensive part of our algorithm, as the number of combinations of $\tau + 1$ points can be very high depending on ϵ as the value of ϵ will determine the size of the core-sets in each dimension. However, one clear advantage of our approach is that this is the only time we need to scan the dataset in the entire algorithm.

SUBSCALE algorithm

As discussed before, we aim to extract the subspace clusters by finding the dense units in the relevant subspaces of the given dataset by using L_1 metric as the distance measure.

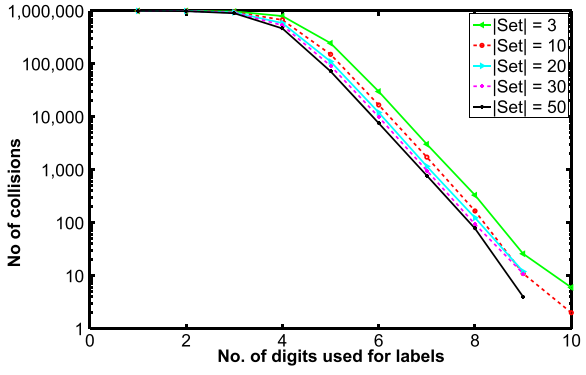

Fig. 4 Experiments with Erdos Lemma. Numerical experiments for probability of collisions with respect to the number of random integers drawn at a time. |*Set*| denotes the cardinality of the set drawn. The number of trials for each fixed integer-set is 1000000

We assume a fixed size of $\tau + 1$ for the dense unit $\mathcal{U}$ which is the smallest possible cluster of dense points. If $|CS|$ is the number of dense points in a 1-D core-set CS then, we can obtain $\binom{|CS|}{\tau+1}$ dense units from one such CS. If we map the data points contained in each dense unit to large integers and compute their sum then each such sum will create a unique signature. From observation 3, if two such signatures match then their corresponding dense units contain the same points with a very high probability. In SUBSCALE algorithm, we first find these 1-D dense units in each dimension and then hash their signatures to a common hash table (Fig. 5). Table 1 contains the notations which will be used throughout this algorithm.

We now explain our algorithm for generating maximal subspace clusters in high-dimensional data.

Step 1: Consider a set, K of very large, unique and random positive integers $\{K_1, K_2, \ldots, K_n\}$. We define M as a *one-to-one* mapping function, $M : \mathbf{DB} \rightarrow K$. Each point $P_i \in \mathbf{DB}$ is assigned a unique random integer K_i from the set K.

Step 2: In each dimension j, we have projections of n-points, $P^j_1, P^j_2, \ldots, P^j_n$. We create all possible dense units containing $\tau + 1$ points that are within an ϵ distance. Instead of the actual points, a dense unit $\mathcal{U}$ will now contain the mapped keys i.e. $\{K_1, K_2, \ldots, K_{\tau+1}\}$.

Step 3: Then, we create a hash table *hTable*, as follows. In each dimension j, for every dense unit $\mathcal{U}^j_a$ we calculate the sum of its elements called **signature**, $\mathcal{H}^j_a$ and hash this signature in *hTable*. Using observation 3, if $\mathcal{H}^j_a$ collides with another signature $\mathcal{H}^k_b$ then the dense unit $\mathcal{U}^j_a$ exists in subspace $\{j, k\}$ with extremely high probability. After repeating this process in all single dimensions, each entry of this hash table will contain a dense unit in the maximal subspace as we can store the colliding dimensions against each signature $\mathcal{H}_i \in$ *hTable*.

Step 4: We now have dense units in all possible maximal subspaces. We process them to create density-reachable sets and hence, maximal clusters. We use DBSCAN in each found subspace for the clustering process and the value of ϵ and τ can be adapted differently as per the dimensionality of the subspace to deal with the curse of dimensionality.

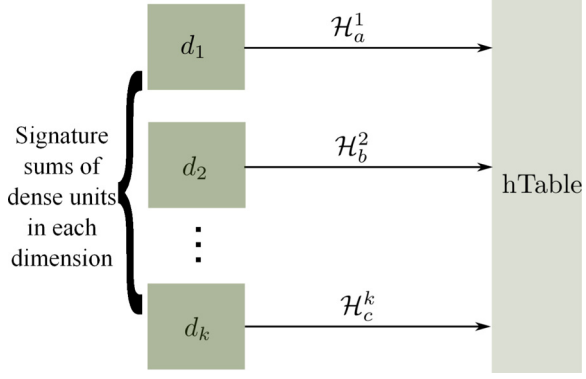

Fig. 5 Collisions among signatures. Signatures from different dimensions collide to identify the relevant subspaces for the corresponding dense units behind these signatures. d_i is the i^{th} dimension and $\mathcal{H}^i_x$ is the signature of dense unit, $\mathcal{U}^i_x$ in i^{th} dimension

The pseudo code is given in Algorithm 1 (*FindDenseUnits*) and Algorithm 2 (*SUBSCALE*) below.

Algorithm 1 FindDenseUnits: Find dense units with their signature sums

Require: Dimension j, A set K of n unique random and large integers

Sort $P_1, P_2, \ldots, P_n$, s.t. $\forall P_x, P_y \in \mathbf{DB}$, $P_x^j \leq P_y^j$
for $i = 1$ **to** $n - 1$ **do**
  $CS \leftarrow M(P_i)$
  // $M : P_i \mapsto K_i$, M is a mapping function
  $numNeighbours \leftarrow 1$
  $next \leftarrow i + 1$
  while $next \leq n$ **and** $P_{next}^j - P_i^j < \epsilon$ **do**
    Append $M(P_{next})$ to CS
    Increment $numNeighbours$
    Increment $next$
  end while
  if $numNeighbours > \tau$ **then**
    Find all combinations of dense units, $\mathcal{U}$ from the core-set CS ($\binom{|CS|}{\tau + 1}$)
  end if
  for all dense unit $\mathcal{U}$ found in the previous step **do**
    Add entry $\{sum(\mathcal{U}), \mathcal{U}\}$ to DU // DU is the data structure to store dense units along with their corresponding signatures in a given dimension
  end for
end for
return DU
// Each entry in DU is a pair $\{sum, \mathcal{U}\}$

In the next Section, we evaluate and discuss the performance of our proposed subspace clustering algorithm.

Results and discussion

We implemented SUBSCALE algorithm and experimented with various datasets upto 6144 dimensions (Table 2). Also, we compared the results from our algorithm with the other relevant state-of-the-art algorithms. The details of the experimental setup, code and datasets are given in the Methods Section. We *fixed* the value of $\tau = 3$ unless stated otherwise and experimented with different values of ϵ for each dataset, starting with the lowest possible value. The minimum cluster size (*minSize*) is set to 4.

Execution time and quality

The effect of changing ϵ values on runtime for 5 and 50 dimensional datasets of same size is shown in Fig. 6. The larger ϵ values result in bigger neighborhoods and generate large number of combinations, therefore, the execution time is higher.

One of the most commonly used methods to assess the quality of the clustering result in the related literature is F1 measure [44]. The open source framework by Müller et al. [45] was used to assess the F1 quality measure of our results. According to F1 measure, the

Algorithm 2 SUBSCALE: Find maximal subspace clusters

Require: **DB** of $n \times k$ points; Set K of n unique, random and large integers

Hashtable $hTable \leftarrow \{\}$
// An entry h_x in hTable is $\{sum, \mathcal{U}, S\}$.
for $j = 1$ **to** k **do**
  $DU \leftarrow FindDenseUnits(j)$
  // Get candidate dense units in dimension j.
  for all du = $\{sum, \mathcal{U}\} \in DU$ **do**
    if some key of hashtable $hTable.h_x == sum$ **then**
      Append dimension j to subspace $h_x.S$
    else
      Add new entry $\{sum, \mathcal{U}, j\}$ to the $hTable$
    end if
  end for
end for
for all set of entries $\{h_x, h_y, \ldots\} \in$ hTable **do**
  if $h_x.S = h_y.S = \ldots$ **then**
    Add entry $\{S, h_x.\mathcal{U} \cup h_y.\mathcal{U} \cup \ldots\}$ to *Clusters*
    // *Clusters* contain maximal dense units in the relevant subspaces
  end if
end for
Run *DBSCAN* on each entry of *Clusters*
return *Clusters*
// *Clusters* is resulting set of maximal subspace clusters

clustering algorithm should cover as many points as possible from the hidden clusters and as fewer as possible of those points which are not in the hidden clusters. F1 is computed as the harmonic mean of *recall* and *precision*. The recall value accounts for the coverage of points in the hidden clusters by the found clusters. The precision value measures the coverage of points in found clusters from other clusters. A high recall and precision value means high $F1$ and thus, better quality [44].

Table 2 Data Sets used for evaluation

Data	Size	Dimensionality
D05	1595	5
D10	1595	10
D15	1598	15
D20	1595	20
D25	1595	25
D50	1596	50
S1500	1595	20
S2500	2658	20
S3500	3722	20
S4500	4785	20
S5500	5848	20
madelon	4400	500
pedestrian	3661	6144

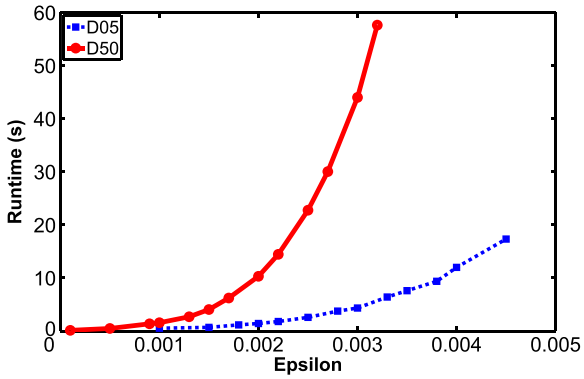

Fig. 6 Effect of ϵ on runtime. The large value of ϵ results in bigger neighbourhood of points and hence, larger number of possible combinations. SUBSCALE execution time increases with bigger ϵ value. Two datasets of same size (1595) but of different dimensions (5 and 50) were evaluated

Figure 7 shows the change in F1 values for 20-dimensional datasets of varying sizes with respect to ϵ. We notice that the quality of clusters deteriorates beyond a certain threshold of ϵ for each data set because clusters can get artificially large due to the larger ϵ values. Thus, a larger value of ϵ is not always necessary to get better quality clusters.

We evaluated SUBSCALE algorithm against INSCY which is one of the most recent subspace clustering algorithm. As shown in Fig. 8, SUBSCALE gave much better runtime performance for similar quality of clusters ($F1 = 0.9$), particularly for the higher dimensional datasets.

Figure 9 shows the effect of increasing the dimensionality of the data on the runtime for SUBSCALE versus other subspace clustering algorithms (INSCY, SUBCLU and CLIQUE available on Open source framework [45]), keeping the data size fixed (Dataset used: D05, D10, D15, D20, D25). We noticed that the performance of CLIQUE algorithm is worst in terms of scalability with respect to the number of dimensions. For CLIQUE implementation, the density threshold of 0.003 and *gridSize* of 3 was used. Same values of $\epsilon = 0.001$ and $\tau = 3$ were used for SUBSCALE, INSCY and SUBCLU algorithms. SUBCLU did not give meaningful clusters because even single points appeared as clusters in the result and a smallest cluster should have at least $\tau + 1$ points. And the CLIQUE algorithm did

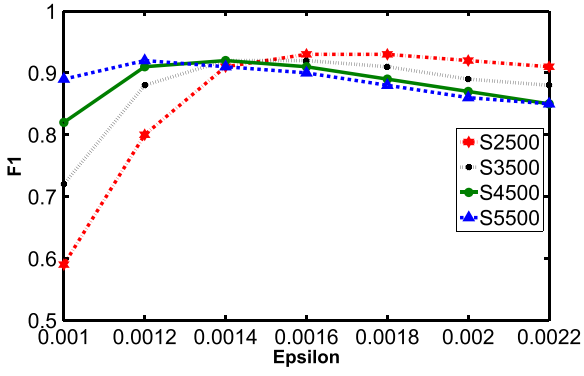

Fig. 7 Effect of ϵ on clustering quality (F1 measure). Dimensionality of datasets used is 20 and size varies. The clustering quality scales down for higher ϵ. Thus, higher ϵ value is not always required for quality clusters

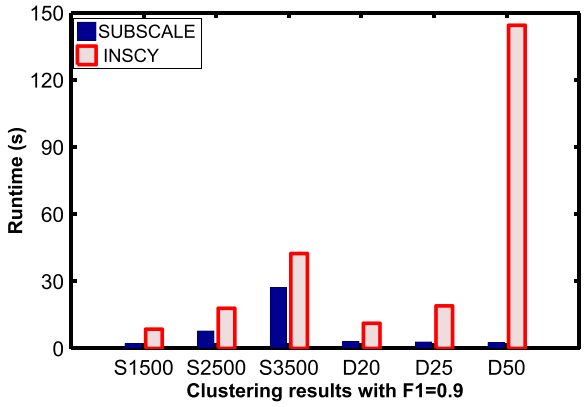

Fig. 8 Runtime comparison for similar quality of clusters in different datasets. For the same quality of clustering results ($F1 = 0.9$), SUBSCALE performed much better for the execution time than INSCY

not execute for more than 15 dimensions. We observe that SUBSCALE clearly performs better than the rest of the algorithms with the increase in number of dimensions. Figure 10 shows the runtime comparison of our algorithm with INSCY and SUBCLU for the fixed number of dimensions ($= 20$) and increasing the size of the data (Dataset used: S1500, S2500, S3500, S4500, S5500). We did not include CLIQUE in Fig. 10 as it crashed for more than 15 dimensional data. We can see that the runtime for SUBSCALE is less than INSCY for the same data. Although the execution time for SUBCLU algorithm seems to be the lowest in both Figs. 9 and 10 but the clustering results were not meaningful as mentioned before.

The $\mathbf{4400} \times \mathbf{500}$ *madelon* dataset available at UCI repository [46] has $\sim \mathbf{2^{500}}$ possible subspaces. We ran SUBSCALE on this dataset to find all possible subspace clusters and Fig. 11 shows the runtime performance with respect to ϵ values. The different values of ϵ ranging from 1.0×10^{-5} to 1.0×10^{-6} were used. We tried to run INSCY by allocating upto 12GB RAM but it failed to run for this high dimensional dataset for any of the ϵ value.

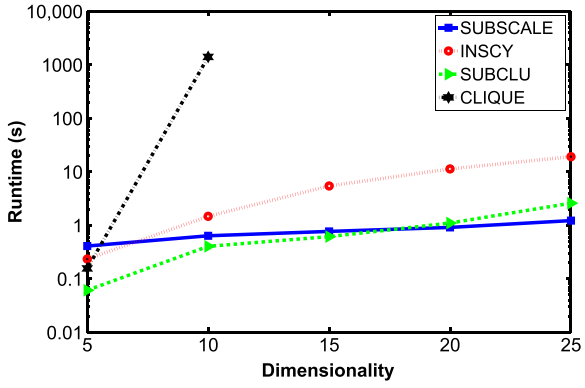

Fig. 9 Runtime comparison for fixed data size=1595. Parameters for CLIQUE: $\tau = 0.003$, *gridSize* $= 3$. Parameters for SUBSCALE, INSCY and SUBCLU (similar values): $\epsilon = 0.001$, $\tau = 3$, *minSize* $= 4$. CLIQUE crashed after 10 dimensions. SUBCLU didn't give meaningful clusters. Dataset used: D05, D10, D15, D20, D25

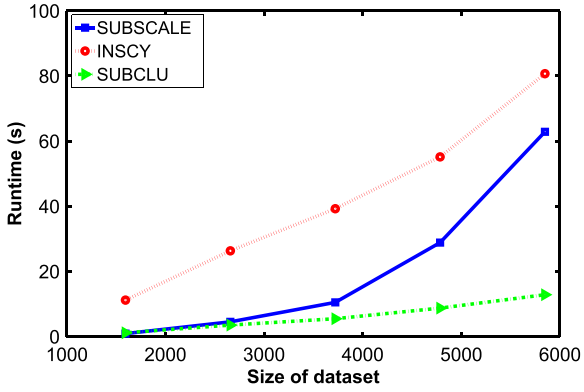

Fig. 10 Runtime comparison for fixed dimensionality=20. Same parameters as in Fig. 8 were used. CLIQUE could not run for this dimensionality. SUBCLU didn't give meaningful clusters. Dataset used: S1500, S2500, S3500, S4500, S5500

Scalability

The performance of our algorithm depends heavily on the number of dense units generated in each dimension. And the number of these dense units depends upon the size of the data, chosen value of ϵ and the underlying data distribution. A larger ϵ value increases the range of the neighbourhood of a point and is likely to produce bigger core-sets and thus, dense units. As we do not have prior information about the underlying data distribution, a single dimension can have any number of dense units. Thus, for a scalable solution to subspace clustering through SUBSCALE, the system must be able to store and process large number of collisions of the dense units.

In order to identify collisions among dense units across multiple dimensions, we need the collision space (*hTable*) big enough to hold these dense units in the working memory of the system. But with the limited memory availability, this is not always possible. If num^j is the number of total dense units in a dimension j, then a k dimensional dataset may have $NUM = \sum_{j=1}^{k} num^j$ dense units in total. The signature technique used in SUBSCALE algorithm has a huge benefit that we can split NUM to the granularity of k.

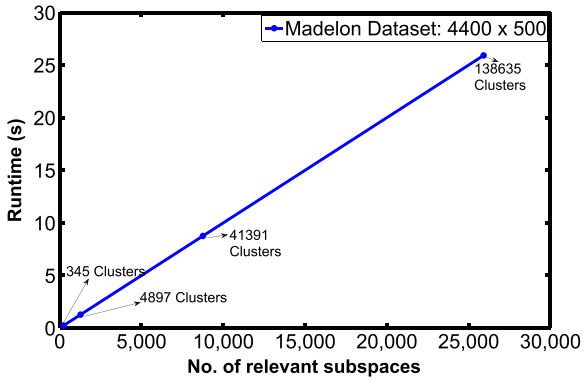

Fig. 11 No of found subspaces/clusters vs runtime (*madelon* dataset). Different *epsilon* values ranging from $1.0E-5$ to $1.0E-6$ were used. The number of clusters as well as subspaces in which these clusters are found, increases with the increase in ϵ value. Other parameters used: $\tau = 3, minSize = 4$

As each dense unit contains $\tau + 1$ points and we are assigning large integers to these points from the key database K to generate signatures $\mathcal{H}$, the value of any signature thus generated would approximately lie between the range $R = \Big((\tau + 1) \times min(K), (\tau + 1) \times max(K)\Big)$, where $min(K)$ is the smallest key being used and $max(K)$ is the largest key being used. The detection of maximal dense units involves matching of the same signature across multiple dimensions using a hash table. Thus, the hash table should be able to retain all signatures in the range R. But we can split this range into multiple chunks so that each chunk can be processed independently using a much smaller hash table. If sp is the split factor, we can divide this range R into sp parts and thus, into sp hash tables where each *hTable* holds at the most $\frac{R}{sp}$ entries for these dense units. But since the keys are being generated randomly from a fixed range of digits, the actual entries will be very less. In a 14-digit integer space, we have 9×10^{13} keys to choose from (9 ways to choose the most significant digit and 10 ways each for the rest of the 13 digits). The number of actual keys being used will be equal to the number of points in the dataset $|\mathbf{DB}|$.

In Fig. 12, we illustrate this with a range of 4-digit integers from 1000 to 9999. Let $|\mathbf{DB}| = 500$ and so we need 500 random keys from 9000 available integer keys. If $\tau = 3$ then, some of these 500 keys will form dense core-sets. Let us assume that $1/5^{th}$ of these keys are in a core-set in some 1-dimensional space, then we would need a hash table big enough to store ≈ 4 million entries i.e. $\binom{100}{4}$. If we choose the split factor of 3 then we have 3 hash tables where each hashtable can store approx 1 million of entries. Typically, Java mostly uses 8 bytes for a *long* key, so 32 bytes for a signature with $\tau = 3$ and additional bytes to store the colliding dimensions (say 40 bytes per entry for an average subspace dimensionality of 10), are required.

We ran SUBSCALE algorithm on *madelon* dataset [46] using different values of *sp* ranging from 1 to 500 and Fig. 13 shows its effect on the runtime performance for two different

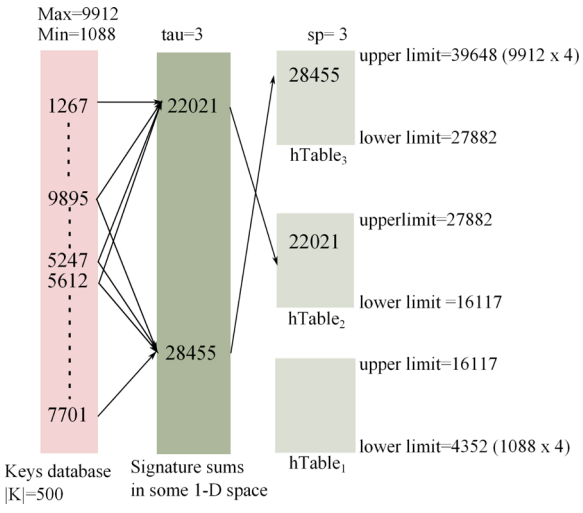

Fig. 12 Illustration of *hTable* splitting. For $\tau = 3$, Split factor $sp = 3$, minimum large-key value $min(K) = 1088$ and maximum large-key value $max(K) = 9912$, approximate range of expected signature sums is $(1088 \times (\tau + 1)\ , 9912 \times (\tau + 1))$. Each signature sum is derived from a dense unit of fixed size = $\tau + 1$

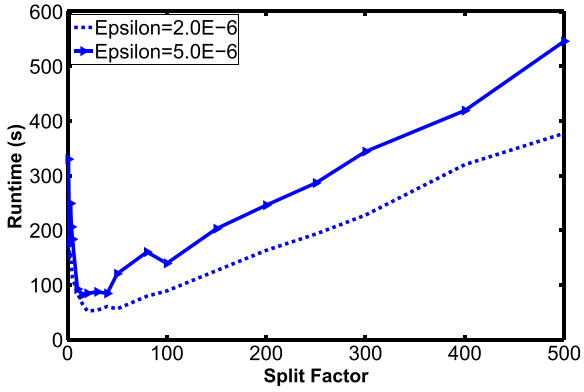

Fig. 13 Runtime vs split factor (*madelon* dataset). The execution time is almost proportional to the split factor after an initial threshold

values of ϵ. The darker line is for larger value of ϵ and hence, higher runtime. But for both of the ϵ values, the execution time is almost proportional to the split factor after an initial threshold.

The size of $hTable_i$ can be adjusted to handle large databases of points by choosing an appropriate split factor *sp*. Instead of generating all dense units in all dimensions *sp* times, if we sort the keys of density-connected points, we can stop the computation of dense units when no more signature sums less than the *upperlimit* of $hTable_i$ are possible.

We successfully ran modified SUBSCALE algorithm for $\mathbf{3661 \times 6144}$ pedestrian dataset [47, 48] (see Methods Section for details of this data). We designed two sets of experiments: a. $\epsilon = 0.000001, sp = 4000$; b. $\epsilon = 0.000002, sp = 6000$. Also, $\tau = 3$ was used for both sets of experiments. We ran the first set on an Intel Core i7-2600 desktop computer with 14GB of allocated RAM for processing and it took ≈336 hrs to finish. About 48 million subspaces containing clusters were discovered out of 2^{6144} search space. For the second set of experiment, since we doubled the ϵ value and larger number of dense units were expected from higher ϵ value, we increased the split factor *sp* to 6000. Instead of using just one computer, we distributed the 6000 hash computations among 64 computers. Each computer received between 50 to 1000 hash computations and 7GB RAM was allocated for each computation on each computer. The computer systems were running either 64-bit Windows or Linux operating system. Again, $\approx$ 48 million subspaces having clusters were found and the computation time was 783 hours in total for all the computers. But the execution time on each computer varied from minimum 4 to maximum 76 hours. However, the run time can be reduced further by increasing the split factor *sp* and distributing the computations to more CPUs.

We encountered memory overflow problems when handling a large number of result subspace clusters and that was due to the increasing size of *Clusters* data structure used in Algorithm 2. We found a solution by distributing the dense points of each identified maximal subspace from the hash table to a relevant file on the secondary storage. The relevance is determined by the cardinality of the found subspace. If a set of points are found in a maximal subspace of cardinality *m*, we can store these dense points in a

file named 'm.dat' along with their relevant subspaces. It will also facilitate running any full-dimensional algorithm like DBSCAN on each of these files as its parameters can be set according to different dimensionality.

Determining the input parameters

An unsupervised clustering algorithm has no prior knowledge of the density distribution for the underlying data. Even though the choice of density measures (ϵ, τ and *minSize*) is very important for the quality of subspace clusters, finding their optimal values is a challenging task.

SUBSCALE initially requires both τ and ϵ to find 1-D dense points. Once we identify the dense points in the maximal subspaces through SUBSCALE, we can then run DBSCAN for the identified subspaces by setting the τ, ϵ and *minSize* parameters according to each subspace. We should mention that finding clusters from these subspaces by running DBSCAN or any other density based clustering algorithm will take relatively very less time. During our experiments, the time taken by DBSCAN on an average comprises less than 5 % of the total execution time for the evaluation datasets given in Table 2. The reason is that each of the identified maximal subspace by SUBSCALE has already been pruned for only those points which have very high probabilities of forming clusters in these subspaces.

In our experiments, we started with the smallest possible ϵ-distance between any two 1-D projections of points in the given dataset. Considering the curse of dimensionality, the points become farther in high dimensional subspaces, so the user may intend to find clusters with larger ϵ values than that used by SUBSCALE for 1-D subspaces. Most of the subspace clustering algoritms use heuristics to adjust these parameters in higher dimensional subspaces. Some authors [49, 50] have suggested the methods to adapt ϵ and τ parameters for the high dimensional subspaces. However, we would argue that the choice of these density parameters is highly subjective to the individual data set as well as the user requirements.

Conclusions

The generation of large and high dimensional data in the recent few years has overwhelmed the data mining community. In this paper, we have presented a novel approach to efficiently find quality subspace clusters without expensive database scans or generating trivial clusters in between. We have validated our idea both theoretically as well as through the numerical experiments. We have compared SUBSCALE algorithm against recent subspace clustering algorithms and our proposed algorithm has performed far better when it comes to handling high-dimensional datasets. We have experimented with upto 6144 dimensional data and we can safely claim that it will work for larger datasets too by adjusting the *splitfactor*. However, the main cost in our SUBSCALE algorithm is the computation of the candidate 1-dimensional dense units. In addition to splitting the hash table computation, SUBSCALE has a high degree of parallelism as there is no dependency in computing dense units across multiple dimensions. We plan to implement a parallel version of our algorithm on General Purpose Graphics Processing Units (GPGPU) in the near future.

Since our algorithm directly generates the maximal dense units, it is possible to implement a query-driven version of our algorithm relatively easily. Such an algorithm will take

a set of (query) dimensions and find the clusters in the subspace determined by this set of dimensions.

Methods

We implemented the SUBSCALE algorithm in Java language on an Intel Core i7-2600 desktop with 64-bit Windows 7 OS and 16GB RAM. The dense points in maximal subspaces were found by SUBSCALE and then for each found subspace containing dense set of points, we used Python Script to apply DBSCAN algorithm from the *scikit* library [51]. However, any full-dimensional density-based algorithm can be used instead of DBSCAN. The open source framework by Müller et al. [45] was used to assess the quality our results and also to compare these results with the other clustering algorithms available through the same platform.

The datasets in Table 2 were normalised between 0 and 1 using WEKA [52] and contained no missing values. The *padestrian* data set is extracted from the attributed pedestrian database [47, 48] using the Matlab code given in APiS1.0 [53]. The *madelon* data set available at UCI repository [46]. The rest of the data sets are freely available at the website of authors of the related work [45].

The source code of SUBSCALE algorithm can be downloaded from the Git repository [54] and its scalable version is available at [55].

Competing interests
The authors declare that they have no competing interests.

Authors' contributions
AD contributed for the underlying idea, helped drafting the manuscript and played a pivotal role guiding and supervising throughout, from initial conception to the final submission of this manuscript. AK developed and implemented the idea, designed the experiments, analysed the results and wrote the manuscript. Both authors read and approved the final manuscript.

Authors' information
AK is currently working toward the PhD degree at the University of Western Australia. Her research interests include data mining, parallel computing and information security. AD is a Professor at the School of Computer Science and Software Engineering at the University of Western Australia. His research interests are in parallel and distributed computing, mobile and wireless computing, bioinformatics, social networks, data mining and software testing.

Acknowledgements
AK was financially supported for her research by the Australian Government through Endeavour Postgraduate Award (http://www.australiaawards.gov.au/). The authors would like to acknowledge the support provided by the University of Western Australia. The authors would also like to thank the four anonymous reviewers for their insightful comments that improved the presentation of the paper considerably.

Received: 27 May 2015 Accepted: 31 July 2015
Published online: 12 August 2015

References

1. Barrett T, Wilhite SE, Ledoux P, Evangelista C, Kim IF, Tomashevsky M, Marshall KA, Phillippy KH, Shermpan PM, Holko M, Yefanov A, Lee H, Zhang N, Robertson CL, Serova N, Davis S, Soboleva A (2013) Ncbi geo: archive for functional genomics data sets-update. Nucleic Acids Res 41(D1):991–995
2. Dewdney PE, Hall PJ, Schilizzi RT, Lazio TJLW (2009) The square kilometre array. Proc IEEE 97(8):1482–1496
3. Fan J, Han F, Liu H (2014) Challenges of big data analysis. National Science Review 1(2):293–314
4. Steinbach M, Ertöz L, Kumar V (2004) The challenges of clustering high dimensional data. In: New directions in statistical physics. Springer, Berlin Heidelberg. pp 273–309
5. Aggarwal CC, Reddy CK (2013) Data clustering: algorithms and applications. Data Mining Knowledge and Discovery Series 1st. CRC Press
6. Xu R, Wunsch D II (2005) Survey of clustering algorithms. Neural Netw IEEE Trans on 16(3):645–678
7. Manning CD, Raghavan P, Schütze H (2008) Hierarchical clustering. In: Introduction to information retrieval Vol. 1. Cambridge university press, New York, USA
8. Zhang T, Ramakrishnan R, Livny M (1996) BIRCH: an efficient data clustering method for very large databases. In: Proc. of the ACM SIGMOD international conference on management of data, vol. 1. ACM Press, USA. pp 103–114
9. Ester M, Kriegel H, Sander J, Xu X (1996) A density-based algorithm for discovering clusters in large spatial databases with noise. Int Conf Knowl Discov Data Min 96(34):226–231

10. Bellman RE (1961) Adaptive control processes: a guided tour. Princeton University Press, New Jersey
11. Beyer K, Goldstein J (1999) When is nearest neighbor meaningful? Proc 7th Int Conf Database Theory. In: Database Theory –ICDT'99. Lecture Notes in Computer Science. Springer, Berlin Heidelberg Vol. 1540. pp 217-235
12. Parsons L, Haque E, Liu H (2004) Subspace clustering for high dimensional data: a review. ACM SIGKDD Explor Newsl 6(1):90–105
13. Babu MM (2004) Introduction to microarray data analysis. In: Grant RP (ed). Computational genomics: Theory and application. Horizon Press, UK. pp 225–249
14. Eisen MB, Spellman PT, Brown PO, Botstein D (1998) Cluster analysis and display of genome-wide expression patterns. Proc Natl Acad Sci 95(25):14863–14868
15. Jiang D, Tang C, Zhang A (2004) Cluster analysis for gene expression data: a survey. IEEE Trans Knowl Data Eng 16(11):1370–1386
16. Cheng Y, Church GM (2000) Biclustering of expression data. Proc Int Conf Intell Syst Mol Biol 8:93–103
17. Yoon S, Nardini C, Benini L, De Micheli G (2005) Discovering coherent biclusters from gene expression data using zero-suppressed binary decision diagrams. IEEE/ACM Trans Comput Biol Bioinforma 2(4):339–353
18. Huttenhower C, Mutungu KT, Indik N, Yang W, Schroeder M, Forman JJ, Troyanskaya OG, Coller HA (2009) Detailing regulatory networks through large scale data integration. Bioinformatics 25(24):3267–3274
19. Jun J, Chung S, McLeod D (2006) Subspace clustering of microarray data based on domain transformation. In: Data Mining and Bioinformatics. Lecture Notes in Computer Science, vol. 4316. Springer, Heidelberg. pp 14–28
20. Eren K, Deveci M, Kktun O, atalyrek mV (2013) A comparative analysis of biclustering algorithms for gene expression data. Brief Bioinforma 14(3):279–292
21. Basri R, Jacobs DW (2003) Lambertian reflectance and linear subspaces. IEEE Trans Pattern Anal Mach Intell 25(2):218–233
22. Elhamifar E, Vidal R (2013) Sparse subspace clustering: algorithm, theory, and applications. IEEE Trans Pattern Anal Mach Intell 35(11):2765–2781
23. Vidal R (2011) Subspace clustering. IEEE Signal Proc Mag 28(2):52–68
24. Ho J, Yang MH, Lim J, Lee KC, Kriegman D (2003) Clustering appearances of objects under varying illumination conditions. In: Computer vision and pattern recognition, 2003. Proceedings. 2003 IEEE computer society conference on, vol. 1. IEEE. pp 1–11
25. Tierney S, Gao J, Guo Y (2014) Subspace clustering for sequential data. In: Computer vision and pattern recognition (CVPR), 2014 IEEE conference On. IEEE. pp 1019–1026
26. Vidal R, Tron R, Hartley R (2008) Multiframe motion segmentation with missing data using PowerFactorization and GPCA. Int J Comput Vis 79(1):85–105
27. Günnemann S, Boden B, Seidl T (2012) Finding density-based subspace clusters in graphs with feature vectors. In: Data mining and knowledge discovery. Springer, US Vol. 25. pp 243–269
28. Jang W, Hendry M (2007) Cluster analysis of massive datasets in astronomy. Stat Comput 17(3):253–262
29. Li T, Ma S, Ogihara M (2004) Document clustering via adaptive subspace iteration. In: Proceedings of the 27th annual international ACM SIGIR conference on research and development in information retrieval. ACM, USA. pp 218–225
30. Aggarwal CC, Wolf JL, Yu PS, Procopiuc C, Park JS (1999) Fast algorithms for projected clustering. In: Proc. of the ACM SIGMOD international conference on management of data. ACM, USA. pp 61–72
31. Woo KG, Lee JH, Kim MH, Lee YJ (2004) FINDIT: a fast and intelligent subspace clustering algorithm using dimension voting. Inf Softw Technol 46(4):255–271
32. Agrawal R, Mannila H, Srikant R, Toivonen H, Verkamo AI (1996) Fast discovery of association rules. Adv Knowl Discov Data Min 12:307–328
33. Agrawal R, Gehrke J, Gunopulos D (1998) Automatic subspace clustering of high dimensional data for data mining applications. In: Proc. of the ACM SIGMOD international conference on management of data. pp 94–105
34. Kailing K, Kriegel HP, Kroger P (2004) Density-connected subspace clustering for high-dimensional data. In: SIAM international conference on data mining. pp 246–256
35. Kriegel H-PH, Kroger P, Renz M, Wurst S (2005) A generic framework for efficient subspace clustering of high-dimensional data. In: IEEE international conference on data mining. IEEE, Washington, DC, USA. pp 250–257
36. Sim K, Gopalkrishnan V, Zimek A, Cong G (2012) A survey on enhanced subspace clustering. Data Min Knowl Disc 26(2):332–397
37. Kaur A, Datta A (2014) Subscale: fast and scalable subspace clustering for high dimensional data. In: Data mining workshop (ICDMW), 2014 IEEE international conference on. IEEE. pp 621–628
38. Joliffe IT (2002) Principle component analysis. 2nd edn. Springer, New York
39. Kriegel HP, Kröger P, Zimek A, Oger PKR (2009) Clustering high-dimensional data: a survey on subspace clustering, pattern-based clustering, and correlation clustering. ACM Trans Knowl Discov Data 3(1):1–58
40. Cheng CH, Fu AW, Zhang Y (1999) Entropy-based subspace clustering for mining numerical data. In: ACM SIGKDD international conference on knowledge discovery and data mining. ACM, NY, USA. pp 84–93
41. Assent I, Emmanuel M, Seidl T (2008) Inscy: Indexing subspace clusters with in-process-removal of redundancy. In: Eighth IEEE international conference on data mining. IEEE. pp 719–724
42. Nagesh H, Goil S, Choudhary A (2001) Adaptive grids for clustering massive data sets. Proc 1st SIAM Int Conf Data Min:pp. 1–17
43. Erdös P, Lehner J (1941) The distribution of the number of summands in the partitions of a positive integer. Duke Mathematical Journal 8(2):335–345
44. Müller E, Günnemann S, Assent I, Seidl T, Emmanuel M, Stephan G (2009) Evaluating clustering in subspace projections of high dimensional data. In: International conference on very large data bases. VLDB Endowment, Lyon, France Vol. 2. pp 1270–1281
45. Müller E, Günnemann S, Assent I, Seidl T, Färber I (2009) Evaluating Clustering in Subspace Projections of High Dimensional Data. http://dme.rwth-aachen.de/en/OpenSubspace/evaluation. Accessed 08 Aug 2015
46. Bache K, Lichman M (2006) UCI machine learning repository. http://archive.ics.uci.edu/ml. Accessed 08 Aug 2015

47. Geiger A, Lenz P, Stiller C, Urtasun R (2013) Vision meets robotics: The KITTI dataset. Int J Rob Res 32 (11):1231–1237
48. Bileschi SM (2006) Streetscenes: Towards Scene Understanding in Still Images. PhD thesis, Massachusettes Inst Tech
49. Jahirabadkar S, Kulkarni P (2014) Algorithm to determine ϵ-distance parameter in density based clustering. Expert Syst Appl 41(6):2939–2946
50. Assent I, Krieger R, Müller E, Seidl T (2007) Dusc: Dimensionality unbiased subspace clustering. In: Seventh IEEE international conference on data mining (ICDM 2007). IEEE. pp 409–414
51. Pedregosa F, Weiss R, Brucher M (2011) Scikit-learn: machine learning in Python. J Mach Learn Res 12:2825–2830
52. Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH (2009) The WEKA data mining software: an update. ACM SIGKDD explorations newsletter 11(1):10–18
53. Zhu Jianqing, Liao Shengcai, Lei Zhen, Yi Dong, Li StanZ (2013) Pedestrian attribute classification in surveillance: database and evaluation. In: ICCV workshop on large-scale video search and mining (LSVSM'13). IEEE, Sydney. pp 331–338
54. GitHub repository for SUBSCALE algorithm. https://github.com/amkaur/subscale.git. Accessed 08 Aug 2015
55. GitHub repository for scalable SUBSCALE algorithm. https://github.com/amkaur/subscaleplus.git. Accessed 08 Aug 2015