Knowl Inf Syst (2017) 52:83–111
DOI 10.1007/s10115-016-1013-1

REGULAR PAPER

Synchronization-based scalable subspace clustering of high-dimensional data

Junming Shao[1] · **Xinzuo Wang**[1] · **Qinli Yang**[1] · **Claudia Plant**[2] · **Christian Böhm**[3]

Received: 16 June 2015 / Revised: 11 October 2016 / Accepted: 18 November 2016 / Published online: 2 December 2016

Abstract How to address the challenges of the "curse of dimensionality" and "scalability" in clustering simultaneously? In this paper, we propose arbitrarily oriented synchronized clusters (ORSC), a novel effective and efficient method for subspace clustering inspired by synchronization. Synchronization is a basic phenomenon prevalent in nature, capable of controlling even highly complex processes such as opinion formation in a group. Control of complex processes is achieved by simple operations based on interactions between objects. Relying on the weighted interaction model and iterative dynamic clustering, our approach ORSC (a) naturally detects correlation clusters in arbitrarily oriented subspaces, including arbitrarily shaped nonlinear correlation clusters. Our approach is (b) robust against noise and outliers. In contrast to previous methods, ORSC is (c) easy to parameterize, since there is no need to specify the subspace dimensionality or other difficult parameters. Instead, all interesting subspaces are detected in a fully automatic way. Finally, (d) ORSC outperforms most comparison methods in terms of runtime efficiency and is highly scalable to large and high-dimensional data sets. Extensive experiments have demonstrated the effectiveness and efficiency of our approach.

Keywords Subspace clustering · Synchronization · High-dimensional data · Large data set

1 Introduction

Clustering is an important analysis technique for discovering and understanding the natural structure of a data set. A clustering algorithm determines a grouping of the data set into

✉ Junming Shao
junmshao@uestc.edu.cn

1 School of Computer Science and Engineering, Big Data Research Center, University of Electronic Science and Technology of China, Chengdu 611731, China

2 Institute for Computer Science, University of Vienna, 1090 Vienna, Austria

3 Institute for Computer Science, University of Munich, 80538 Munich, Germany

so-called clusters such that objects from the same cluster are as similar as possible whereas objects assigned to different clusters differ from each other as much as possible. To know this structure is important and valuable because the different clusters often represent different classes of objects which have been unknown previously. Clustering is an essential tool for knowledge discovery in a large variety of applications including biology, medicine, economy, and society. Therefore, clustering has attracted a huge volume of attention, with multiple books, e.g., [20], surveys, and research papers, e.g., [5,38] to mention a few. Thanks to the modern technology advance, nowadays, tremendous amounts of large data sets in a large variety of application domains have been produced. These large real-world data sets are often represented as sparse, high-dimensional feature vectors, contaminated by outliers and noisy objects. Clustering such high-dimensional large data becomes difficult for traditional clustering methods due to two challenging problems: "curse of dimensionality" and "scalability."

To cure the curse of dimensionality in clustering, the new concept of *subspace clustering* has been introduced which automatically detects clusters in subspaces of the original feature space. A number of subspace clustering algorithms have already been proposed, which can be mainly distinguished as axis-parallel subspaces clustering, e.g., [2,4,21], arbitrarily oriented subspace clustering (also called generalized subspace clustering, or correlation clustering), e.g., [3,9,34], and mixed membership subspace clustering [16]. However, most of these algorithms fail to discover clusters of complex shapes and different densities. Let's take CLIQUE [4] for example. This approach detects dense units by using a grid of regular size to divide each dimension into bins. Adjacent dense units are then combined to form clusters. Obtaining meaningful results is dependent on the proper tuning of the grid size and the density threshold parameters. Even after well tuning both parameters, the algorithm will likely fail if at least two clusters with quite different densities are present in the data set. This problem also exists in other approaches, such as density-based subspace clustering SUBCLU [21]. Another challenge for subspace clustering regards the search strategy for the suitable subspaces. The number of possible axis-parallel subspaces is exponential (2^d) in the number of dimensions, d, and the number of arbitrarily oriented subspaces is even infinite. Therefore, a complete enumeration of all possible subspaces to be checked for clusters is not feasible. Consequently, all previous solutions rely on specific assumptions and heuristics, and try to find promising subspaces during the clustering process, for instance in an iterative optimization. We will see that these previous approaches of learning suitable subspaces work well if (but only if) subspace clusters are locally well separated and no outlier objects (belonging to no cluster) exist. In the presence of outliers in the local neighborhood of cluster points or cluster representatives in the entire feature space, most previous subspace clustering algorithms fail to detect subspace clusters, because the algorithms try to find suitable subspaces for each cluster from the local neighborhood of cluster points or cluster representatives in the entire feature space.

Moreover, the growing size of the data sets produced in diverse fields is posing another increasing challenge for most established clustering algorithms. Due to the high time complexity (e.g., $d^2 \cdot n^2 + d^3 \cdot n$ for 4C [9]), most existing algorithms become infeasible for large data sets due to the unacceptable computation time. More scalable subspace clustering should be proposed to handle large-scale data sets.

To deal with these problems, in this paper, we consider subspace clustering from a different point of view, *synchronization*. Synchronization is a prevalent phenomenon in nature. It is known that synchronization is rooted in human life from the metabolic processes in our cells to the highest cognitive tasks we perform as a group of individuals [7]. A paradigmatic example of a synchronization phenomenon in nature is the synchronous flashing

of fireflies observed in South Asian forests. Recently, many synchronization-based models [1,10] and data mining algorithms [10,18,22,30–32] have been proposed and showed many desirable properties. Here, we introduce the concept of synchronization to subspace clustering. The key idea of synchronization-based subspace clustering is to regard each data object as a phase oscillator, the feature vector of an object as its phase, and simulate the dynamical behaviors of the objects over time. Through nonlinear weighted interaction among objects, objects with similar attributes in arbitrarily oriented subspaces tend to synchronize together.

1.1 Contributions

Inherited by the concept of synchronization, our scalable subspace clustering approach, ORSC (arbitrarily ORiented Synchronized Clusters) shows many desirable properties, most importantly:

1. *High-performance clustering* ORSC faithfully captures the natural cluster structure of the large-scale high-dimensional data by dynamic clustering. As the simulated nonlinear object movement follows the intrinsic structure of the data, our ORSC approach reliably detects clusters of arbitrary number, shape, and size in subspaces, even in difficult settings with noise and outliers (Fig. 9; Table 3).
2. *Efficient subspace searching* Thanks to the powerful concept of synchronization, ORSC does not need to scan all possible subspaces and uses an intuitive and efficient strategy: *finding all subspace clusters by searching all synchronized objects instead*.
3. *Scalability* Using entropy-based data partitioning, ORSC is able to cluster large data sets (millions of objects) efficiently in a simple yet effective *divide-and-conquer* fashion. As it turns out, ORSC is not only more accurate than state-of-the-art subspace clustering algorithms (e.g., 4C [9], ORCLUS [3], Curler [34]), but also outperforms these algorithms in terms of scalability on larger data sets (Sect. 6; Figs. 13, 14, 15).
4. *Parametrization* ORSC is easy to parameterize, as there is no need to specify the desirable number of clusters. Moreover, in contrast to existing methods, there is also no need to specify the subspace dimensionality and all interesting subspace clusters are detected automatically.

The remainder of this paper is organized as follows: In Sect. 2, we briefly survey the related work. Section 3 presents our algorithm in detail. Section 4 extends the algorithm to handle large-scale data sets. The relationship and distinction of synchronization-based clustering algorithms are discussed in Sect. 5. Section 6 contains an extensive experimental evaluation. We give a short discussion and finally conclude the paper in Sect. 7.

2 Related work

As most traditional clustering algorithms fail to detect clusters embedded in high-dimensional data, various subspace clustering approaches have been studied [3,4,9,12,15]. Subspace clustering algorithms like CLIQUE [4] and its extensions ENCLUS [11], OptiGrid [17], projected clustering algorithms such as PROCLUS [2], DOC [27], and SUBCLU [21] only find axis-parallel clusters. Pattern-based subspace clustering methods (e.g., [25,36]) aim to group objects that exhibit a similar trend in a subset of attributes into clusters rather than objects with low distance. However, they limit themselves to finding only clusters that represent pairwise positive correlations in the data set. Here, we focus on the more

recent correlation clustering algorithms which can find clusters in arbitrarily oriented subspaces. Recently, many arbitrarily oriented subspace clustering algorithms such as ORCLUS [3], 4C [9], Curler [34] are proposed to find clusters with arbitrarily oriented principle axes.

ORCLUS [3] is one of the iterative top-down search methods for arbitrarily projected subspaces. It integrates PCA into k-means and includes three steps: assign clusters, determine subspaces, and merge them. However, it requires users to specify the number of clusters and the size of the subspace dimensionality in advance. If the estimation does not match with the actual number of clusters, the result of ORCLUS tend to fail. Another problem is that ORCLUS cannot handle the noisy and sparse data efficiently. Moreover, due to using random sampling to improve computation speed and scalability, it may suffer from missing smaller clusters. 4C [9] combines PCA and density-based clustering (DBDSCAN) to identify local subgroups of the data objects sharing a uniform but arbitrarily complex correlation. It formalizes the correlation-connected cluster as a dense region of points in the d-dimensional feature space with at least one principal axis with low variation along this axis. Like ORCLUS, 4C assumes the clustering structure is dense in the entire feature space; thus, it cannot handle sparse data properly. Moreover, both ORCLUS and 4C limit them to identify linear correlation clusters without considering nonlinear correlation clusters. Actually, in real-life data sets, correlation between attributes could however be nonlinear. To address the nonlinear issue, Curler [34] is proposed which allows to detect both global and local orientations of the clusters. This method merges the microclusters generated by the variant of the EM algorithm according to their co-sharing level. Therefore, the resulting clusters may represent a more complex, not necessarily liner correlation. However, like ORCLUS, the performance of Curler is very sensitive to noise and strongly depends on the suitable parametrization.

2.1 Synchronization and models

Recently, synchronization has attracted a large volume of interest in physics, biology, ecology, sociology, communication, and other fields of science and technology. It is known that synchronization is rooted in human life from the metabolic processes in our cells to the highest cognitive tasks we perform as a group of individuals [7]. In general, synchronization means adjustment of phases or frequencies of periodic oscillators due to weak interaction. It is usually used in experimental studies and in the modeling of interaction between different systems demonstrating oscillating behavior. Due to the interaction of two (or more) systems, their states can coincide. Currently, many synchronization-based models have been proposed in diverse fields [1,6,10,18,22,28–32,37]. For instance, Arenas et al. [6] investigate the synchronization phenomena for network analysis, and study the relationship between topological scales and dynamic timescales in complex networks. Kim et al. [22] analyze the cell cycle-specific gene expression to discover the groups of genes by modeling each gene as an oscillator. Aeyels and De Smet [1] introduce a mathematical model for the dynamics of chaos systems. They characterize the data structure by a set of inequalities in the parameters of the model and apply it to a system of interconnected water basins. Böhm et al. [10] propose an extensive Kuromato model to simulate each object's dynamics during the process toward synchronization and discover cluster structure and outliers automatically. In contrast to other algorithms, synchronization-based clustering algorithms have some desirable properties: (a) high-quality clusters detection, (b) noise robustness and (c) a natural hierarchical data analysis.

3 The algorithm ORSC

In this section, we introduce ORSC, to find all possible synchronized clusters in arbitrarily oriented subspaces. Before we can give an overview of the approach, we first illustrate the main underlying concepts of synchronization-based subspace clusters.

3.1 Synchronization: a new viewpoint for subspace clustering

As indicated in Sect. 2, synchronization is a universe concept in our real world. For example, synchronization seems to be a central mechanism for neuronal information processing within a brain area as well as for communication between different brain areas. Results of animal experiments also indicate that synchronization of neuronal activities in the visual cortex appears to be responsible for the binding of different but related visual features so that a visual pattern can be recognized as a whole. Inspired by these synchronization phenomena and models, we consider the subspace analysis from a novel perspective: synchronization. The key point is to view each dimension of one object as an oscillator and move dynamically according to a weighted interaction model (cf. Sect. 3.2). Instead of modeling the object movement based on the physical concept of gravity [8], here the object dynamics is governed by the proposed extended Kuramoto model. Since the phase in traditional synchronization models (e.g., Kuramoto model) is constrained in $[-\pi\ \pi]$, the feature values of objects may be arbitrary in real-world applications. In addition, to ensure object movements driven by its local data structure, the coupling function should be non-decreasing under the phase range. Therefore, the input data sets are first normalized to [0 1] to fit this constraint. Afterward, for each object, we search its similar objects as interaction partners. The strength of interactions from these similar objects is determined by its local structure.

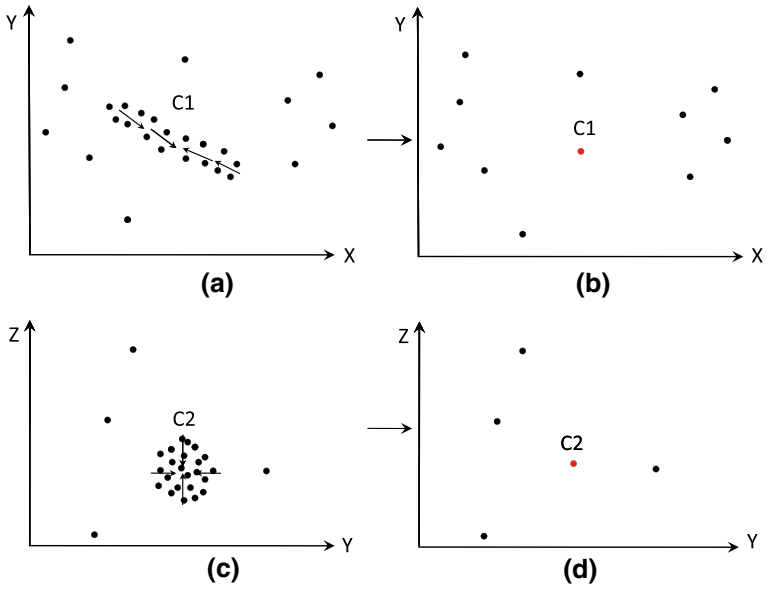


Fig. 1 Illustration of synchronized clusters. **a** Cross section on X–Y axis. **b** Synchronized cluster on X–Y axis. **c** Cross section on Y–Z axis. **d** Synchronized cluster on Y–Z axis

To ensure that all objects of a common arbitrarily oriented subspace cluster can move together, PCA is integrated into the interaction model to determine the main directions of the local cluster structure. Through the weighted interactions, all objects in arbitrarily oriented subspace clusters can easily synchronize together and form synchronized clusters. For illustration, we demonstrate the concept with the help of a simple three-dimensional data set. Figure 1a, c represents the X–Y and Y–Z projection of the data, respectively. In the lower dimensionality on X–Y axis, the cluster $C1$ is embedded in the perpendicular space plane of the direction of the cluster. Meanwhile, the cluster $C2$ exists in the Y–Z axis. To find such clusters, in this context, we define the notion of *an arbitrarily oriented synchronized cluster*, which is a set of objects synchronized together through mutual interaction driven by their local data structure in arbitrarily oriented subspaces. Specifically, like oscillators, each object in a data set interacts with similar objects. Objects will change their states and move to other objects through this nonlinear interaction, which we will elaborate in detail in Sect. 3.2. The arrows in Fig. 1a, c illustrate the main directions of movements for objects. Figure 1b, d further indicates the final states of objects (red points) after synchronization. Finally, it is clear that all correlated objects move together and have the same state in subspaces. These correlated objects are thus formed as synchronized clusters (cf. Fig. 1b, d). To further support large-scale data clustering, an entropy-based data partitioning strategy is employed, which allows ORSC cluster large data sets (millions of objects) efficiently. In the following, we start to elaborate how to construct the weighted interaction model.

3.2 Weighted interaction model

Currently, the most successful way to explore the synchronization phenomena is the Kuramoto Model [23,24], which is motivated by the behavior of systems of chemical and biological oscillators. It is a mathematical model used to describe the dynamics of a large set of phase oscillators by coupling the sine of their phase differences. All frequencies of oscillators should be identical or nearly identical. Formally, the *Kuramoto model* (KM) consists of a population of N coupled phase oscillators, where the phase of the ith unit, denoted by θ_i, evolves in time according to the following dynamics:

$$\frac{\mathrm{d}\theta_i}{\mathrm{d}t} = \omega_i + \frac{K}{N}\sum_{j=1}^{N}\sin(\theta_j - \theta_i), \quad (i = 1, \ldots, N) \tag{1}$$

where ω_i stands for natural frequency of the ith unit and the constant K describes the coupling strength between units. $\sin(\cdot)$ is the coupling function. This model well describes the collective behavior of all coupled phase oscillators toward synchronization, which implies that all the oscillators interact with each other and will synchronize together finally. However, this situation rarely occurs in real-life systems. Local synchronization is observed more frequently, what means a local ensemble of oscillators synchronize together, where the whole set of oscillators is split into several clusters of mutually synchronized oscillators.

Therefore, in order to introduce the Kuramoto model into subspace clustering, we reconsider it in a different way.

1. *Local interaction fashion* To exploit the hidden clusters or patterns in arbitrarily oriented subspaces, the local structure of data should be investigated. Therefore, we focus on the dynamics of objects in a local way.
2. *Weighted interaction* In high-dimensional space, the correlations in the dimensions are often specific to data locality: Some objects are correlated with respect to a given set

of dimensions, and others are correlated with respect to different dimensions. Thus, the coupling strengths of interactions of objects in relevant or irrelevant dimensions should be considered with different weights.

In the following, we will reformulate our interaction model based on the above two criteria.

3.2.1 Local interaction

To formalize the local interaction for each object, the intuitive way is to consider its ε-neighborhood as follows.

Definition 1 (ε-*Neighborhood*) Given $\varepsilon \in \mathcal{R}$ and $x \in \mathcal{D}$, the ε-neighborhood of an object x, denoted by $N_\varepsilon(x)$, is defined as:

$$N_\varepsilon(x) = \{y \in \mathcal{D} | \text{dist}(y, x) \leq \varepsilon\} \tag{2}$$

where $\text{dist}(y, x)$ is a metric distance function and Euclidean distance is used here.

However, such ε-neighborhood search cannot fit our goal well since it does not consider the local data distribution. We intend to look for objects which are close considering the local subspace cluster structure. Therefore, we use the Mahalanobis distance instead of Euclidean distance to determine similar objects.

Definition 2 (ε-*Neighborhood with Mahalanobis distance*) Given $\varepsilon \in \mathcal{R}$ and $x \in \mathcal{D}$, the ε-neighborhood of an object x with Mahalanobis distance, $N_\epsilon^m(x)$, is defined as:

$$N_\varepsilon^m(x) = \left\{y \in \mathcal{D} | \sqrt{(y - x) \cdot \Sigma_x^{-1} \cdot (y - x)^T} \leq \epsilon\right\} \tag{3}$$

where Σ_x is the covariance matrix of ε-neighborhood of x. Since the Mahalanobis distance considers the local data distribution, it can better search similar objects considering the local cluster structure and is also less sensitive to noise. Figure 2 illustrates the similar objects determination of an object P with different distance functions.

According to Definition 2, we extend the Kuramoto model in a local fashion, where each object interacts with its ε-neighborhood with Mahalanobis distance over time. Moreover, since we have no external knowledge of each object, all objects are assumed to have the same frequency ω, which fits the condition of Kuramoto model. Here, we view each dimension of an object as a phase oscillator. Its original value represents the initial phase.

Formally, let $x \in R^d$ be an object in the data set $\mathcal{D}$ and x_i be the ith dimension of the data object x. $N_\varepsilon^m(x)$ is the ε-neighborhood of object x. According to Eq. (1), the dynamics of

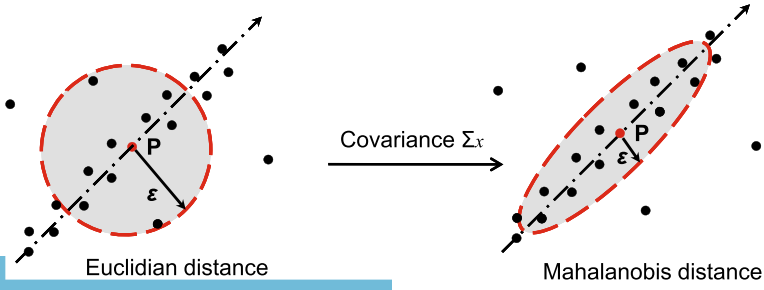


Fig. 2 ϵ-Range search with different distance functions

each dimension x_i of the object x with a local interaction is written as:

$$\frac{\mathrm{d}x_i}{\mathrm{d}t} = \omega + \frac{K}{|N_\varepsilon^m(x)|} \sum_{y \in N_\varepsilon^m(x)} \sin(y_i - x_i) \tag{4}$$

Let $\mathrm{d}t = \Delta t$, then:

$$x_i(t+1) = x_i(t) + \Delta t \cdot \omega + \frac{\Delta t \cdot K}{|N_\varepsilon^m(x(t))|} \cdot \sum_{y(t) \in N_\varepsilon^m(x(t))} \sin(y_i(t) - x_i(t)) \tag{5}$$

Since the term $\Delta t \cdot \omega$ is the same for each dimension of all objects and thus can be ignored. Let $C = \Delta t \cdot K$, the dynamics of each dimension x_i of an object x over time is written as:

$$x_i(t+1) = x_i(t) + \frac{C}{|N_\varepsilon^m(x(t))|} \cdot \sum_{y(t) \in N_\varepsilon^m(x(t))} \sin(y_i(t) - x_i(t)) \tag{6}$$

where $t = (0, \ldots, T)$ is the time step.

3.2.2 Weighted interaction

For most existing interaction models, e.g., [1,6,10,23], the coupling strength of the object interactions is constant. However, this is not appropriate for subspace clustering since clusters exist in different subspaces. Thus, to ensure all cluster objects can synchronize in the corresponding subspaces, we consider the strength of interactions followed by the local data structure of the objects. For an object x, we expect that the interactions along the main directions of the local cluster structure (relevant and potentially correlated dimensions) are imposed much higher weights while those in irrelevant dimensions have lower weights. Therefore, to determine the main directions of the local cluster structure, PCA is used to decompose the covariance matrix Σ of objects $N_\varepsilon^m(x)$, which is denoted by $\Sigma = VEV^T$. The matrix V is called eigenvector matrix and the diagonal matrix E is called eigenvalue matrix. The eigenvectors represent the principal directions of these similar objects, and the eigenvalues represent the variance along these directions. We normalize the eigenvalues into the interval (0, 1) by dividing each eigenvalue λ_i by the sum of all eigenvalues. Then, the normalized eigenvalues are viewed as the interaction weights along with the corresponding principal directions. Finally, we project the difference vector between two objects onto these orthogonal eigenvectors and couple the difference with the corresponding normalized eigenvalues. Formally, the weighted interaction between two objects is defined as follows.

Definition 3 (*Weighted interaction*) Let $x \in R^d$ be an object in the data set $\mathcal{D}$. $N_\varepsilon^m(x(t))$ is the ε-Neighborhood of the object x and $y \in N_\varepsilon^m(x(t))$. $\vec{v}_1, \ldots, \vec{v}_d$ and $\lambda_1, \ldots, \lambda_d$ are the eigenvectors and eigenvalues by PCA decomposition of the covariance matrix of $N_\varepsilon^m(x(t))$. The weighted interaction between the object $y \in N_\varepsilon^m(x(t)$ and the object x, denoted by $\mathrm{WI}_{(y-x)}$, is defined as:

$$\mathrm{WI}(y, x) = \sum_{k=1}^{d} \lambda_i \cdot \sin(\mathrm{proj}(\Delta(y, x), \vec{v}_k)) \tag{7}$$

where $\Delta(y, x) = y - x$ means the difference vector between y and x, $proj(\Delta(y, x), \vec{v}_i)$ means the projection of vector $\Delta(y, x)$ onto $\vec{v}_i$. Since the eigenvectors $\vec{v}_i$ are unit vectors, therefore,

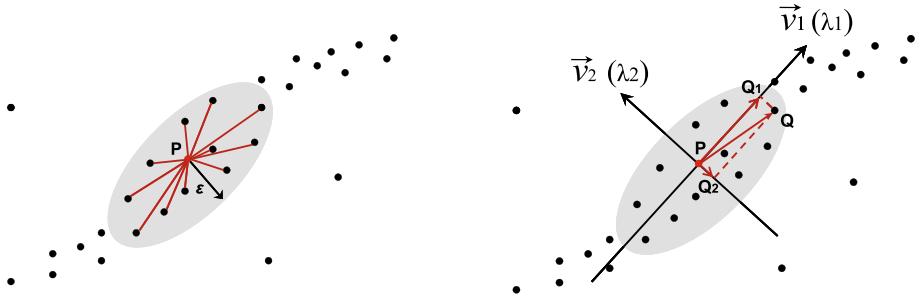


Fig. 3 Weighted interaction between two objects

$$\text{proj}(\Delta(y,x),\vec{v_i}) = (\langle\Delta(y,x),\vec{v_i}\rangle)\cdot\vec{v_i} \tag{8}$$

where $\langle,\rangle$ means the inner product.

To illustrate the weighted interaction, Fig. 3 gives an example with a two-dimensional data set. Given an object P, first, the ε-neighborhood of object P with Mahalanobis distance is obtained. Then, we decompose the covariance matrix of these objects by PCA and obtain the eigenvectors $(\vec{v_1}, \vec{v_2})$ and eigenvalues (λ_1, λ_2) respectively. For each interaction with object P, e.g., $Q-P$ interaction, the difference vector $\overrightarrow{QP}$ is projected to the first direction $\vec{v_1}$ with $\overrightarrow{Q_1P}$ and the second direction $\vec{v_2}$, denoted by $\overrightarrow{Q_2P}$. The interaction between objects Q and P is finally determined with $\lambda_1 \cdot sin(\overrightarrow{Q_1P}) + \lambda_2 \cdot sin(\overrightarrow{Q_2P})$.

Finally, the dynamics of each dimension x_i of the object x is governed by:

$$x_i(t+1) = x_i(t) + \frac{1}{|N_\varepsilon^m(x(t))|}\cdot\sum_{y(t)\in N_\varepsilon^m(x(t))} \cdot\sum_{k=1}^{d}\lambda_k\cdot\sin(\text{proj}_{(i)}(\Delta(x(t),y(t)),\vec{v_k})) \tag{9}$$

where $\text{proj}_{(i)}$ means the ith dimension of the projected vector. The object x at time step $t = 0 : x(0)(x_1(0);\cdots;x_d(0))$ represents the initial state of the object. The $x_i(t+1)$ describes the new state value of ith dimension of object x at time point $(t+1)$.

To determine the termination of the dynamic process, a synchronization-order parameter r is defined as measuring the degree of synchronization of objects.

Definition 4 (*Synchronization-order parameter*) The synchronization-order parameter r characterizing the degree of synchronization is defined as the average movements of objects over time:

$$r = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{1}{|N_\varepsilon(x)|}\sum_{y\in N_\varepsilon(x)}\text{WI}_{(y-x)}\right) \tag{10}$$

The value of r decreases when more and more objects synchronize together as time evolves. The process toward synchronization terminates when r converges, which indicates that there is no further change of objects.

3.3 Simulation of the object dynamics

After formulating our interaction model, we can simulate the dynamics of objects to investigate all clusters in arbitrarily oriented subspaces. Generally, the dynamics of objects involve the following steps:

1. At $t = 0$, all objects in the data set have their own states (feature vectors).
2. As time evolves ($t > 0$), for each object $x(t)$, we search its similar objects with Mahalanobis distance $N_\varepsilon^m(x(t))$ and then apply PCA to decompose the covariance matrix of $N_\varepsilon^m(x(t))$ to obtain the corresponding eigenvectors and eigenvalues. The new state of object $x(t)$ is then determined by Eq. 9.
3. During the process toward synchronization, the order parameter $r(t)$ (Eq. 10) is calculated to terminate the simulation when r converges.

To illustrate the dynamics of objects according to our interaction model, for the ease of illustration, a three-dimensional data set is provided as an example. Figure 4a ($t = 0$) shows the initial states of objects, where three clusters exist in the data set: a three-dimensional Gaussian cluster in full dimensions, a two-dimensional linear correlation cluster in arbitrarily oriented subspace and one-dimensional linear cluster in Z axis. When $t > 0$, for each object, it starts to interact with similar objects according to the local cluster structure. For relevant dimensions, the object interaction is imposed much higher strength while lower impact for irrelevant dimensions. For example, in Fig. 4b–c, the objects in the Gaussian cluster gradually move together from all directions in the three dimensions since these objects belong to a common 3D subspace cluster. We can see that the eigenvalues along with main directions of these objects (eigenvectors) decomposed by PCA are very similar ($\lambda_1 \approx \lambda_2 \approx \lambda_3$, see Fig. 4d). Therefore, the object interactions are imposed similar strengths and gradually group together. This situation is different from objects in the other two clusters. For the objects in the two-dimensional correlation cluster, the main directions of the cluster structure are not

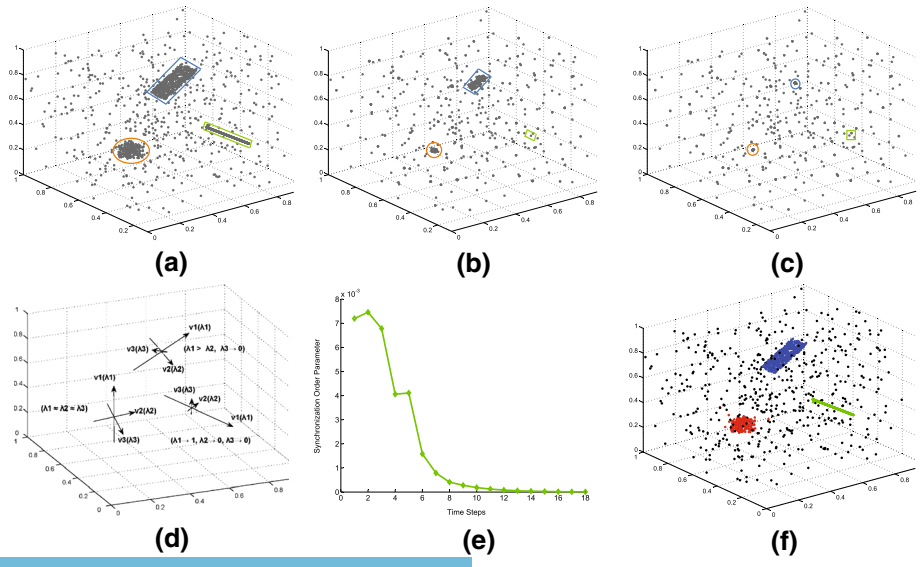


Fig. 4 Illustration of dynamics of objects. **a** t = 0, **b** t = 6, **c** t = 10, **d** main directions, **e** order parameter, **f** synchronized clusters

along with the coordinate axis but arbitrarily oriented. The eigenvalues on the corresponding eigenvectors decomposed by PCA are thus very different, where ($\lambda_1 > \lambda_2$ and $\lambda_3 \to 0$). These cluster objects therefore mainly move toward two directions ($\vec{v_1}$) and ($\vec{v_2}$) with weights λ_1 and λ_2 respectively (Fig. 4d). Similarly, the one-dimensional linear cluster objects tend to move toward one direction ($\vec{v_1}$). Through such weighted interactions, finally, all cluster objects synchronize together in the corresponding subspaces (Fig. 4f). During the process toward synchronization, the synchronization-order parameter will decrease and finally converge (Fig. 4e).

3.4 Synchronized clusters search

After the simulation of dynamics of objects by our interaction model, cluster objects in arbitrarily oriented subspaces synchronize together. To find these synchronized clusters, the intuitive way is to find all synchronized phases and corresponding objects. The principle of our strategy is to consider the subspace search from objects instead of dimensionality. Specifically, for each object, we investigate whether other objects synchronize with it in any dimension. If it does not synchronize with any dimension of other objects (with same phase), this object is viewed as noise. If it synchronizes with some dimensions of other objects, these synchronized dimensions are regarded as a synchronized subspace and these synchronized objects are formed as a synchronized cluster. We extract all synchronized subspaces and assign synchronized objects in corresponding subspaces. We repeat this process for each object, and finally we get all synchronized subspaces and corresponding synchronized clusters.

To illustrate the search strategy, let us look at Table 1. Supposing there are 7 objects with 4 dimensions, after weighted interaction among objects, the final states of objects are outlined in the left part of Table 1. For each object, we find its synchronized dimensions and objects. For example, object #1 synchronizes with objects #2 to #4 in dimensions of 1 and 2. Therefore, a synchronized subspace (1,2) is created and the corresponding objects #1 to #4 are added to a cluster in this subspace. Similarly, object #2 synchronizes with objects #3 and #4 in dimensions (1,2,4) and a new subspace is thus created. This process is repeated until all objects are investigated. In addition, since object #7 does not synchronize with any other object in any dimension, it is viewed as noise in the full-dimensional space.

Once we detect all synchronized clusters, we can further determine their dimensionality. The reason is that objects in synchronized clusters moving together do not need to be parallel with axis but arbitrarily orientated of the synchronized cluster objects. Therefore, if a cluster is located in an axis-parallel subspace, the synchronized subspace is the exact subspace.

Table 1 Illustration of the subspace search strategy

Obj.	$d1$	$d2$	$d3$	$d4$	Syn. dim.	New subs.	Cluster
1	0.1	0.2	0.1	0.3	1, 2	(1, 2)	(1 2 3 4)
2	0.1	0.2	0.2	0.2	1, 2, 4	(1, 2, 4)	(2 3 4)
3	0.1	0.2	0.7	0.2	1, 2, 3, 4	(1, 2, 3, 4)	(3, 4)
4	0.1	0.2	0.7	0.2	1, 2, 3, 4	–	–
5	0.3	0.4	0.3	0.3	3	(3)	(5, 6)
6	0.9	0.5	0.3	0.1	3	–	–
7	0.7	0.6	0.4	0.5	Null	–	Noise

```
algorithm [S, C] = ORSC(D, ε)
    D' = Simulation(D, ε);
    [S, C] = SearchCluster(D')
Return S, C;

//Objects' dynamics Simulation.
Function D' =Simulation(D, ε);
    while(loopFlag=true)
      for(each object x ∈ D)
        Search N_ε(x) of object x;
        Compute N_ε^m(x) by Definition 2 in N_ε(x);
        Determine the interaction directions and weights by PCA;
        Obtain new state of object x with interaction model (Eq.9);
      end for
      Set D' to be the new states of all objects;
      Compute synchronization order parameter r;
      if( r converges)
        loopFlag=false;
      end if
    end while
return D';

//Search Synchronized Clusters.
Function [S, C]=SearchCluster(D');
    S = null; C = null; //S saves subspaces and C saves clusters
    for(each object x ∈ D')
      for(each object y ∈ D')
        If y synchronized with x in dimensions L;
          If (L exists in S) and synchronized phases are same
            Add y to the exist corresponding cluster;
          Else
            Create a new synchronized subspace L;
            Create a new cluster CL;
            Add x and y to the new cluster CL;
            S.add(L), C.add(CL);
          End If
        Else
          y is viewed as noise object;
      End For
    End For
return S, C;
```

Fig. 5 Pseudocode of the ORSC algorithm

However, if a cluster is embedded in an arbitrarily oriented subspace, the synchronized subspace only stands for the original feature space that these clusters accommodated.

Definition 5 (*Dimensionality of a synchronized cluster*) Let $\mathcal{S} \subseteq \mathcal{D}$ be a synchronized cluster and $\Sigma = VEV^T, V = \lambda_1, \ldots, \lambda_d$ be the eigenvalues of $\mathcal{S}$ in descending order. Given a $\delta \approx 0$, the dimensionality of $\mathcal{S}$ *w.r.t* δ is η if $d - \eta$ eigenvalues of $\mathcal{S}$ are close to zero ($\Phi(\lambda_i) \leq \delta$ and $\Phi(\lambda_i) = \lambda_i/\lambda_1$), which is denoted by $\text{DimSynClu}_{\delta}^{\eta}$.

Finally, the Pseudocode of the ORSC Algorithm is illustrated in Fig. 5.

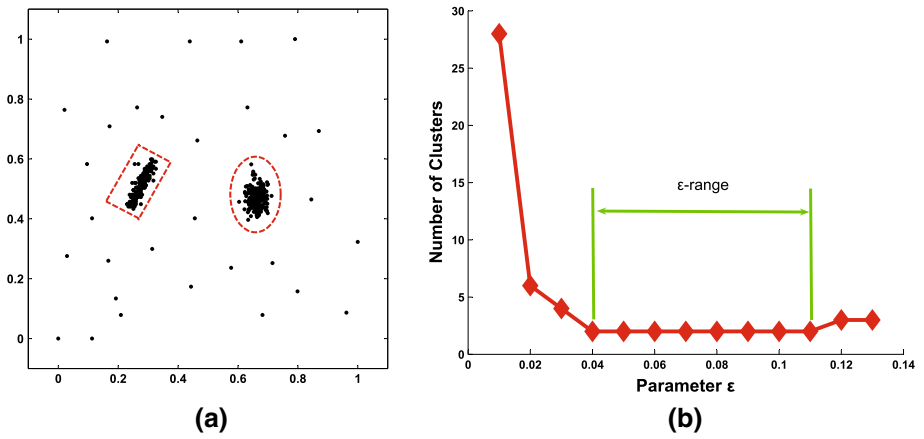


Fig. 6 Impact of parameter setting for clustering

3.5 Parameter setting

To simulate the dynamics of objects, an interaction range (ε) needs to be specified in our interaction model. The question is: How to determine the ε value and how does the clustering results change when the ε value is adjusted? In order to generate a stable interaction among objects, a heuristic way is to use the average value of the k-nearest neighbor distance determined by a sample from the data set for a small k. Figure 6a shows a simple data set which consists of 2 clusters plus outliers. We start with a very small value and then gradually increase this value to see the change of clustering results. With different values for parameter ε, we compute the number of clusters which are detected. We can see that the same clustering results (2 clusters) are obtained with a fairly long stable range (see Fig. 6b). In contrast to other subspace clustering algorithms, the parameter of our algorithm is much more flexible and more robust to noise. The reason is that our clustering is a dynamic process, where each object moves during the process toward synchronization. With a small interaction range, in the beginning, a few objects interact with each other. But after several time steps, these objects in a cluster will gradually move closer and thus more and more objects can interact with each other and finally synchronize together. For relative larger interaction range, the only difference is that more objects interact with each other at the beginning and thus tend to synchronize much faster. For other established subspace clustering algorithms, such as ORCLUS [3], it is required to specify the dimensionality of clusters and also the number of clusters in this embedded subspaces, which are usually unknown beforehand. This problem usually exists in other popular algorithms, such as 4C and Curler. For ORSC, after dynamic clustering, all objects in a cluster in arbitrary subspace will synchronize together via the weighted nonlinear interaction. Afterward, we can search all embedded potential clusters by simply search all synchronized phases and corresponding objects (cf. Sect. 3.4).

4 Scaling up synchronization-based subspace clustering to massive data

The proposed synchronization-based subspace clustering allows for an efficient subspace search; however, like most established subspace clustering algorithms, it still does not scale

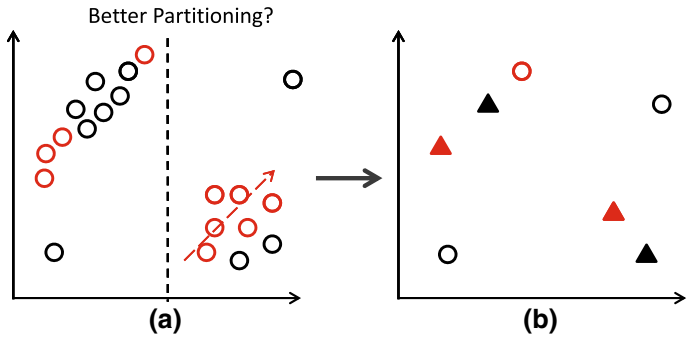


Fig. 7 Synchronization-based subspace clustering by random partitioning. **a** Random data partitioning. **b** Synchronization-based data representation

well with the number of observations (i.e., $O(N^2)$). To uncover the embedded subspace clusters on large data sets containing millions of objects, a simple yet effective strategy: *divide-and-conquer*, has been further applied in this study. With this simple strategy, the subspace clustering on large-scale data is replaced by clustering on each part separately. We will demonstrate that the *divide-and-conquer* concept with entropy-based data partitioning fits to synchronization-based subspace clustering well due to its unique local and dynamic clustering fashion.

For illustration, Fig. 7 provides a simple example, where objects with red color represent one data partition while objects with black color indicate another partition. By dynamic clustering, partition one (red color) is represented by two new synchronized objects (red triangle markers) and one temporary outlier object (red circle). Similarly, partition two (black color) is replaced by two new synchronized objects (black triangles) and two outlier objects (black circles). Finally, the new data set has four new synchronized objects and three outlier objects (Fig. 7b). It is intuitive to observe that the global cluster structure of the original data set is preserved in the new generated data set. By clustering on the new data set, we can finally identify two clusters and two outlier objects successfully. The desirable property of local structure preservation is due to the salient features of our synchronization-based clustering, whose cluster formation is driven by the *local data structure*, and all objects in a cluster dynamically *move together* (cf. Fig. 4). However, for synchronization-based subspace clustering, the interaction directions are dependent on the local data structure (cf. Sect. 3.2.2). If the partitioning largely destroys the local data topology, the interaction directions and weights tend to be different to those on whole data set. Although the changes are happening in local way, the quality of clustering may still be affected. Therefore, to alleviate the potential effects, an entropy-based data partitioning strategy is proposed.

4.1 Entropy-based data partitioning

To efficiently uncover embedded clusters in any subspaces building upon the concept of *divide-and-conquer*, we expect to have a good data partitioning which has no or little effect on clustering. Hence, we apply a heuristic way by *information entropy*, a measure for the irregularity of information content. The basic idea is to compute the entropy for the data of each dimension to determine whether this dimension is suitable for partitioning or not. If the data in one dimension are more randomly distributed (e.g., uniform distribution), it tends to be more difficult to yield a good partitioning. As the entropy measures the irregularity of

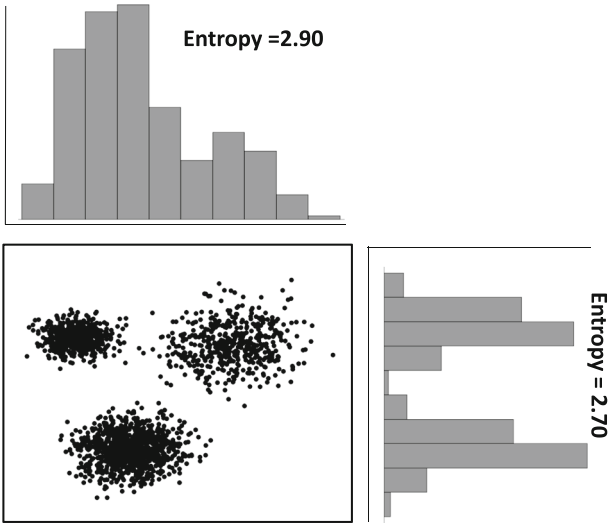


Fig. 8 Illustration of the selection of partitioning dimensions

data, the entropy will be higher if the data are more uniformly distributed. Namely, the higher the entropy of a certain dimension, the worse it is for data partitioning as it will destroy the data locality. Take Fig. 8 as an example. Here, we have a two-dimensional data consisting of three clusters. By observing the data distribution for each dimension, it is intuitive to see that the data locality is much better preserved by splitting the data along the dimension of lower entropy. Formally, we compute the entropy (Shannon entropy) for the data on each dimension as follows.

$$\text{entropy}(D_j) = -\sum_{i}^{m} p(x_i) \log_2 p(x_i), \tag{11}$$

where D_j indicate the jth dimensional data, m is the number of bins for constructing a histogram on this dimension, and $m = 10$ in this study, $p(x_i)$ is the probability of the data points to fall into the ith bin.

Generally, the entropy-based partitioning involves the following stages. First, we compute the entropy for the data on each dimension according to Eq. 11. Afterward, the entropy values for all dimensions are ordered, and the dimensions with low entropy values will be selected to split the data into two parts at each time. If the data set is still too large, the process is repeated for each dimension again until each data partition is small enough for clustering. Using entropy-based data partitioning, the data locality is more likely to be maintained and therefore the quality of clustering is preserved.

4.2 Iterative dynamic clustering

Building upon the concept of *divide-and-conquer* and entropy-based data partitioning strategy, ORSC allows handling large-scale data by partitioning them into thousands of subsets and clustering each separately. To distinguish this approach from the original ORSC algorithm, we refer to it as Parallel ORSC (PORSC). By performing subspace clustering on each subset, the data can be further represented by a set of synchronized objects, and each of which belongs to a specific subspace. New data sets can be generated by combining all

the synchronized objects which are hosted in the same subspace. Specifically, the PORSC algorithm involves the following steps.

1. *Data partitioning* The original data set is split into many small subsets (with fixated size, i.e., $pSize$) based on the strategy of entropy-based data partitioning.
2. *Dynamic clustering* Simulate the object dynamics according to the weighted interaction, and finally the synchronized clusters embedded in any subspace can be identified (cf. Sect. 3).
3. *New data generation* Collect all synchronized objects in every subspace from all subsets to form new data sets (each subspace corresponds to a new data set). Here, each subspace will host its own objects respectively.
4. *Final clustering* We perform the final clustering on each new data set. Here, only the clusters in the full embedded subspace are investigated.

4.3 Complexity analysis

For the simulation of the dynamical process of objects at each time step, we need to search the ε-neighborhood of each object with Mahalanobis distance. The covariance matrix with Euclidean distance computation can be evaluated in $O(N \cdot d)$ time. The covariance matrix is then used to calculate the Mahalanobis distance, and the time complexity approximately requires $O(d^3)$ time. We further use PCA to decompose the covariance matrix with Mahalanobis distance, and it thus requires $O(d^3)$ time. For the interaction of each object, it requires $O(d^3)$ time. Therefore, for all objects together, the simulation of dynamics at one time step is $O(N^2 \cdot d + N \cdot d^3)$. If there exists an efficient index, the complexity reduces to $O(T \cdot N \log N \cdot d + \log N \cdot d)$. For all time steps toward synchronization, the time complexity of objects' simulation is $O(T \cdot (N^2 \cdot d + N \cdot d^3))$ in worst case. T is the number of time steps. In most cases, T is small with $5 \leq T \leq 20$.

For the time complexity of subspaces and the corresponding clusters search, the most bottom-up establishing algorithms of subspace clustering need $O(2^d)$ time. For our search, we do not need iterate all subspaces but finding all synchronized subspaces by investigating the synchronized dimensions of each object. Therefore, the time complexity reduces to $O(N^2 \cdot d)$. Finally, the time complexity of our algorithm in worst case is $O(T \cdot (N^2 \cdot d + N \cdot d^3) + N^2 \cdot d)$.

Relying on the entropy-based data partitioning, the running time is significantly reduced due to the synchronization-based data representation and the *divide-and-conquer* strategy. The dynamic clustering for each partitioning data is $O(T \cdot (N_p^2 \cdot d + N_p \cdot d^3) + N_p^2 \cdot d)$, where N_p is the fixed size number of objects for each partition data and $N_p \ll N$. The time complexity for all partitions is thus $O(p_l \cdot T \cdot (N_p^2 \cdot d + N_p \cdot d^3) + N_p^2 \cdot d)$, where p_l is the number of partitions.

5 Relationship to other clustering paradigms

5.1 Synchronization-based clustering algorithms

Following the common concept, several synchronization-based clustering algorithms [10, 18,22,30,33,37] have been introduced recently. The key idea of clustering approaches by synchronization (e.g., in *Sync* [10]) is to view each data object as a phase oscillator, the feature vector of an object as its phase, and simulate the dynamical behaviors of the objects over time. By the interaction with similar objects, the phase of an object gradually aligns

with its neighborhood, resulting in a nonlinear object movement driven by the local cluster structure. Finally, the objects in a cluster are synchronized together and have the same phase. These algorithms have demonstrated attractive properties compared to many existing clustering algorithms. Beyond that, synchronization-based clustering also allows for a natural hierarchical analysis [29] and handling large data sets via simple yet effective *divide-and-conquer* strategy [30]. All these algorithms aim at discovering the cluster structure in the full-dimensional data space. The proposed algorithm, ORSC, aims at identifying clusters in subspaces of high-dimensional large-scale data sets, which is a very difficult task for existing synchronization-based clustering algorithms. For this purpose, we introduce a new model to support weighted interaction depending on the feature relevance. Beyond that, the parallel version of ORSC is further introduced to support scalable subspace clustering. Therefore, equipped with the benefits of traditional synchronization-based clustering and its scalability, ORSC shows its superiority over the state-of-the-art algorithms in both effectiveness and efficiency aspects (cf. Sect. 6).

5.2 Other dynamic clustering algorithms

For synchronization-based clustering algorithms, one salient feature is that cluster formation is driven by the local data structure, and all objects in a cluster dynamically move together. The idea is related to affinity propagation algorithms [13,14], which take as input measures of similarity between pairs of data points and simultaneously consider all data points as potential exemplars. Real-valued messages are exchanged between data points until a high-quality set of exemplars and corresponding clusters gradually emerges. Inspired by the physical phenomena, gravitation-based clustering algorithms [8,19,26] have been proposed which simulate how particles (data points) group together driven by the gravity force. All these algorithms share a similar schema: exchange values between similar data objects based on a given interaction model. Synchronization-based clustering algorithms are distinguished from other dynamic clustering algorithms by their physical mechanism and the corresponding interaction model. The potential advantages synchronization-based clustering algorithms are twofolds: (1) the dynamics of objects are driven by the local topological structure; therefore, the derived clusters allow reflecting its intrinsic cluster structure. Compared to state-of-the-art algorithms, synchronization-based clustering algorithms tend to find arbitrarily shaped clusters with arbitrary size and (2) Local data structural preservation. Unlike other algorithms, the clustering results by synchronization-based can be further clustered as they well preserve the local data structure [30]. It thus supports iterative clustering and allows performing clustering on large data sets.

6 Experimental evaluation

To extensively study the performance of ORSC and the fast implementation PORSC, we perform experiments on several synthetic and real-world data sets. We compare the performance of ORSC to synchronization-based clustering algorithm Sync [10], the dynamic clustering algorithm Affinity Propagation [13], and the subspace clustering algorithms: ORCLUS [3], 4C [9], and Curler [34]. We select these particular algorithms because they are representatives of different algorithmic paradigms: ORCLUS is a K-means style iterative partitioning algorithm; 4C is a local density-based method; Curler tries to find nonlinear correlation clusters based on EM clustering. We implemented ORSC and PORSC in Java. The source code of Curler is obtained from the authors, and the Java source codes for other comparison algo-

rithms are available in the ELKI package (http://elki.dbs.ifi.lmu.de/). All experiments have been performed on a workstation with 2.4 GHz CPU and 32 GB RAM. Beyond that, we also evaluate the performance of the fast implementation of ORSC (called PORSC in this paper) and compare it with the original ORSC algorithm on both effectiveness and efficiency.

Moreover, we report two established measures for cluster quality [35]: normalized mutual information (NMI), and adjusted mutual information (AMI) to evaluate different clustering results. For both measures, higher values represent better clustering performance. We also provide precision (P) and recall (R) as validity measures to analyze each individual cluster.

6.1 Effectiveness

6.1.1 Synthetic data

We start the evaluation of ORSC with several synthetic data sets to facilitate presentation and demonstrate its benefits. Figure 9 displays the clustering results on a three-dimensional synthetic data with all comparing subspace clustering algorithms. The data consist of 11 clusters of different dimensionality, object density, and correlation strength plus noise (Fig. 9a). For ORSC with parameter $\varepsilon = 0.06$, it successfully detects all these correlation clusters, including five one-dimensional linear correlation clusters, two two-dimensional linear

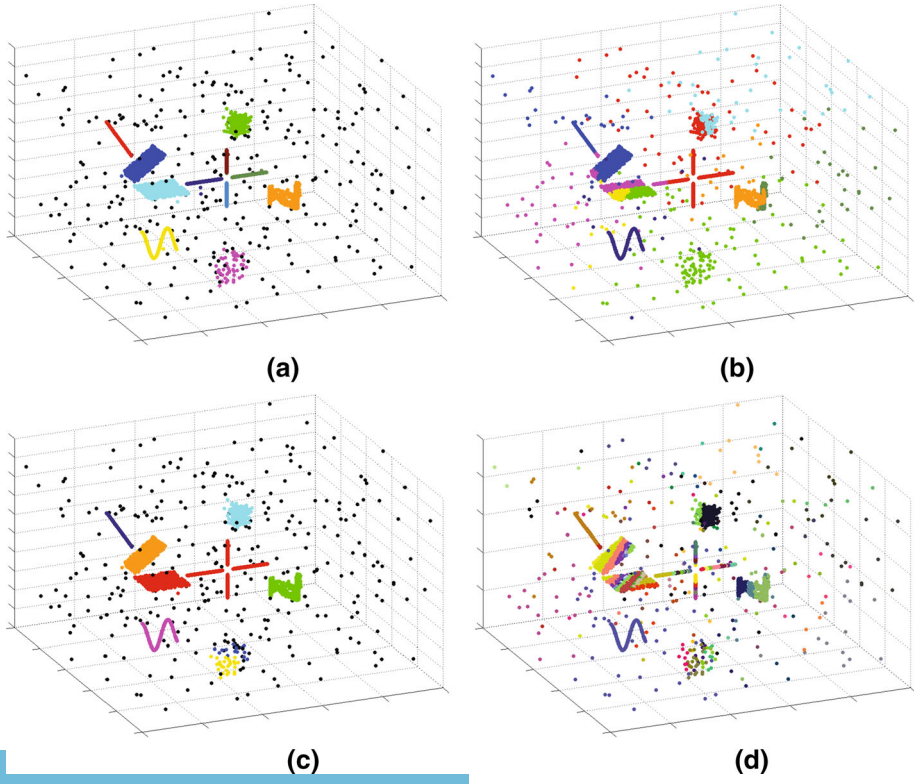


Fig. 9 Comparison with different algorithms on a 3D synthetic data. **a** ORSC, **b** ORCLUS, **c** 4C, **d** Curler

plane clusters, two two-dimensional nonlinear clusters, and two three-dimensional clusters (Fig. 9a). ORCLUS requires the number of cluster K and the average subspace dimensionality l as input parameter. We specify $K = 11$ and obtain the best results with $l = 3$, which are indicated in Fig. 9b. Figure 9c displays the best clustering results of 4C with parameters $\epsilon = 0.03$, MinPts $= 6$ and $\lambda = 3$. For Curler, we use all default parameter values which are suggested by authors. It obtains as many as 150 clusters (Fig. 9d). For better investigating different clustering results, we focus on the detailed view of the 2D linear plane cluster and four 1D linear clusters in the center of the data set (cf. Fig. 10). It is obvious that ORSC outperforms the competitors, where all correlation clusters are successfully detected with high precision and recall. The evaluation of the clustering results is further illustrated in Table 4.

To further evaluate our algorithm ORSC, we generate five high-dimensional data sets. In each data set, several clusters are hidden in subspaces of varying dimensionality plus noise. Detailed description of the five data sets is shown in Table 2. For comparison, we check whether each algorithm can detect these clusters with suitable parameters. We report its precision and recall for each cluster. The results with different subspace clustering algorithms are depicted in Table 3. In all these data sets, ORSC finds the synthetic clusters in corresponding subspaces with both high recall and precision. In contrast to the comparing algorithms, there is no need to specify the subspace dimensionality and all interesting subspace clusters are detected. For 4C and ORCLUS, we manually specify the subspace dimensionality although

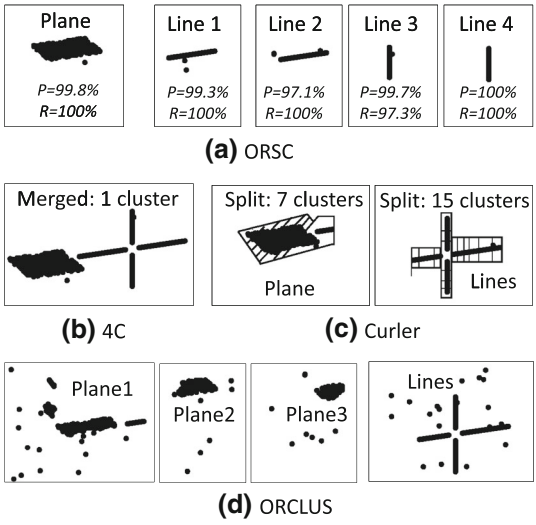


Fig. 10 Detailed view of clustering results with different algorithms on part of 3D synthetic data

Table 2 Detailed information of high-dimensional synthetic data sets

	Dimensionality	Number of clusters	Dimension of each cluster
DS1	5	1	3
DS2	10	1	5
DS3	15	2	10, 5
DS4	20	2	10, 10
DS5	30	3	20, 15, 10

Table 3 Comparative evaluation of different subspace approaches on high-dimensional synthetic data sets

Data	True clusters found by				
	ORCLUS	4C	Curler	ORSC	PORSC
DS1	1 (Dim.: 3)	1 (Dim.: 3)	1 (Dim.: 3)	1 (Dim.: 3)	1 (Dim.: 3)
	($P = 32.7\%$; $R = 91.2\%$)	($P = 100\%$; $R = 100\%$)	($P = 99.0\%$; $R = 20.4\%$)	($P = 100\%$; $R = 97.0\%$)	($P = 99.6\%$; $R = 99.4\%$)
DS2	1 (Dim.: 5)	1 (Dim.: 5)	1 (Dim.: 5)	1 (Dim.: 5)	1 (Dim.: 5)
	($P = 27.8\%$; $R = 62.2\%$)	($P = 100\%$; $R = 94.2\%$)	($P = 48.4\%$; $R = 11.8\%$)	($P = 100\%$; $R = 97.4\%$)	($P = 100\%$; $R = 98.2\%$)
DS3	2 (Dim.: 10,5)	1 (Dim.: 10)	2 (Dim.: 10,5)	2 (Dim.: 10,5)	2 (Dim.: 10,5)
	($P = 16.9$, 74.4%)	($P = 100\%$, $R = 99.6\%$)	($P = 14.5$, 12.0%)	($P = 100$, 99.6%)	($P = 100$, 100%)
	($R = 14.0$, 61.6%)		($R = 19.3$, 16.0%)	($R = 99.6$, 100%)	($R = 100$, 100%)
DS4	2 (Dim.: 10,10)	2 (Dim.: 10,10)	2 (Dim.: 10,10)	2 (Dim.: 10,10)	2 (Dim.: 10,10)
	($P = 17.9$, 100%)	($P = 100$, 100%)	($P = 12.5$, 100%)	($P = 100$, 99.6%)	($P = 100$, 99.6%)
	($R = 13.2$, 74.0%)	($R = 100$, 100%)	($R = 12.5$, 100%)	($R = 100$, 99.6%)	($R = 100$, 100%)
DS5	3 (Dim.: 20,15,10)	1 (Dim.: 20)	3 (Dim.: 20,15,20)	3 (Dim.: 20,15,10)	3 (Dim.: 20,15,10)
	($P = 21.7$, 12.3, 13.2%)	($P = 100\%$; $R = 98.5\%$)	($P = 100$, 99.5, 76.9%)	($P = 100$, 98.5, 99.6%)	($P = 100$, 100, 100%)
	($R = 100$, 67.5, 72.5%)		($R = 99.0$, 100, 5%)	($R = 100$, 99.5, 100%)	($R = 98.5$, 100, 100%)

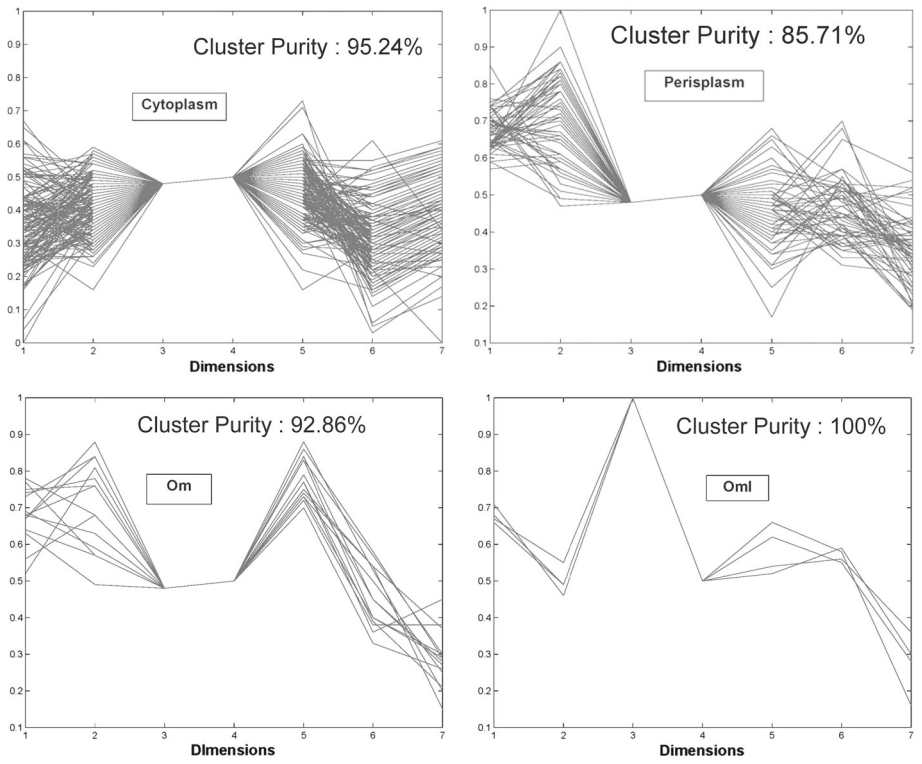


Fig. 11 Clusters found by ORSC on the Ecoli data set ($\varepsilon = 0.25$)

Table 4 Evaluation on different data sets

Methods	Synthetic data		Ecoli data		Wine data	
	NMI	AMI	NMI	AMI	NMI	AMI
ORSC	0.981	0.980	0.682	0.670	0.701	0.695
PORSC	0.943	0.943	0.672	0.657	0.686	0.680
Sync	0.943	0.943	0.640	0.629	0.696	0.691
AP	0.674	0.665	0.546	0.539	0.431	0.425
4C	0.830	0.829	0.338	0.328	0.474	0.469
ORCLUS	0.598	0.596	0.452	0.430	0.191	0.182
Curler	0.583	0.561	0.060	0.049	0	0

it is difficult to know in real world. 4C performs very well on the first two low-dimensional data sets. But it fails to find five-dimensional cluster in 15-dimensional data set. This is also the same for the fifth data set, where it cannot find the embedded 10-dimensional cluster and 15-dimensional cluster because both of them are not dense enough. The algorithm of ORCLUS is sensitive to noise, and most noise objects are included in the detected clusters. In addition, ORCLUS tends to merge these subspace clusters together resulting in low precision. Similarly, the algorithm Curler is sensitive to noise and cannot yield good results for all data sets. However, in contrast to ORCLUS, it tends to split true clusters into several distinct clusters and the obtained clusters usually lack in recall.

Table 5 ORSC clustering results on wine data

Cluster ID	Type 1	Type 2	Type 3	Precision (%)	Recall (%)
1	58	3	0	95.2	98.3
2	0	53	0	100	73.6
3	0	5	48	90.6	100
4	0	4	0	100	5.6

6.1.2 Small real-world data sets

In the following, we evaluate the performance of ORSC on several small-to-large real-world data sets which are publicly available at the UCI machine learning repository.

Ecoli data This Ecoli data deriving from a study on protein location consists of 336 instances. It includes eight classes with are highly unbalanced having from 2 to 142 objects per class. Each instance is described by 7 attributes. ORSC detects 6 meaningful clusters for this data set which results in an NMI of 0.682. Each cluster is mainly represented as one type, see Fig. 11 in detail. Cluster 1 is composed of 147 instances and 140 out of them belong to cytoplasm, $P = 95.24\%$ and $R = 97.90\%$. Cluster 2 includes 56 instances and mainly represents periplasm with $P = 85.71\%$ and $R = 92.31\%$. The type of inner membrane without signal sequence is identified by cluster 3 and cluster 4 together. Similarly, the cluster 5 and cluster 6 represent the type outer membrane and outer membrane lipoprotein with high precision 92.86 and 100% recall, respectively. Moreover, with the entropy-based data partition, the fast implementation PORSC yields similar clustering results with an interaction range $\varepsilon = 0.3$ and pSize $= 200$ even on the small data set, which results in a good performance of NMI $= 0.672$. The slight decrease of clustering performance is due to the fact that the local data structure is slightly affected by the partition procedure, although the entropy-based strategy is provided to alleviate this effect. However, the results (cf. Table 4) demonstrate that the performance of PORSC is very close to ORSC, and the running time decreases significantly (cf. Sect. 6.1.3). For Sync, although it derives from the same concept, its performance is worse than ORSC. The main reason is that ORSC allows characterizing the local data structure for interaction in a better way. Affinity propagation produces the best result (NMI $= 0.546$) when we specify the corrected number of clusters manually. For 4C algorithm, it obtains best results with parameters MinPts $= 6$, $\epsilon = 0.15$ and $\lambda = 7$, which results NMI $= 0.338$. It detects 6 clusters, but many instances are wrongly clustered. ORCLUS obtains much better results (NMI $= 0.452$) with parameters $k = 8$ and $l = 7$. However, like 4C, many instances are wrongly assigned. The algorithm of Curler has a bad clustering result with NMI $= 0.060$, which cannot predict the protein location effectively. The evaluation of the clustering results is further illustrated in Table 4.

Wine data The well-known wine data set is the result of a chemical analysis of wine grown in the same region in Italy but deriving from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wine. The class distribution is as follows: Type1: 59; Type2: 71, Type3: 48. Table 5 gives the clustering results of ORSC with parameter $\varepsilon = 0.6$. It detects 4 clusters and 8 instances are viewed as noise. Three main detected clusters match with corresponding wine types with high precision and recall. In detail, Cluster 1 includes nearly all instances (58 out of 59 instances) of Type 1 plus 3

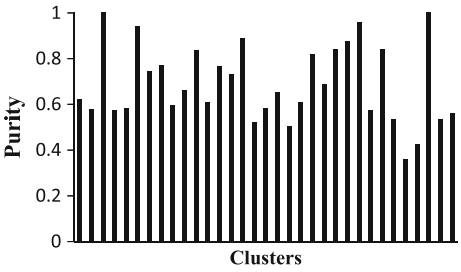


Fig. 12 Clusters found by parallel ORSC on the cover type data set ($\varepsilon = 0.7$)

instances of Type 2. Fifty-three instances in Cluster 2 completely belong to Type 2. Cluster 3 consists of all instances of Type 3. In addition, the Cluster 4 includes four instances of Type 2. It is clear that ORSC can discover interesting patterns of the data set effectively. Similarly, PORSC detects four clusters with high value of recall and precision, and 19 instances are identified as outliers, resulting in NMI = 0.686 and Pur. = 0.955. With the same concept, Sync finds three clusters with high value of NMI = 0.696. For affinity propagation, it produces three clusters while the quality of clusterings are not promising (NMI = 0.431). For 4C with parameter MinPts = 6, $\epsilon = 0.5$ and $\lambda = 13$, it discovers 2 clusters and 34 instances are regarded as noise (NMI = 0.474). As many instances of the two clusters are wrongly clustered, 4C is difficult to distinguish the three types of wine. The algorithm of ORCLUS cannot obtain good clustering results (NMI = 0.191) although we manually specify the number of clusters with parameters $k = 3$ and $l = 13$. For Curler, it even cannot find any correlation cluster for the data set with different parameters. All instances are regarded as one single cluster. The evaluation of these clustering results is further indicated in Table 4.

6.1.3 Large-scale real-world data sets

Beyond the small real data sets, we further evaluate the performance of Parallel ORSC on several large-scale real-world data sets which are publicly available at the UCI machine learning repository.

Cover type data The Covtype data set containing 581,012 instances describes seven forest cover types on a 3030 meter grid with 54 different geographic measurements. In contrast to existing subspace clustering algorithms, Parallel ORSC allows handling large-scale data sets. Here, 33 clusters are found and 9344 instances are viewed as noise. The detailed information of each single cluster is further demonstrated in Fig. 12. We can observe that our method allows identifying the high-quality clusters.

Shuttle The shuttle data set contains 58,000 instances, and each instance has nine attributes. All the instances are labeled into seven classes, most of the them belong to the first class (Rad Flow), which contains 45,586 instances. Parallel ORSC identifies 20 meaningful clusters with NMI = 0.365 and Pur = 0.843. One hundred twenty-nine instances are identified as outliers. The largest cluster found by Parallel ORSC contains 48117 instances which mainly belong to the first class, with a high precision and recall of 81.3, 85.8%, respectively. Among the identified 19 clusters, 14 of them are found with perfect purity, which means all the clusters are rightly identified. Only two of the clusters are reported with relatively lower purity, because some instances are wrongly clustered to other classes (i.e., cluster 2 contains only 2 instances which identified as the class (Bpv Open) and (Rad Flow), respectively).

Table 6 Performances on real data sets

Data	#Obj	#dim	#Cluster	Parallel ORSC	
				NMI	Pur
Shuttle	58000	9	7	0.365	0.843
Covtype	581012	54	7	0.158	0.590
Network intrusion	4898431	34	23	0.839	0.990

Network intrusion The network intrusion data has 4,898,431 instances recording 2 weeks TCP dump data for a local area network simulating a true Air Force environment. Each instance has 42 attributes and 32 numerical attributes of them are chosen for clustering. All of the instances belong to 23 classes, while most of them concentrate on three main classes. Parallel ORSC finds eight clusters with high quality (e.g., in terms of NMI = 0.839 and Pur = 0.991). Specifically, seven clusters have perfect purities of values of 1 or very close to 1 (i.e., 0.9787, 0.9999 and 0.9989, respectively), and the worst cluster still has a purity of 0.902.

The detailed results of Parallel ORSC in terms of different evaluation measures on all real sets are shown in Table 6. From the table, we can see that Parallel ORSC not only allows for an efficient clustering, but also preserving the properties of ORSC.

6.2 Efficiency

The efficiency of our algorithm is evaluated by comparing the runtime of both ORSC and the Parallel ORSC (PORSC) with state-of-the-art algorithms. Results show that ORSC is the fastest when dealing with large and high-dimensional data sets.

Comparison with state-of-the-art algorithms Fig. 13a shows the results of runtime experiments for a 3D synthetic data set varying numbers of objects in the range of 500 to 30,000. ORCLUS and ORSC scale nearly linear against the size of the data set, while 4C scales quadratically. The runtime of Curler is rather high for large number of objects, and thus, only the runtime for maximal 10,000 objects is displayed. For comparison of the scalability in the dimensionality d, data sets consisting of 2000 objects with dimensionality ranging from 5 to 50 are generated. Figure 13b displays the results of runtime against dimensionality. Since all algorithms are involved with PCA, they scale similar with the dimensionality. ORCLUS is the fastest approach, but its effectiveness is not satisfying (cf. Sect. 6.1) and ORSC outperforms 4C and Curler. In summary, ORSC scales very well against the number of objects as well as dimensionality.

Scaling up experiments In this section, we further evaluate the speedup of the fast ORSC against original ORSC to demonstrate its scalability.

To assess the scalability of PORSC with respect to database size, we first generate five synthetic data sets with different database sizes from one thousand objects to ten million objects). Figure 14a shows the runtime of PORSC with respect to different database sizes. It indicates that the larger the database size, the higher speed-up of the clustering (e.g., approximately 18 and 650 times faster for the 1000 and 10,000 objects, respectively). Unlike other state-of-the-art subspace clustering algorithms, PORSC is able to cluster one million data points in only 5 min.

Moreover, to investigate the scalability of PORSC with respect to dimensionality, five synthetic data sets containing four thousand objects with different dimensions ranging from 20 to 100 are generated. In Fig. 14b, PORSC shows a nearly linear running time against the number of dimensions. Meanwhile, the benefits of parallel implementation of ORSC are more pronounced with increasing dimensionality.

6.3 Impact of parametrization

In this section, we perform sensitivity analysis for PORSC with respect to the interaction range ϵ and partition size on the real-world cover type data set.

Partition size We assess the effect of the partition size on subspace clustering by using different settings ranging from 200 to 1000. The results in Fig. 15a demonstrate that the performance of PORSC is robust to the partition size (*psize*).

Interaction range To evaluate the impact of interaction range on clustering, we preform PORSC on cover type data set with different interaction range ranging from 0.1 to 0.4 (fixating $psize = 200$). Figure 15b demonstrates that the parametrization of interaction range is flexible and easy to specify as the clustering results are not very sensitive to it.

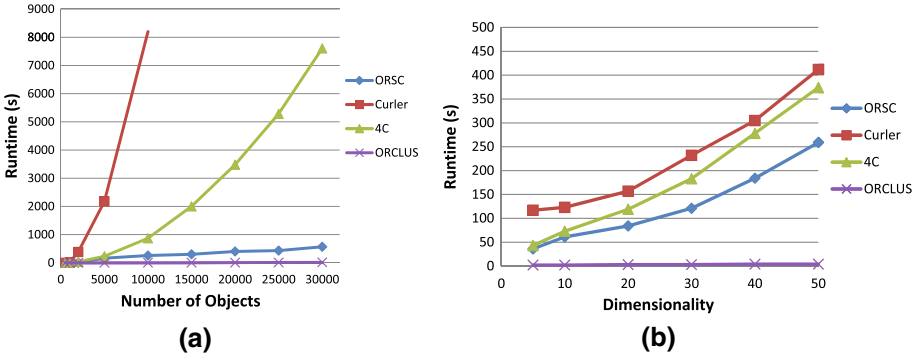


Fig. 13 Scalability of ORSC against the database size and dimensionality. **a** Database size. **b** Dimensionality

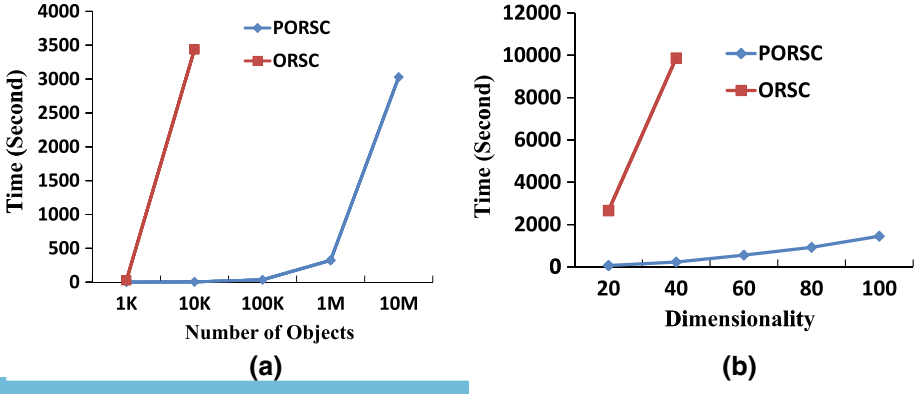


Fig. 14 Scalability of PORSC versus ORSC w.r.t. dimensions and database sizes. **a** Dimensions. **b** Database sizes

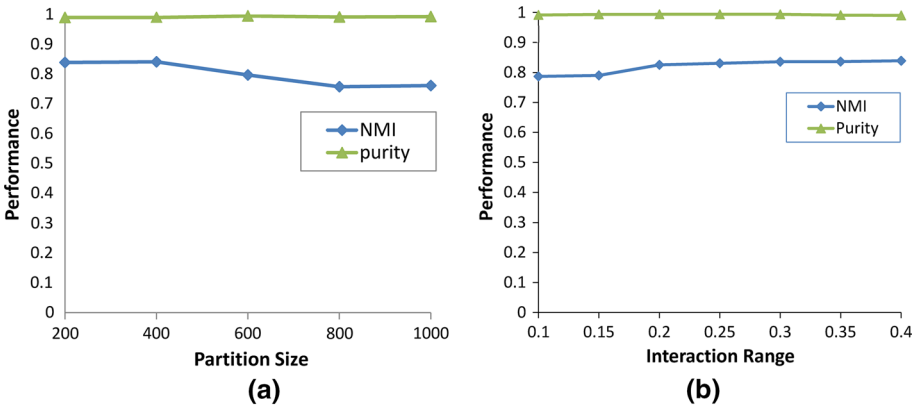


Fig. 15 Effect of the partition size and interaction range on clustering. **a** Partition size. **b** Interaction range

7 Discussion and conclusions

In this paper, we introduce ORSC, a novel subspace clustering algorithm based on synchronization. we consider each dimension of the object as a phase oscillator and simulates each object's dynamics according to our proposed interaction model. Through the weighted nonlinear interaction among objects, all correlated objects hidden in arbitrarily oriented subspaces can be synchronized together. The search of these synchronized clusters is transformed into the problem to find synchronized phases. Our extensive experiments demonstrate that the ORSC algorithm shows several desirable properties: (a) ORSC can naturally detect arbitrarily oriented subspace clusters driven by the true topological structure of the data without assuming any data distribution. (b) ORSC is robust against noise points and outliers since they are difficult to synchronize with other objects. (c) ORSC is flexible and easy to parameterize. In contrast to previous methods, there is no need to specify the subspace dimensionality, and all interesting subspace clusters are detected. (d) ORSC outperforms most comparison methods in terms of runtime efficiency and is highly scalable to large and high-dimensional data sets.

In ongoing and future work, we focus on exploiting automatical parametrization techniques for our interaction model based on information-theoretical principle. In addition, we want to closely investigate the movement patterns of objects and extend this idea to streaming data.

Acknowledgements The research was supported partially by the National Natural Science Foundation of China (Grant Nos. 61403062, 61433014, 41601025), China Postdoctoral Science Foundation (2014M552344, 2015M580786), Science-Technology Foundation for Young Scientist of SiChuan Province (2016JQ0007) and Fundamental Research Funds for the Central Universities (Grant Nos. ZYGX2014J053, ZYGX2014J091).

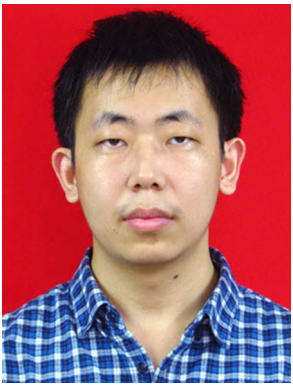
References

1. Aeyels D, De Smet F (2008) A mathematical model for the dynamics of clustering. Phys D Nonlinear Phenom 273(19):2517–2530
2. Aggarwal CC, Wolf JL, Yu PS et al (1999) Fast algorithms for projected clustering. ACM SIGMOD international conference on management of data, pp 61–72

3. Aggarwal CC, Yu P S (2000) Finding generalized projected clusters in high dimensional spaces. ACM SIGMOD international conference on management of data, pp 70–81
4. Agrawal R, Gehrke JE, Gunopulos D et al (1998) Automatic subspace clustering of high dimensional data for data mining applications. ACM SIGMOD international conference on management of data, pp 94–105
5. Ankerst M, Breunig MM, Kriegel HP et al (1999) Optics: ordering points to identify the clustering structure. ACM SIGMOD international conference on management of data, pp 49–60
6. Arenas A, Diaz-Guilera A, Perez-Vicente CJ (2006) Synchronization reveals topological scales in complex networks. Phys Rev Lett 96(11):1–4
7. Arenas A, Diaz-Guilera A, Kurths J et al (2008) Synchronization in complex networks. Phys Rep 469:93–153
8. Bahrololoum A, Nezamabadi-pour H, Saryazdi S (2015) A data clustering approach based on universal gravity rule. Eng Appl Artif Intell 45:415–428
9. Böhm C, Kailing K, Kröger P et al (2004) Computing clusters of correlation connected objects. ACM SIGMOD international conference on management of data, pp 455–466
10. Böhm C, Plant C, Shao J et al (2010) Clustering by synchronization. ACM SIGKDD international conference on knowledge discovery and data mining, pp 583–592
11. Cheng CH, Fu AW, Zhang Y (1999) Entropy-based subspace clustering for mining numerical data. ACM SIGKDD international conference on knowledge discovery and data mining, pp 84–93
12. Elhamifar E, Vidal R (2013) Sparse subspace clustering: algorithm, theory, and applications. IEEE Trans Pattern Anal Mach Intell 35(11):2765–2781
13. Frey B, Dueck D (2007) Clustering by passing messages between data points. Science 315:972–976
14. Givoni I, Chung C, Frey B (2011) Hierarchical affinity propagation. 27th conference on uncertainty in artificial intelligence, Barcelona, Spain
15. Goil S, Nagesh H, Choudhary A (1999) MAFIA: efficient and scalable subspace clustering for very large data sets. ACM SIGKDD international conference on knowledge discovery and data mining, pp 443–452
16. Günnemann S, Faloutsos C (2013) Mixed membership subspace clustering. IEEE international conference on data mining, pp 221–230
17. Hinneburg A, Keim DA (1999) Optimal grid-clustering: towards breaking the curse of dimensionality in high-dimensional clustering. International conference on very large data bases, pp 506–517
18. Huang J, Sun H, Kang J et al (2013) ESC: an efficient synchronization-based clustering algorithm. Knowl Based Syst 40:111–122
19. Indulska M, Orlowska M (2002) Gravity based spatial clustering. ACM international symposium on advances in geographic information systems, pp 125–130
20. Jain AK, Dubes RC (1988) Algorithms for clustering data. Prentice-Hall, Upper Saddle River
21. Kailing K, Kriegel HP, Kröger P (2004) Density-connected subspace clustering for high-dimensional data. SIAM international conference on data mining, p 4
22. Kim CS, Bae CS, Tcha HJ (2008) A phase synchronization clustering algorithm for identifying interesting groups of genes from cell cycle expression data. BMC Bioinform 9:1
23. Kuramoto Y(1975) Self-entrainment of a population of coupled nonlinear oscillators. In: Araki H (ed) Proceedings of the international symposium on mathematical problems in theoretical physics. Lecture notes in physics. Springer, New York, pp 420–422
24. Kuramoto Y (1984) Chemical oscillations, waves, and turbulence. Springer, Berlin
25. Liu J, Wang W (2003) Op-cluster: clustering by tendency in high dimensional space. IEEE international conference on data mining, pp 187–194
26. Oyang Y, Chen C, Yang T (2001) A study on the hierarchical data clustering algorithm based on gravity theory. Principles of data mining and knowledge discovery, pp 350–361
27. Procopiuc CM, Jones M, Agarwal PK et al (2002) A Monte Carlo algorithm for fast projective clustering. ACM SIGMOD international conference on management of data, pp 418–427
28. Shao J (2012) Synchronization on data mining: a universal concept for knowledge discovery. LAP LAMBERT Academic Publishing, Saarbrücken
29. Shao J, He X, Böhm C et al (2013) Synchronization-inspired partitioning and hierarchical clustering. IEEE Trans Knowl Discov Data Eng 25(4):893–905
30. Shao J, Yang Q, Dang H et al (2016) Scalable clustering by iterative partitioning and point attractor representation. ACM Trans Knowl Discov Data 11(1):5
31. Shao J, Ahmadi Z, Kramer S (2014) Prototype-based Learning on concept-drifting data streams. ACM SIGKDD international conference on knowledge discovery and data mining, pp 512–521
32. Shao J, Böhm C, Yang Q et al (2010) Synchronization based outlier detection. ECML/PKDD 2010, pp 245–260

33. Shao J, He X, Yang Q et al (2013) Robust synchronization-based graph clustering. Pacific-Asia conference on knowledge discovery and data mining, pp 249–260
34. Tung AKH, Xu X, Ooi BC (2005) Curler: finding and visualizing nonlinear correlated clusters. ACM SIGMOD international conference on management of data, pp 467–478
35. Vinh NX, Epps J, Bailey J (2009) Information theoretic measures for clusterings comparison: is a correction for chance necessary?. In: The 26th international conference on machine learning, pp 1073–1080
36. Wang H, Wang W, Yang J et al (2002) Clustering by pattern similarity in large data sets. ACM SIGMOD international conference on management of data, pp 394–405
37. Ying W, Chung F, Wang S (2014) Scaling up synchronization-inspired partitioning clustering. IEEE Trans Knowl Data Eng 26(8):2045–2057
38. Zhang T, Ramakrishnan R, Livny M (1996) An efficient data clustering method for very large databases. ACM SIGMOD international conference on management of data, pp 103–114

Junming Shao received his Ph.D. degree with highest honer ("Summa Cum Laude") at the University of Munich, Germany, in 2011. He became the Alexander von Humboldt Fellow in 2012. Currently, he is professor of Computer Science at the University of Electronic Science and Technology of China. His research interests include data mining and neuroimaging. He not only published papers on top-level data mining conferences like KDD, ICDM, SDM (two of those papers have won the Best Paper Award), but also published data mining-related interdisciplinary work in leading journals including Brain, Neurobiology of Aging, and Water Research.



Xinzuo Wang received his Bachelor Degree in the Institute of Computer Science at the University of Electronic Science and Technology of China. Currently, he is studying for his mater degree at the Northwestern University, USA. His research interests include data mining and machine learning, especially high-dimensional clustering, scalable clustering and heterogenous network mining.

Qinli Yang received the Ph.D. degree in 2011 from the university of Edinburgh and is an associate professor working in the University of Electronic Science and Technology of China. Her research focuses on interdisciplinary work of water resources research and data mining, especially for hydrological data mining in the changing environment. She not only contributed to leading journals including Water Research, Environmental Modelling and Software but also published work in top-level data mining conferences like KDD and ICDM.

Claudia Plant received her Ph.D. in 2007 and currently is the head of research group Data Mining at University of Vienna. Her research focusses on database-related data mining, especially clustering, parameter-free data mining and integrative mining of heterogeneous data. She not only contributed to top-level database and data mining conferences like KDD, SIGMOD, ICDM, ICDE but also published application-related interdisciplinary work in leading journals including Bioinformatics, Cerebral Cortex and Water Research.

Christian Böhm is professor of Computer Science at the University of Munich (Ludwig Maximilians Universität), Germany. He received his Ph.D. in 1998 and his habilitation in 2001. His former affiliations include the Technische Universität München and the University of Health Sciences, Medical Informatics and Technology, Hall in Tyrol, Austria. His research focus is on database systems and data mining, particularly index structures for similarity search and clustering algorithms. He has received 4 research awards including the SIGMOD best paper award 1997 and the SIAM SDM best paper honorable mention award in 2008.