EasyChair Preprint

№ 1438

An effective multi-level synchronization clustering method based on a framework of "divide and collect" and SSynC algorithm

Xinquan Chen

August 28, 2019

Title Page

An effective multi-level synchronization clustering method based on a framework of "divide and collect" and SSynC algorithm

Xinquan Chen[1, 2]

[1]School of Computer & Information, Anhui Polytechnic University, Wuhu, 241000, China

[2]Key Laboratory of Intelligent Information Processing and Control, Chongqing Three Gorges University, Chongqing, 404100, China

chenxqscut@126.com

Author: Xinquan Chen

Corresponding Author: Xinquan Chen

*** Corresponding author**. Tel.: 0086-15123428097.

E-mail address: chenxqscut@126.com (X. Chen).

Post Address:

Xinquan Chen

School of Computer & Information, Anhui Polytechnic University, Wuhu, 241000, China

Conflicts of interest: None

An effective multi-level synchronization clustering method based on a framework of “divide and collect” and SSynC algorithm

Abstract: Facing big data, general clustering methods cannot process all data in main memory one time. In order to conquer this problem, this paper presents an effective Multi-Level Synchronization Clustering (MLSynC) method based on SSynC algorithm by using a framework of “divide and collect” and a linear weighted Vicsek model. MLSynC method has different process with SynC algorithm, ESynC algorithm, and SSynC algorithm. In this paper, we present two concrete implementations of MLSynC method, a two-level framework algorithm and a recursive algorithm. By the theoretic analysis, we find the time complexity of MLSynC method is less than SSynC algorithm. By some simulated experiments of some artificial data sets, eight UCI data sets, and three picture data sets, we observe that MLSynC method not only gets better local synchronization effect but also needs less iterative times and time cost than SynC algorithm. Moreover, we also observe that MLSynC method not only needs less time cost than ESynC algorithm and SSynC algorithm, but also almost gets the same local synchronization effect as ESynC algorithm and SSynC algorithm if the partition of the data set is proper. Further comparison experiments with some classical clustering algorithms demonstrate the clustering effect of MLSynC method.

Keywords: Divide and collect; Kuramoto model; Shrinking synchronization clustering; A linear weighted Vicsek model; Near neighbor point set

1. Introduction

Clustering is an unsupervised learning method that tries to find some obvious distribution structures and patterns in unlabeled data sets by maximizing the similarity of the objects in a common cluster and minimizing the similarity of the objects in different clusters (Jain et al., 1999). Clustering has been used in many areas such as machine learning, pattern recognition, image processing, marketing and costumer analysis, agriculture, security and crime detection, information retrieval, and bioinformatics. Cluster is often one important step in the process of data analysis.

Clustering algorithms have been studied for decades. There have been hundreds of clustering algorithms until now, but none of them is all purpose. Almost all clustering algorithms have flaws. Some clustering algorithms are suitable for dealing with data with certain types, and others are suitable for handling data with special distribution structures. Many real data have complex distributions, diversiform types,

great capacity, noises, or isolates. So there is a continuous demand for researching different kinds of clustering methods. In order to obtain better clustering results in real-world applications where the amount of data is often very large and the types of data are diversiform, researchers try their best to develop new efficient and effective clustering algorithms.

The traditional clustering algorithms are usually classified into partitioning methods (Bezdek, 1981; MacQueen, 1967), hierarchical methods (Bouguettaya et al., 2015; Guha et al., 1998; Karypis et al., 1999; Zhang et al., 1996), density-based methods (Ankerst et al., 1999; Ester et al., 1996; Roy et al., 2005), grid-based methods (Agrawal et al., 1998; Wang et al., 1997), model-based methods (Theodoridis et al., 2006), and graph-based methods (Tan et al., 2005; Theodoridis et al., 2006; Zahn, C. T., 1971). Recent clustering methods have quantum clustering algorithms (Horn et al., 2002), spectral clustering algorithms (Luxburg, 2007; Schölkopf et al., 1998), affinity propagation clustering algorithms (Frey et al., 2007), synchronization clustering algorithms (Böhm et al., 2010; Huang et al., 2013; Shao et al., 2013a, 2013b, 2014; Chen, 2014, 2017, 2018; Hang et al., 2017), and so on.

Recently, several original clustering algorithms, such as Affinity Propagation (AP) algorithm (Frey et al., 2007), Synchronization Clustering (SynC) algorithm (Böhm et al., 2010), and clustering by fast search and find of Density Peaks (DP) algorithm (Rodriguez et al., 2014), were published. AP algorithm is a new type of clustering algorithm published on *Science* in 2007. After AP algorithm was published, clustering based on probability graph models grew a new research direction. As we know, SynC algorithm is the first synchronization clustering algorithm. After SynC algorithm was presented, synchronization clustering attracts some researchers. Some synchronization clustering methods (Huang et al., 2013; Shao et al., 2013a, 2013b, 2014; Chen, 2014, 2017, 2018; Hang et al., 2017) were published from different views. DP algorithm is a clustering algorithm based on the assumption that “cluster centers can be characterized by a higher density than their neighbors and by a relatively large distance from points with higher densities”. In DP algorithm, the number of clusters can be obtained automatically, outliers can be identified easily, and even nonspherical clusters can be explored quickly. So we think DP algorithm can lead a new research direction in clustering field.

Synchronization clustering is a kind of novel clustering approach. The original synchronization clustering algorithm (named as SynC) says that it can find the

intrinsic structure of the data set without any distribution assumptions and handle outliers by dynamic synchronization (Böhm et al., 2010).

In the circumstance of big data, general clustering algorithms cannot process a large amount of data in main memory one time. In order to conquer this problem, basing on SynC algorithm (Böhm et al., 2010), ESynC algorithm (Chen, 2017), and SSynC algorithm (Chen, 2014), this paper researches a Multi-Level Synchronization Clustering (MLSynC) method based on SSynC algorithm by using a framework of "divide and collect" and a linear weighted Vicsek model. MLSynC method has a different process with SynC algorithm, ESynC algorithm, and SSynC algorithm. SynC algorithm (Böhm et al., 2010) is based on an extensive Kuramoto model, ESynC algorithm (Chen, 2017) is based on a linear version of Vicsek model, SSynC algorithm (Chen, 2014) is based on a linear weighted Vicsek model, and MLSynC method uses the strategy of "divide and conquer" and a linear weighted Vicsek model for clustering. Because the linear weighted Vicsek model has nicer superposition characteristic for clustering, MLSynC method can be used to process big data effectively and efficiently.

The idea of "divide and collect" in MLSynC method is similar to MapReduce framework (Dean et al., 2008) in some aspects, although it is developed independently. MapReduce is a parallel and distributed programming model that is used to process very large data sets on a cluster. In MapReduce framework, the Map and Reduce operations affect the clustering result very much, so they cannot be directly extended to clustering field. MLSynC method can be used for clustering on a cluster with a parallel programming model or on a personal computer with a serial programming method. If the partition of the data set is proper, MLSynC method is both efficient and effective.

The remainder of this paper is organized as follows. Section 2 lists some related work. Section 3 gives some basic knowledge. Section 4 introduces MLSynC method. Section 5 validates MLSynC method by some simulated experiments. Conclusions and future works are presented in Section 6.

2. Related work

This paper is inspired by several papers (Vicsek et al., 1995; Jadbabaie et al., 2003; Wang et al., 2009; Böhm et al., 2010; Chen, 2014, 2017) and the strategy of "divide and conquer".

In 2010, Böhm et al. presented a novel clustering approach, SynC algorithm, inspired by the synchronization principle. SynC algorithm can find the intrinsic structure of the data set without any distribution assumptions and handle outliers by dynamic synchronization. In order to implement automatic clustering, those natural clusters can be discovered by using the Minimum Description Length (MDL) principle (GrÄunwald, 2005). After SynC algorithm was presented, some researchers published several synchronization clustering papers from different views (Shao et al., 2010, 2011, 2013a, 2013b, 2014; Huang et al., 2013; Chen, 2014, 2017, 2018; Hang et al., 2017). In order to find subspace clusters of some high-dimensional sparse data sets, a novel effective and efficient subspace clustering algorithm, ORSC (Shao et al., 2011), was proposed. In order to detect the outliers from a real complex data set more naturally, a novel outlier detection algorithm was presented from a new perspective, "Out of Synchronization" (Shao et al., 2010). In order to find the intrinsic patterns of a complex graph, a novel and robust graph clustering algorithm, RSGC (Shao et al., 2013a), was proposed by regarding the graph clustering as a dynamic process towards synchronization. In order to explore meaningful levels of the hierarchical cluster structure, a novel dynamic hierarchical clustering algorithm, hSync (Shao et al., 2013b), was presented based on synchronization and the MDL principle. In 2013, Huang et al. (2013) also presented a synchronization-based hierarchical clustering method basing on the work of Böhm et al. (2010). Inspired by the work of Böhm et al. (2010) and Vicsek model, Chen (2014) presented a Shrinking Synchronization Clustering (SSynC) algorithm by using a linear weighted Vicsek model. Inspired by the work of Böhm et al. (2010), Chen (2017) proposed an Effective Synchronization Clustering (ESynC) algorithm based on a linear version of Vicsek model. Simulations validate that the linear version of Vicsek model is an effective synchronization model for clustering. Based on the metaphor of gravitational kinematics and central force optimization method, Hang et al. (2017) presented a local synchronization clustering algorithm, which can find clusters of those data sets with arbitrary size, shape, and density, and determine the number of clusters automatically. Chen (2018) present three Fast Synchronization Clustering (FSynC) algorithms basing on the work of Böhm et al. (2010) and the grid-based index method in Chen (2013). FSynC algorithm, which is a parametric algorithm, is an improved version of SynC algorithm by combining multidimensional grid partitioning method and Red-Black tree structure to construct the near neighbor point sets of all points (Chen, 2018).

Recent, Cao et al. (Cao et al., 2017) proposed a fuzzy SV-k-modes algorithm that can cluster categorical data with set-valued attributes by using a fuzzy k-modes clustering process. The fuzzy SV-k-modes algorithm can cluster categorical data with single-valued and set-valued attributes together. Spurek et al. (Spurek et al., 2017) proposed an active function Cross-Entropy Clustering (afCEC) method by using Gaussians in curvilinear coordinate systems. The afCEC method can adapt well to curved and strongly nonlinear data and automatically determine the number of clusters. Güngör et al. (Güngör et al., 2017) proposed a Gaussian Density Distance (GDD) clustering algorithm by using both Gaussian kernel and distances to form clusters according to the density and shape of data set. The GDD algorithm can find best possible clusters without any prior information and parameters.

3. Some basic knowledge

Suppose there is a data set $S = \{X_1, X_2, \ldots, X_n\}$ in a d-dimensional Euclidean space. Naturally, we use Euclidean metric as our dissimilarity measure, $dis(\cdot, \cdot)$. In order to describe our method clearly, some concepts are presented first.

Definition 1 The δ near neighbor point set $\delta(P)$ of point P is defined as:

$$\delta(P) = \{X \mid 0 < dis(X, P) \leq \delta, X \neq P, X \in S\}, \tag{1}$$

where $dis(X, P)$ is the dissimilarity measure between point X and point P in the data set S. Parameter δ is a predefined threshold.

Definition 2 (Böhm et al., 2010). The extensive Kuramoto model for clustering is defined as:

Point $X = (x_1, x_2, \ldots, x_d)$ is a vector in d-dimensional Euclidean space. If each point X is regarded as a phase oscillator based on Kuramoto model, with an interaction in the δ near neighbor point set $\delta(X)$, then the dynamics of the k-th dimension x_k ($k = 1, 2, \ldots, d$) of point X over time is described by:

$$x_k(t+1) = x_k(t) + \frac{1}{|\delta(X(t))|} \sum_{Y \in \delta(X(t))} \sin(y_k(t) - x_k(t)), \tag{2}$$

where $X(t = 0) = (x_1(0), x_2(0), \cdots, x_d(0))$ represents the original phase of point X, $x_k(t+1)$ describes the renewal phase value in the k-th dimension of point X at the t-step evolution, and $Y = (y_1, y_2, \ldots, y_d)$ is a δ near neighbor point of point X at the t-step evolution.

Definition 3 (Chen, 2017). A linear version of Vicsek model for clustering is defined as:

Point $X = (x_1, x_2, \cdots, x_d)$ is a vector in d-dimensional Euclidean space. If each point X is regarded as an agent based on a linear version of Vicsek model, with an interaction in the δ near neighbor point set $\delta(X)$, then the dynamics of point X over time according to Jadbabaie et al. (2003) and Wang et al. (2009) is described by:

$$X(t+1) = \frac{1}{\left(1 + \left|\delta(X(t))\right|\right)}\left(X(t) + \sum_{Y \in \delta(X(t))} Y\right), \tag{3}$$

where $X(t = 0) = (x_1(0), x_2(0), \cdots, x_d(0))$ represents the original location of point X, and $X(t+1)$ describes the renewal location of point X at the t-step evolution.

Definition 4 (Chen, 2014). A core is defined as:

In Shrinking Synchronization Clustering (SSynC) algorithm, point X can be regarded as an active core C if and only if:

(1). Point X is active in the current synchronization step.

(2). Point X is not labeled as an attributive point of another core.

At this time, the other points in the ε near neighbor point set $\varepsilon(C)$ of core C should be labeled as attributive points of core C, where parameter ε is a small real number that is less than parameter δ. Usually, if the distance of two points is less than ε, then they are regarded in the same cluster.

The data structure of core C can be defined as:

$$\mathrm{DS}(C) = (\text{Core\_Id, Core\_Location, Parent\_CoreId, Number\_ContainingPoints}). \tag{4}$$

In Eq.(4),

Core_Id is the identification number in the original data set.

Core_Location is the current location of core C. It is a d-dimensional vector expressed by $C = (c_1, c_2, \cdots, c_d)$.

Parent_CoreId is the identification number of the parent of core C. In the first step of dynamic clustering, the Parent_CoreId of core C is itself. In the middle or final step of dynamic clustering, the Parent_CoreId of core C is the Core_Id of the attributive core of core C.

Number_ContainingPoints is the number of points that are represented or contained by core C.

Definition 5 (Chen, 2014)**.** A synchronization model for clustering a core set is defined as:

Core $C = (c_1, c_2, \cdots, c_d)$ is a vector in a d-dimensional Euclidean space. If each core C is regarded as a phase oscillator based on an extended linear version of Vicsek

model (this model is also named as the linear weighted Vicsek model), with an interaction in the δ near neighbor point set $\delta(C)$, then the dynamics of core C over time is described by:

$$C(t+1) = \frac{1}{(count(C(t)) + \sum_{Y \in \delta(C(t))} count(Y))} \left(count(C(t)) \cdot C(t) + \sum_{Y \in \delta(C(t))} (count(Y) \cdot Y) \right), \quad (5)$$

where $C(t = 0) = (c_1(0), c_2(0), \cdots, c_d(0))$ represents the original phase of core C, $C(t+1)$ describes the renewal phase value of core C at the t-step evolution, and $count(C)$ represents the value of the Number_ContainingPoints of core C.

In the dynamics clustering, if the Parent_CoreId of core C is itself and the value of the Number_ContainingPoints of core C is equal to 1, then Eq.(5) is equivalent with Eq.(3). Actually, in the dynamics clustering, if core C is represented by its parent core (which means that the value of the Number_ContainingPoints of the parent core of core C is added by $count(C)$), then Eq.(5) can be used for saving time and space.

Definition 6 (Chen, 2014)**.** The data set $S = \{X_1, X_2, \cdots, X_n\}$ using the linear weighted Vicsek model described by Eq.(5) for clustering is said to achieve local synchronization if the final locations of all points satisfy:

$$parent(X_i(t = T)) = RC_k(T),\ i = 1, 2, \cdots, n,\ k = 1, 2, \cdots, K. \quad (6)$$

In Eq.(6), where T is the times of the final synchronization, K is the number of the root cores in the final synchronization step, $RC_k(T)$ is the k-th root core in the final synchronization step, and $parent(X_i(t = T))$ is the parent of core $C_i(T)$ after the path-compressed process in the final synchronization step.

Usually, the final location of point X_i (i = 1, 2, $\cdots$, n) depends on the value of parameter δ and the original location of itself and the original locations of other points in the data set S.

Definition 7. The data set $S = \{X_1, X_2, \ldots, X_n\}$ uses the linear weighted Vicsek model described by Eq.(5) for synchronization clustering. In each evolution step of synchronization clustering, all cores become some trees with synchronization action. When the number of root cores in the t-step evolution is equal to that in the $(t+1)$-step evolution, an average difference between the root cores in the t-step evolution and the root cores in the $(t+1)$-step evolution is defined as:

$$differInRootCores(t, t+1) =$$

$$\frac{1}{n_t} \sum_{k=1}^{n_t} dis(RC_k(t).Core\_Location, RC_k(t+1).Core\_Location),\ k = 1, 2, \cdots, n_t, \quad (7)$$

where n_t is the number of the root cores in the t-step evolution, $RC_k(t)$.Core_Location is the location of the k-th root core in the t-step evolution, and $dis(RC_k(t).\text{Core\_Location}, RC_k(t+1).\text{Core\_Location})$ is the dissimilarity between the location of the k-th root core in the t-step evolution and that in the $(t+1)$-step evolution.

Apparently, if the average difference between the root cores in the t-step evolution and that in the $(t+1)$-step evolution computed by Eq.(7) is less than a predefined threshold, we think SSynC algorithm can exit.

Theorem 1 (Chen, 2014). The data set $S = \{X_1, X_2, \cdots, X_n\}$ using the linear weighted Vicsek model described by Eq.(5) for clustering will achieve local synchronization, if parameter δ satisfies:

$$\delta_{\min} \le \delta \le \delta_{\max}. \tag{8}$$

Suppose $e_{\min}(\text{MST}(S))$, which is equal to $\min\{dis(X_i, X_j) | (X_i, X_j \in S) \wedge (X_i \neq X_j)\}$, is the weight of the minimum edge in the Minimum Span Tree (MST) of the complete graph of the data set S, and $e_{\max}(\text{MST}(S))$ is the weight of the maximum edge in the MST of the complete graph of the data set S. Apparently, there is $\delta_{\min} = e_{\min}(\text{MST}(S))$. If the data set S has no isolate, then usually there is $e_{\max}(\text{MST}(S)) \le \delta_{\max} \le \max\{dis(X_i, X_j) | (X_i, X_j \in S) \wedge (X_i \neq X_j)\}$. If the data set S has isolates, we should filtrate all isolates at first.

Proof: if $\delta < \delta_{\min}$, then for any point X_i ($i = 1, 2, \cdots, n$), there is $\delta(X_i) = \varnothing$. In this case, any point in the data set S cannot synchronize with other points, so synchronization will not happen.

In another case, that is $\delta > \delta_{\max}$, then for any point X_i ($i = 1, 2, \cdots, n$), there is $\delta(X_i(t)) = S - \{X_i(t)\}$. According to Eq.(5), there is $X_i(t+1) = \text{mean}(S)$. Here, mean($S$) is the mean of all points in the data set S. Any point in the data set S will synchronize with all other points, so global synchronization happens. After one time synchronization, all points in the data set S will synchronize to their mean location.

Apparently, if $\delta_{\min} \le \delta \le \delta_{\max}$, local synchronization will happen. And the final result of synchronization is affected by the value of parameter δ and the original locations of all points in the data set S.

Property 1 (Chen, 2014). The data set $S = \{X_1, X_2, \cdots, X_n\}$ using the linear weighted Vicsek model described by Eq.(5) for clustering will obtain an effective

result of local synchronization with some obvious clusters or isolates, if parameter δ satisfies:

$$\max\{longestEdgeInMst(cluster_k) \mid k = 1, 2, \cdots, K\} < \delta < \min\{dis(cluster_i, cluster_j) \mid i, j = 1, 2, \cdots, K\}. \quad (9)$$

In Eq.(9), $longestEdgeInMst(cluster_k)$ is the weight of the longest edge in the minimum spanning tree of the k-th cluster, $dis(cluster_i, cluster_j)$ is the weight of the minimum edge connecting the i-th cluster and the j-th cluster, and K is the number of clusters in the final synchronization step.

Proof: Suppose the data set $S = \{X_1, X_2, \ldots, X_n\}$ has K obvious clusters. If parameter δ is larger than or equal to $\max\{longestEdgeInMst(cluster_k) \mid k = 1, 2, \cdots, K\}$, then data points in the same cluster will synchronize. If parameter δ is less than $\min\{dis(cluster_i, cluster_j) \mid i, j = 1, 2, \cdots, K\}$, then data points in different obvious clusters cannot synchronize.

4. An effective multi-level synchronization clustering method based on a framework of "divide and collect" and SSynC algorithm

Facing big data, general clustering methods cannot process all data in main memory one time. In order to conquer this problem, we present an effective Multi-Level Synchronization Clustering (MLSynC) method by using a framework of "divide and collect" and a linear weighted Vicsek model. The framework of "divide and collect" is an application of the strategy of "divide and conquer" in clustering field. The linear weighted Vicsek model described by Eq.(5) is used in SSynC algorithm.

Although we use the Euclidean metric as our dissimilarity measure in this paper, this method is by no means restricted to this metric and this kind of data space. If we can construct a proper dissimilarity measure in a hybrid-attribute space, this method can still be used.

4.1 The description of SSynC algorithm

Shrinking Synchronization Clustering (SSynC) algorithm (Chen, 2014) is presented in another paper by Chen, X. SSynC algorithm based on the synchronization model represented by Eq.(5) has a similar process with SynC algorithm and ESynC algorithm. In order to make a difference between SSynC algorithm and MLSynC method, we introduce it below.

Algorithm name: Shrinking Synchronization Clustering algorithm (named as

SSynC algorithm).

Input: Data set $S = \{X_1, X_2, \cdots X_n\}$, dissimilarity measure $dis(\cdot, \cdot)$, parameter δ, and parameter ε (a very small real number, if the distance of two points is less than ε, then they are regarded in the same cluster).

Output: The final core set $C(T) = \{C_1(T), C_2(T), \cdots, C_n(T)\}$ and the number of root cores in $C(T)$.

The main procedure of SSynC algorithm is described by **Table 1**.

Table 1

The initialization and iterative synchronization clustering procedure of SSynC algorithm.

1: IterateStep t is set as zero firstly, that is: $t \leftarrow 0$;
/* Create initial core set $C(t=0) = \{C_1(t=0), C_2(t=0), \ldots, C_n(t=0)\}$. */
2: **for** ($i = 1$; $i \le n$; i++)
3: **{**
4: $C_i(t=0)$.Core_Id $\leftarrow i$;
5: $C_i(t=0)$.Core_Location $\leftarrow X_i$;
6: $C_i(t=0)$.Parent_CoreId $\leftarrow i$;
7: $C_i(t=0)$.Number_ContainingPoints $\leftarrow 1$;
8: **}**
/* Create initial active point set $AP(t=0)$. */
9: $AP(t=0) \leftarrow \{X_1, X_2, \cdots, X_n\}$;
10: NumberOfAP($t=0$) $\leftarrow n$;
11: **while** ((the dynamical synchronization clustering does not satisfy its convergent condition) **and** ($t < 50$))
12: **{**
13: **for** (each point $Y(t)$ in the active point set $AP(t)$)
14: **{**
15: According to Definition 1, construct the δ near neighbor point set $\delta(Y(t))$ for point $Y(t)$ in the active point set $AP(t)$;
16: Compute the renewal value, $Y(t+1)$, of $Y(t)$ using Eq.(5);
17: **}** /* After the above for repetition, we get an update point set $AP(t+1)$ that is composed of the renewal value $Y(t+1)$ of each point $Y(t)$ in the active point set $AP(t)$. */
18: **for** (each unlabeled point $Y(t+1)$ in the point set $AP(t+1)$)
19: **{**
20: The member "Core_Location" of the corresponding core of point $Y(t+1)$ is updated by the value of $Y(t+1)$;
21: According to Definition 1, construct the ε near neighbor point set $\varepsilon(Y(t+1))$ for point $Y(t+1)$ in the point set $AP(t+1)$;
22: **for** (each unlabeled point $Z(t+1)$ in the ε near neighbor point set $\varepsilon(Y(t+1))$ of point $Y(t+1)$)
23: **{**
24: Point $Z(t+1)$ is labeled as inactive point;
25: The member "Parent_CoreId" of the corresponding core of point $Z(t+1)$ is assigned by the member "Core_Id" of the corresponding core of point $Y(t+1)$;
26: The member "Number_ContainingPoints" of the corresponding core of point $Z(t+1)$ is added into the member "Number_ContainingPoints" of the corresponding core of point $Y(t+1)$;
27: **}**
28: **}**
29: Delete all labeled inactive points from $AP(t+1)$; /* After this deleting process, $AP(t+1)$ only contains those active points, which are also the root nodes in its disjoint-set forest. */
30: NumberOfAP($t+1$) is assigned by the current number of unlabeled points of the renewal

active point set $AP(t+1)$;
31: IterateStep t is increased by 1, that is: t++;
32: **if** (NumberOfAP(t+1) == NumberOfAP(t) **and** (the difference between $AP(t+1)$ and $AP(t)$ is very small)) /* Here, NumberOfAP(t+1) == NumberOfAP(t) means the number of points in the renewal active point set $AP(t+1)$ is equal to the number of points in the active point set $AP(t)$, and the difference between $AP(t+1)$ and $AP(t)$ can be computed by Eq.(7). */
33: We think the dynamical clustering reaches its convergent result, and then exit from the while repetition;
34: **}**
35: Compress the paths of some inactive core points in $C(t)$ just like the joint-set method such that the largest height of leaf core points is less than or equal to two (Note: the height of root core points is one).
36: Finally we get a core set $C(T) = \{C_1(T), C_2(T), \cdots, C_n(T)\}$ and the number of root cores in $C(T)$, where T is the times of the above while repetition. The final convergent set $C(T)$ reflects the natural clusters or isolates of the data set S.

4.2 The application condition of MLSynC method

Online Resource 1 of Supplementary Material of this paper presents a figure (named as **Fig. 1 of Supplementary Material**) that compares the synchronization clustering results of MLSynC method using two different partitioning methods, a random partitioning method and a direct partitioning method. From **Fig. 1 of Supplementary Material**, we observe that if the spatial distributions of two partitioned data subsets vary very large and the clustering structure of the original data set is dissevered by the partitioning, MLSynC method will get a different clustering result with SSynC algorithm. If the spatial distributions of two partitioned data subsets vary very small, or the partitioning is uniform, MLSynC method will get a similar clustering result with SSynC algorithm.

4.3 The description of a two-level framework algorithm of MLSynC method

MLSynC method has a different clustering process with SynC algorithm (Böhm et al., 2010), ESynC algorithm (Chen, 2017), and SSynC algorithm (Chen, 2014). **Figure 1** presents a two-level framework algorithm of MLSynC method. In the two-level framework algorithm of MLSynC method, original data set that is usually large and cannot be processed in main memory one time is partitioned into m subsets. Each subset is processed by a clustering model (or a clustering machine) based on SSynC algorithm. After collected all root cores from the m clustering models, a clustering model based on SSynC algorithm is used.

In **Figure 1**, if m is too large, then the two-level framework algorithm of MLSynC method should be replaced by a three-level (or four-level above) framework algorithm. A three-level framework algorithm of MLSynC method is presented in Online Resource 2 of Supplementary Material of this paper.

From **Figure 1**, we observe that MLSynC method has a natural framework based

on SSynC algorithm by integrating the clustering results of all subsets. MLSynC method also has nicer incremental clustering ability. For example, in **Figure 1**, if every subset in $\{S_2, S_3, \cdots, S_m\}$ only has one point, then the two-level framework algorithm becomes an incremental clustering algorithm.

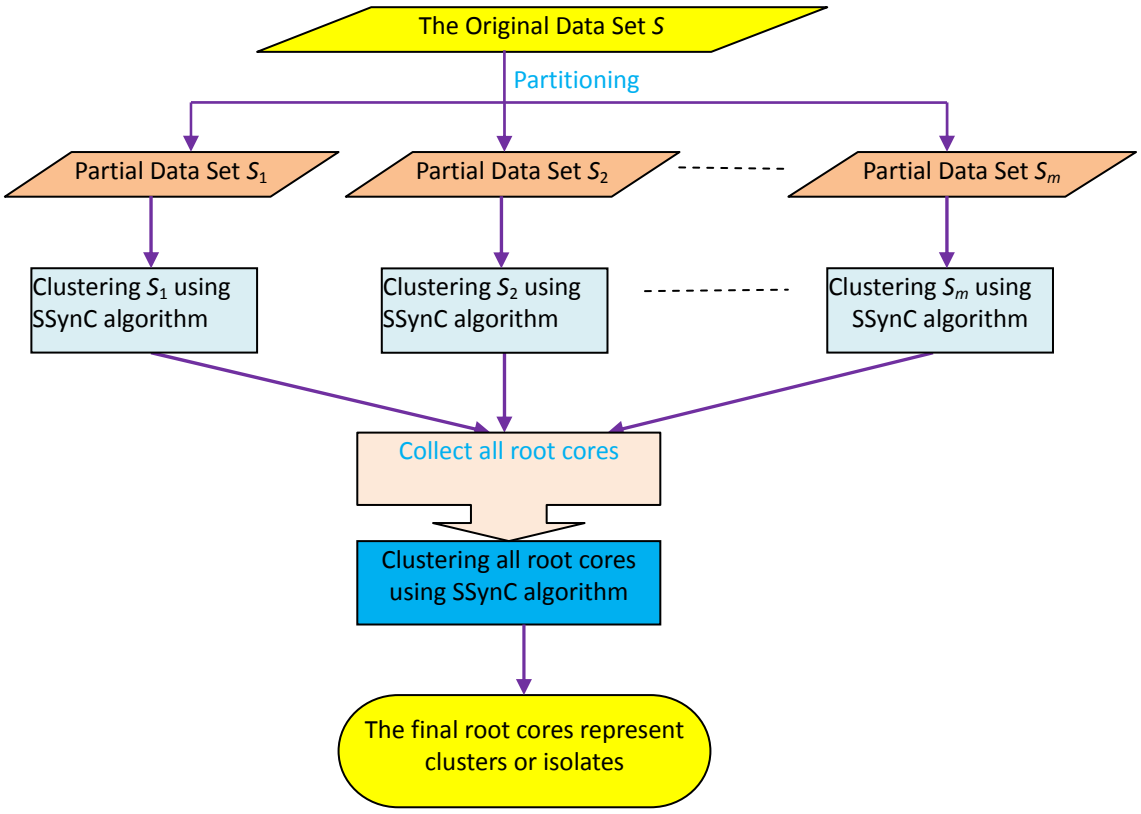


Fig. 1 A two-level framework algorithm of MLSynC method

Here, we present a description of the two-level framework algorithm of MLSynC method.

Algorithm name: The two-level framework algorithm of MLSynC method

Input: Data set $S = \{X_1, X_2, \cdots, X_n\}$, dissimilarity measure $dis(\cdot, \cdot)$, parameter m, a setting set of parameter δ: $\{\delta_1, \delta_2, \cdots, \delta_m, \delta_{merge}\}$, and parameter ε.

Output: The clustering result of the data set $S = \{X_1, X_2, \cdots, X_n\}$.

The main procedure of the two-level framework algorithm of MLSynC method is described by **Table 2**.

Table 2

The main procedure of the two-level framework algorithm of MLSynC method.

1 **Step1:** Partition the data set $S = \{X_1, X_2, \cdots, X_n\}$ that is read from a file or a data base into m sections, $\{S_1, S_2, \cdots, S_m\}$. Usually, $S = S_1 \cup S_2 \cup \cdots \cup S_m$ should be satisfied.
2 **Step2:** Clustering each subset S_i (i = 1, 2, …, m) from $\{S_1, S_2, \cdots, S_m\}$ using SSynC algorithm. That is:
2.1: int NumOfClusters[m]; /* Each element in array NumOfClusters is used to record the number

of clusters of each subset. */
2.2: **for** (i = 1; $i \le m$; i++)
2.3: **{**
2.4: Create the initial core set CS_i from the subset S_i, according to Eq.(4) of Definition 4;
2.5: NumOfClusters[i] ← SSynC(CoreSet CS_i, float δ_i, float ε); /* The initial value of the core set CS_i is the input of SSynC algorithm. After finished the clustering procedure, core set CS_i records the clustering result of subset S_i. Here, NumOfClusters[i] is used to record the return value of SSynC algorithm. */
2.6: **}**
3 **Step3:** Create a new core set, *CS*, by collecting all root cores from the above clustering results of m subsets. /* Here, m subsets are $\{S_1, S_2, \cdots, S_m\}$. */
4 **Step4:** Clustering the collected core set *CS* using SSynC algorithm. That is:
4.1: int FinalNumOfClusters ← SSynC(CoreSet *CS*, float δ_{merge}, float ε); /* FinalNumOfClusters is used to record the final number of clusters of *CS*. */
/* After the collected core set *CS* is operated by SSynC algorithm, the paths of some inactive cores are compressed just like the joint-set method such that the largest height of leaf cores is less than or equal to two (note: the height of root cores is one). */
5 **Step5:** The final root cores of the core set *CS* operated by SSynC algorithm represent the clusters or isolates of the data set $S = \{X_1, X_2, \cdots, X_n\}$. Suppose finally the set *CS* has K root cores, we think the data set $S = \{X_1, X_2, \cdots, X_n\}$ has K clusters or isolates. The location of the root core that represents some cores is regarded as their cluster center, and the location of the root core that represents only one or several cores is regarded as the final synchronization location of one or several isolates.

Note: In **Table 2**, Step1 and Step2 belong to "divide stage", and Step3 and Step4 belong to "collect stage".

4.4 The recursive algorithm of MLSynC method

Here, we present a description of the recursive algorithm of MLSynC method.

Algorithm name: The recursive algorithm of MLSynC method

Input: Data set $S = \{X_1, X_2, \cdots, X_n\}$, dissimilarity measure $dis(\cdot, \cdot)$, parameter m, a setting set of parameter δ, and parameter ε.

Output: The clustering result of the data set $S = \{X_1, X_2, \cdots, X_n\}$.

The main procedure of the recursive algorithm of MLSynC method is described by **Table 3**.

Table 3

The main procedure of the recursive algorithm of MLSynC method.

1 **Step1:** Create an initial core set *InitCS* = $\{C_1, C_2, \cdots, C_n\}$ from the data set $S = \{X_1, X_2, \cdots, X_n\}$ according to Eq.(4) of Definition 4.
2 **Step2:** Call the dichotomy recursive function, MLSynC_Recursion, which is described by the following:
/* MLSynC_Recursion is a dichotomy recursive function that can operate some huge data sets that are stored in in-memory or disk. *InitCS* is the core set used as the input of this algorithm, which represents the input data that may be loaded into in-memory step by step from disk. *First* is the label or index of the first record in *InitCS*, and *Last* is the label or index of the last record in *InitCS*. *ResultCS* that represents clusters and isolates will be used to store the clustering results of *InitCS*. The return value of this function records the number of clusters and isolates. */
2.1: int MLSynC_Recursion (InputData *InitCS*, int *First*, int *Last*, OutputData *ResultCS*)
2.2: **{**
2.3: **if** ((*Last* - *First*) > *FitNumber*) /* Parameter *FitNumber* is a predefined threshold or a maximum that the computer system can operate the loaded data in its in-memory directly. */
2.4: **{**
2.5: int *MidLocation* ← Divide_*InitCS* (*InitCS, First*, *Last*); /* The function, Divide_*InitCS*, will partition *InitCS* into two parts. The return value of the function records the middle

location of *InitCS*. */
2.6: int *NumberCS1* ← MLSynC_Recursion (*InitCS*, *First*, *MidLocation*, *OutputCS1*);
2.7: int *NumberCS2* ← MLSynC_Recursion (*InitCS*, *MidLocation* + 1, *Last*, *OutputCS2*);
2.8: InputData *NewInputCS* ← *OutputCS1* ∪ *OutputCS2*; /* *NewInputCS* can be obtained by connecting *OutputCS1* and *OutputCS2* directly. */
2.9: int *ResultNumberCS* ← SSynC (*NewInputCS*, *NumberCS1* + *NumberCS2*, *ResultCS*);
/* SSynC algorithm is used to clustering *NewInputCS*. *ResultCS* that represents clusters and isolates will store the clustering results of *NewInputCS*. The return value of the function records the number of clusters and isolates. */
2.10: **}**
2.11: **else**
2.12: int *ResultNumberCS* ← SSynC (*InitCS*, *Last* - *First*, *ResultCS*); /* *InitCS* is operated by SSynC algorithm directly. */
2.13: **return** *ResultNumberCS*;
2.14:**}**
3 **Step3:** The parameter in the function MLSynC_Recursion of **Step2**, *ResultCS*, represents the clusters or isolates of the data set $S = \{X_1, X_2, \ldots, X_n\}$.

4.5 The comparison of the dynamic clustering processes among SynC algorithm, ESynC algorithm, SSynC algorithm, and MLSynC method

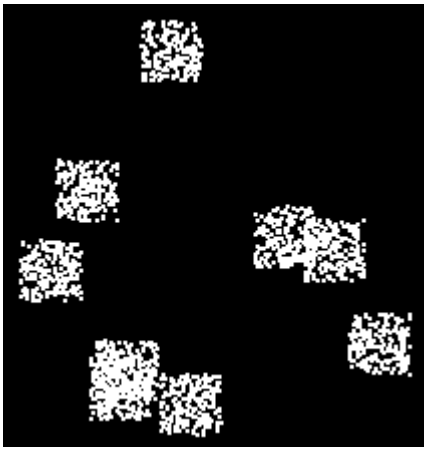
SynC algorithm uses the extensive Kuramoto model described by Eq.(2) at each step evolution that is a nonlinear renewal model. ESynC algorithm uses the linear version of Vicsek model described by Eq.(3) at each step evolution that is a linear renewal model. SSynC algorithm uses the linear weighted Vicsek model described by Eq.(5) at each step evolution that is a linear weighted renewal model. MLSynC method uses a framework of "divide and collect" and the linear weighted Vicsek model described by Eq.(5).

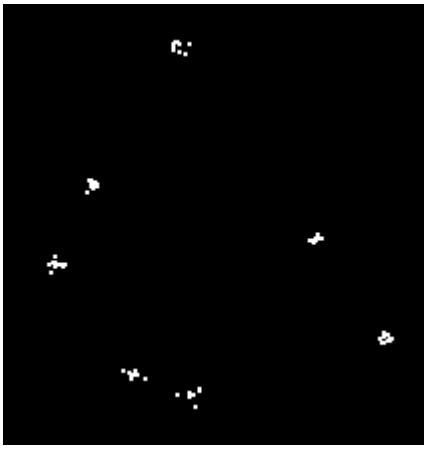
Figure 2 compares the tracks of 2000 data points from DS0 among the clustering processes of SynC algorithm, ESynC algorithm, SSynC algorithm, and MLSynC method. From Figure 2, we observe that MLSynC method, ESynC algorithm, and SSynC algorithm have better local synchronization effect than SynC algorithm.

Figure 3 (a) compares a measure index of clustering result, the cluster order parameter with *t*-step evolution (*t*: 0 - 49) (Böhm et al., 2010), among SynC algorithm, ESynC algorithm, SSynC algorithm, and MLSynC method. **Figure 3** (b) compares another measure index of clustering result, the *t*-step average length of edges (*t*: 0 - 49) (Chen, 2017), among SynC algorithm, ESynC algorithm, SSynC algorithm, and MLSynC method. And Figure (c) compares the relation between the final number of clusters and parameter δ after finished clustering using the four algorithms respectively.

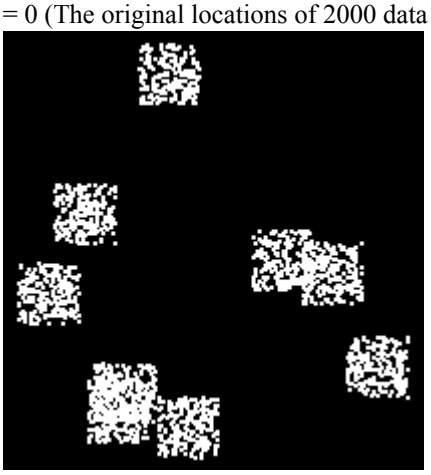
From **Figure 3** (a) and (b), we observe that the *t*-step average length of edges is better than the cluster order parameter with *t*-step evolution in measuring the final

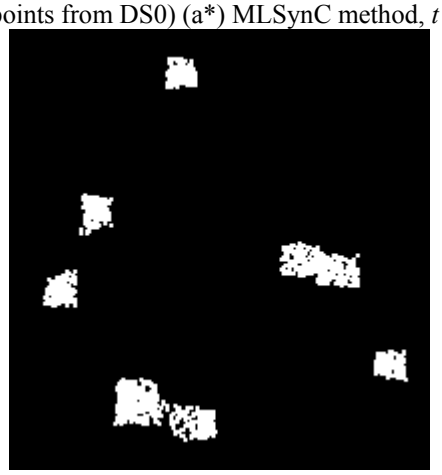
synchronization results. From **Figure 3** (c), we observed that parameter δ has a long valid interval in ESynC algorithm, SSynC algorithm, and MLSynC method. From **Figure 3** (c), we also observe that the smaller parameter δ is set in SynC, ESynC, SSynC, and MLSynC, the larger the final number of clusters is. In many data sets with obvious clusters, if we use a proper partitioning method to group the data set, MLSynC method can often get the correct number of clusters when parameter δ chooses any value from its valid interval. ESynC algorithm and SSynC algorithm can often get the correct number of clusters when parameter δ chooses any value from its valid interval. But the final number of clusters using SynC algorithm is often much larger than the actual number of clusters whenever parameter δ gets any value in a long interval.



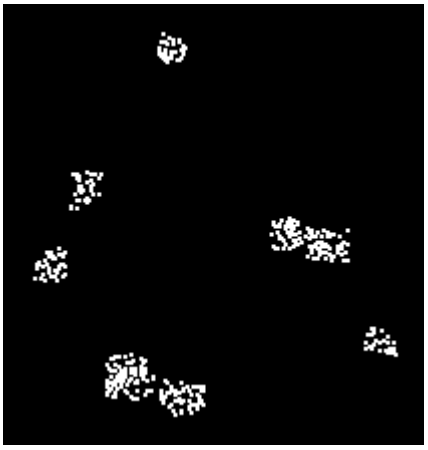


(a) $t = 0$ (The original locations of 2000 data points from DS0) (a*) MLSynC method, $t = 0$

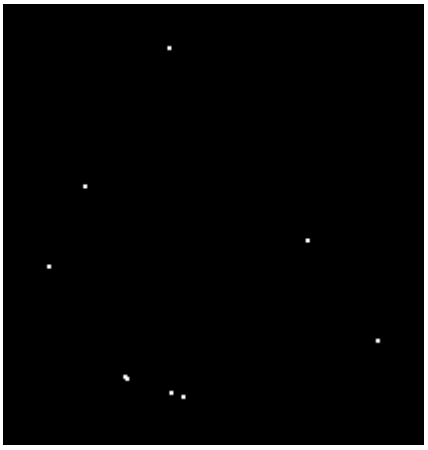




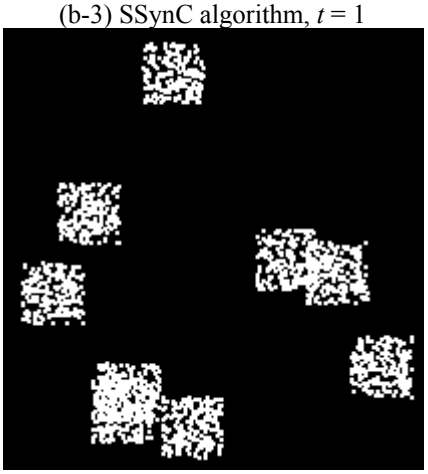
(b-1) SynC algorithm, $t = 1$ (b-2) ESynC algorithm, $t = 1$



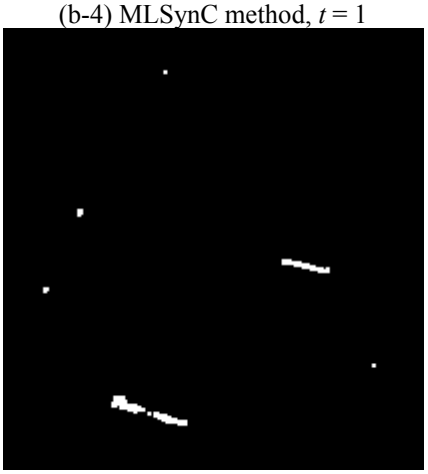
(b-3) SSynC algorithm, $t = 1$



(b-4) MLSynC method, $t = 1$

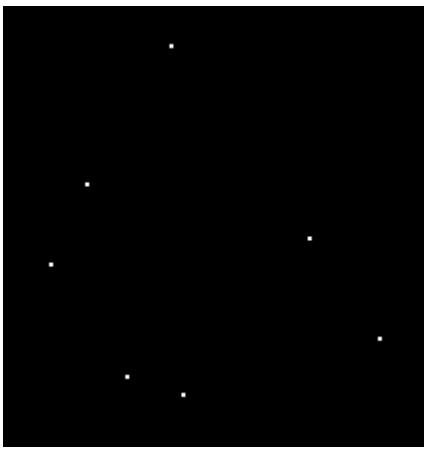


(c-1) SynC algorithm, $t = 2$

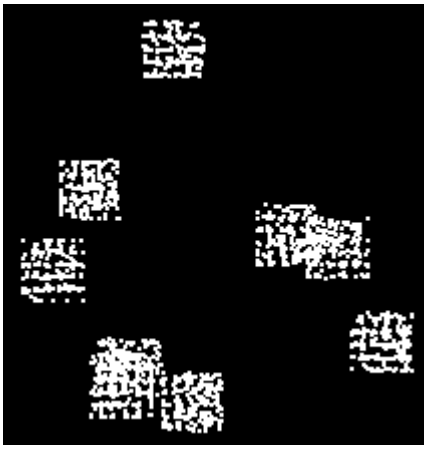


(c-2) ESynC algorithm, $t = 2$

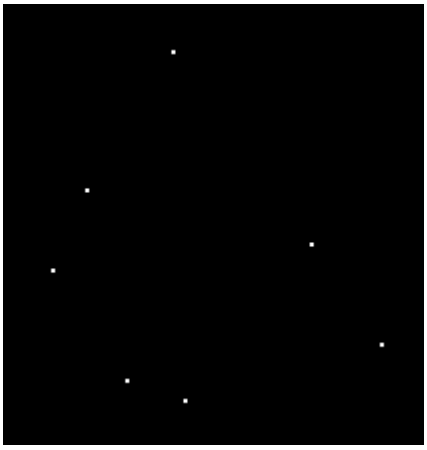
(c-3) SSynC algorithm, $t = 2$



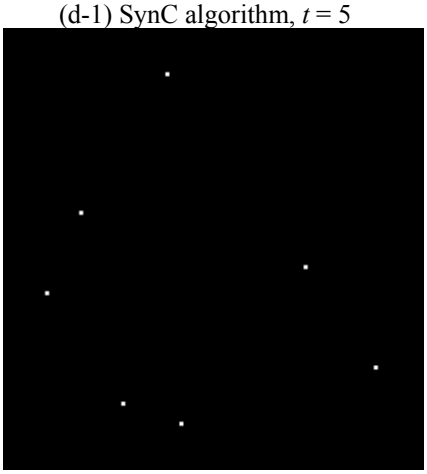
(c-4) MLSynC method, $t = 2$



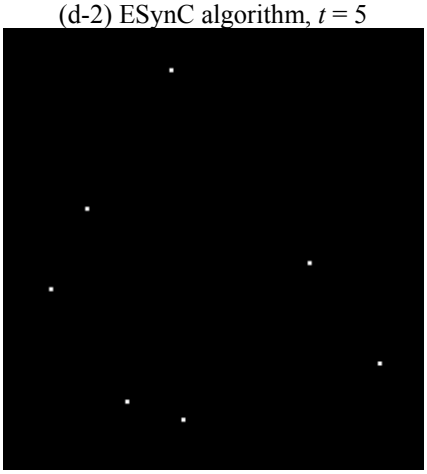
(d-1) SynC algorithm, $t = 5$



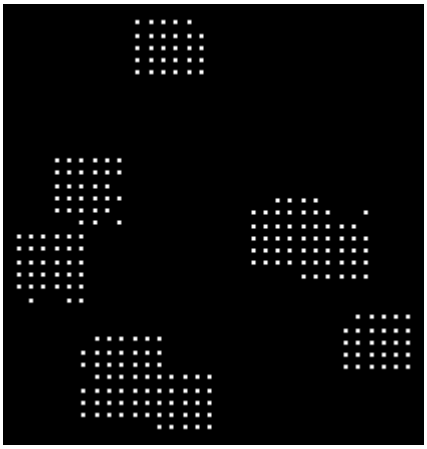
(d-2) ESynC algorithm, $t = 5$



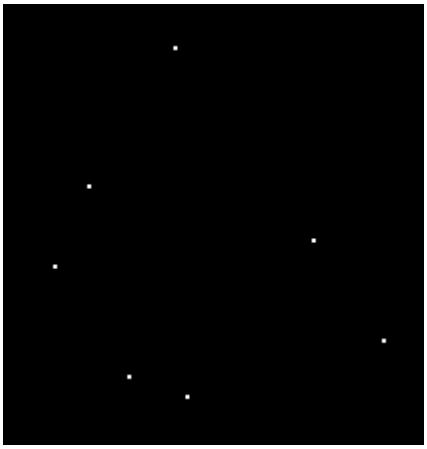
(d-3) SSynC algorithm, $t = 5$



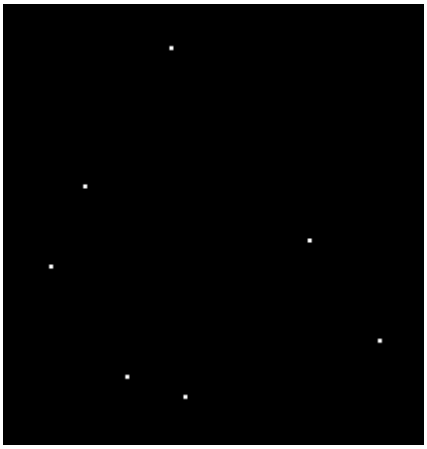
(d-4) MLSynC method, $t = 5$



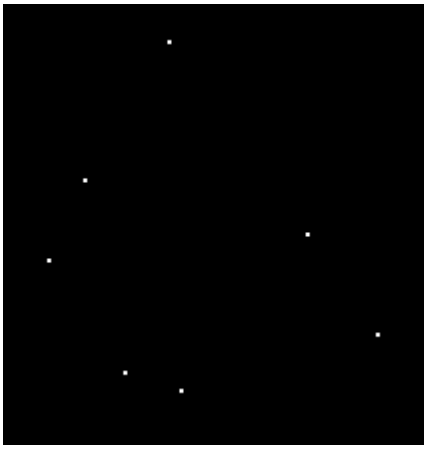
(e-1) SynC algorithm, $t = 45$



(e-2) ESynC algorithm, $t = 45$

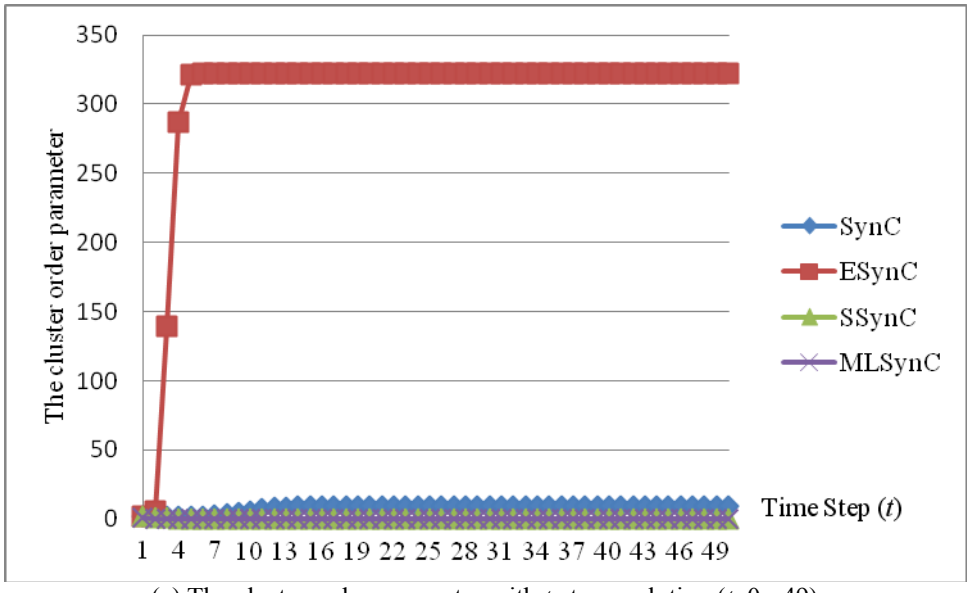


(e-3) SSynC algorithm, $t = 45$

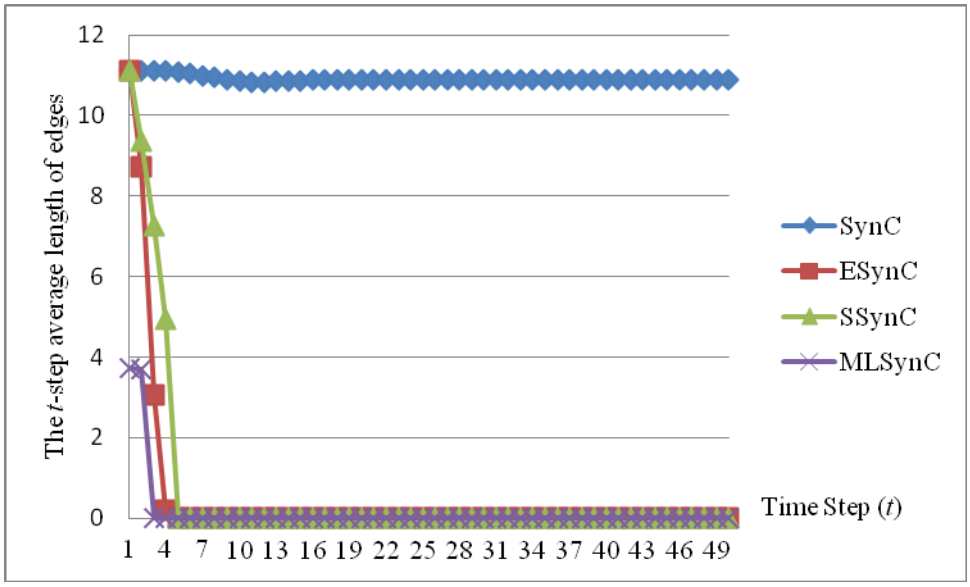


(e-4) MLSynC method, $t = 45$

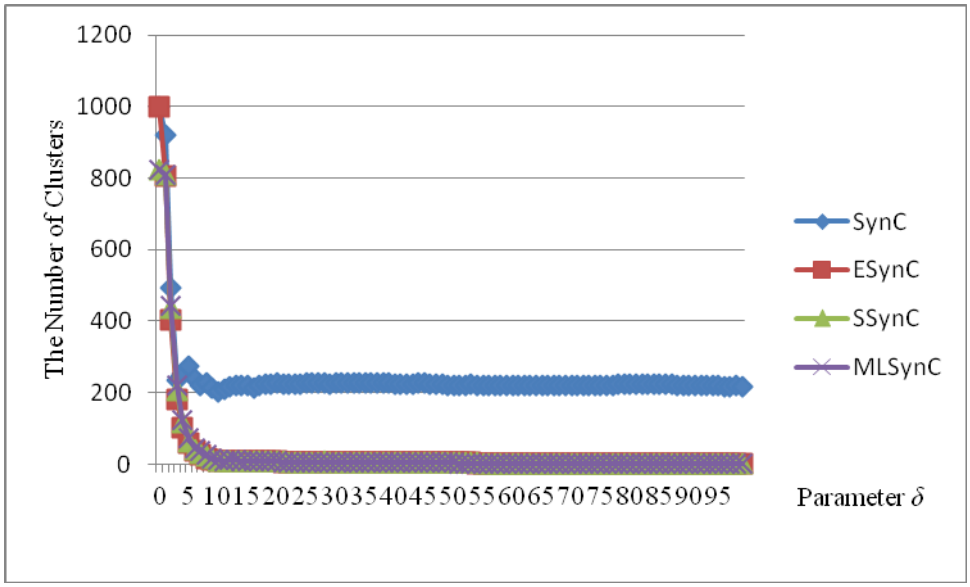
Fig. 2 The comparison of the dynamical clustering processes with time evolution among SynC algorithm, ESynC algorithm, SSynC algorithm, and MLSynC method. From (a) to (e) of Fig. 2, the data set is from DS0 with 2000 points, parameter δ is set as 18 in the four algorithms, and parameter ε is set as 1 in SSynC algorithm and MLSynC method. In MLSynC method, parameter m is set as 10, and the two-level framework algorithm is used. Fig. 2 (a*), (b-4), (c-4), (d-4), and (e-4) are the evolution displays in the "collect stage" of the two-level framework algorithm of MLSynC method.



(a) The cluster order parameter with t-step evolution (t: 0 - 49)



(b) The t-step average length of edges (t: 0 - 49)



(c) The relation between the final number of clusters and parameter δ (δ: 0 - 99) among four synchronization algorithms.

Fig. 3 The comparison of SynC algorithm, ESynC algorithm, SSynC algorithm, and MLSynC method in two measure indexes of clustering result and the relation between the final number of clusters and parameter δ. In Fig. 3, the data set is from DS0 with 2000 points, and parameter ε is set as 1 in SSynC algorithm and MLSynC method. In MLSynC method, parameter m is set as 10, the two-level framework algorithm is used. In Fig. 3 (a) and (b), parameter δ is set as 18 in the four algorithms. In MLSynC method of Fig. 3 (a) and (b), two indexes (The cluster order parameter with t-step evolution and the t-step average length of edges) are computed in the "collect stage" of the two-level framework algorithm of MLSynC method.

4.6 Time and space complexity analysis of MLSynC method

From the time complexity analysis of SSynC algorithm that presented in another paper, we know that SSynC algorithm needs Time = $O(d\cdot(n_{(t=0)}{}^2 + n_{(t=1)}{}^2) + \ldots + n_{(t=T\text{-}1)}{}^2)) < O(Tdn^2)$, which is usually less than SynC algorithm and ESynC algorithm.

Here T is the times of synchronization in SSynC algorithm and $n_{(t)}$ is the number of active cores in the t-step synchronization evolution.

4.6.1 Time and space complexity analysis of the two-level framework algorithm of MLSynC method

In the two-level framework algorithm of MLSynC method, Step1 needs Time = $O(n)$ and Space = $O(n)$.

The time cost of Step2 is:

$$\text{Time} = O\left(d \cdot \sum_{i=1}^{m}\left(n_i^2(t=0)+n_i^2(t=1)+\ldots+n_i^2(t=T_i-1)\right)\right) \approx O(d\cdot(\check{n}_{(t=0)}{}^2+\check{n}_{(t=1)}{}^2+\ldots +\check{n}_{(t=MaxT\text{-}1)}{}^2)/m). \quad (10)$$

In Eq. (10), T_i is the synchronization times in the i-th clustering module based on SSynC algorithm, $n_i(t = 0)$ is the initial number of active cores in the i-th clustering module, $\check{n}_{(t\,=\,0)}$ is the initial average number of active cores in the m clustering modules, and $MaxT$ is the max synchronization times in the m clustering modules.

Suppose subset S_i (i = 1, 2, …, m) has K_i clusters or isolates, then Step3 needs Time = $O(K_1 + K_2 + \ldots + K_m) = O(|CS|)$. Here $|CS|$ is the number of elements in the core set CS.

Step4 needs Time = $O(d\cdot((|CS|_{(t=0)})^2 + (|CS|_{(t=1)})^2 + \ldots + (|CS|_{(t=T\text{-}1)})^2))$. Here T is the synchronization times using SSynC algorithm in Step 4 and $|CS|_{(t)}$ is the number of active cores in the t-step synchronization evolution.

Step5 needs Time = $O(n)$ and Space = $O(n)$.

4.6.2 Time and space complexity analysis of the recursive algorithm of MLSynC method

In the recursive algorithm of MLSynC method, Step1 needs Time = $O(n)$ and Space = $O(n)$.

In many cases, the time cost of Step2 is:

$$T(n) = \begin{cases} O(d\cdot(n_{(t=0)}^2+n_{(t=1)}^2+\ldots+n_{(t=T-1)}^2)) & \text{if} \quad n \le FitNumber, \\ O(1)+2\cdot T(n/2) & \text{if} \quad n > FitNumber. \end{cases} \quad (11)$$

Step3 needs Time = $O(n)$ and Space = $O(n)$.

According to our analysis, usually MLSynC method needs less time than SSynC algorithm.

4.7 The setting of parameters δ, ε, m and in MLSynC method

For each subset or collected core set in MLSynC method, SSynC algorithm is used for clustering with proper values of parameter δ and parameter ε. In Böhm et al.

(2010), parameter δ is optimized by the MDL principle. In Chen (2015), two other methods were presented to estimate parameter δ. Here, we can also select a proper value for parameter δ according to Theorem 1 and Property 1.

5.7.1 The setting of parameter δ in MLSynC method

(1) The optimization of parameter δ in Böhm et al. (2010)

Parameter δ can affect the results of clusters. In Böhm et al. (2010), parameter δ can be optimized by a heuristic method and the MDL principle. In the heuristic method presented by Böhm et al. (2010), parameter δ is initialized with the average value of the k-nearest neighbor distance determined from the data set for a small k. For example, $k = 3$ is recommended in their experiments. Then parameter δ is increased with a reasonable step size. For example, the step size is recommended by the difference between the average (k+1)-nearest neighbor distance and the average k-nearest neighbor distance.

The proper value of parameter δ is determined by minimizing the total coding cost $L(S, M)$ of a clustering model M. Here, $L(S, M) = L(S|M) + L(M)$.

$L(S|M)$ is denoted by the following equation:

$$L(S|M) = -\sum_{k=1}^{K} \sum_{X \in C_k} \log\big(P(X)\big), \tag{12}$$

where $P(X)$ is the probability of point X assigned to the k-th cluster C_k, S is the data set, and K is the number of clusters.

And the cost for coding the clustering model M, $L(M)$, is denoted by the following equation:

$$L(M) = \sum_{k=1}^{K} \sum_{j=1}^{|C_k|} \log\left(\frac{n}{|C_k|}\right) + \sum_{k=1}^{K} \frac{d}{2} \log\big(|C_k|\big), \tag{13}$$

where M is the clustering model, C_k is the k-th cluster, n is the number of points in the data set, and d is the number of dimensions.

(2) The heuristic selection of parameter δ in Chen (2017a)

In Chen (2017a), parameter δ can be selected by the following heuristic equation:

$$\max\{longthestEdgeInMst(cluster_k) \mid k = 1, 2, \ldots, K\} \le \delta < \min\{dis(cluster_i, cluster_j) \mid i, j = 1, 2, \ldots, K\}, \tag{14}$$

where $longthestEdgeInMst(cluster_k)$ is the weight of the longest edge in the minimum spanning tree of the k-th cluster, $dis(cluster_i, cluster_j)$ is the weight of the minimum

edge connecting the i-th cluster and the j-th cluster, and K is the number of clusters in the final synchronization step.

(3) A linear-searching exploring method of parameter δ

Usually, parameter δ has a very long valid range for many kinds of data sets. Some simulated experiments in Chen (2017a) and this paper also validate this conclusion. Some times, parameter δ have several long valid ranges for different clustering levels. So we can explore the valid range of parameter δ by the linear-searching method.

5.7.2 The setting of parameter ε in MLSynC method

Parameter ε affects the time cost of MLSynC method slightly. Usually, parameter ε has a long valid interval. For example, if parameter $\delta > 15$, then the valid interval of parameter ε is about in (0, 10]. In simulations, we almost get the same results for several different values (such as 0.00001, 0.0001, 0.001, 0.01, 0.1, 1, and 10) of parameter ε.

5.7.3 The setting of parameter m in MLSynC method

In the two-level framework algorithm of MLSynC method, parameter m affects its whole time cost. If parameter m is set too large, then the number of points in each part of the data set becomes less. So SSynC algorithm in the two-level framework algorithm of MLSynC method needs less clustering time for each subset, and the two-level framework algorithm of MLSynC method needs much divide and collect time. Usually, there is a balance between the increase of "divide and collect" time and the decrease of clustering time with the increase of parameter m. So we think that parameter m cannot be set too large.

4.8 The improvement of MLSynC method

In MLSynC method, one improved version of SSynC algorithm can be obtained by combining multidimensional grid partitioning method and Red-Black tree structure to construct the near neighbor point sets of all active cores. The improving method that can decrease its time cost is introduced in Chen (2018). Generally, we first partition the data space of the data set $S = \{X_1, X_2, \ldots, X_n\}$ by using a kind of multidimensional grid partitioning method. Then design an effective index of all grid cells and construct δ near neighbor grid cell set for each grid cell. If every grid cell uses a Red-Black tree to index its active cores in each synchronization step, then constructing the δ near neighbor point set for every active core will become quicker when the number of grid cells is proper.

Before synchronization iterative evolution, if we set a proper value for parameter δ to filtrate isolates, then these isolates can be set as inactive cores that will not be operated in the next iterative evolution. This improvement of implementation technique is often effective in some data sets.

4.9 The convergence of MLSynC method

In MLSynC method, SSynC algorithm is used to clustering each subset and the collected core set. So the convergence of MLSynC method is completely depended on the convergence of SSynC algorithm. According to the convergence analysis of SSynC algorithm and our simulations, we know that MLSynC method is also convergent.

5. Simulated experiments

5.1 Experimental design

Our experiments are finished on a personal computer (Capability Parameters: Pentium(R) Dual-Core CPU T4500 2.30 GHz, 2G Memory). Experimental programs are developed using C and C++ language under Windows XP.

To verify the improvement in clustering effect and time cost of MLSynC method, there will be some simulated experiments of some artificial data sets, eight UCI data sets (Frank et al., 2010), and three bmp pictures in the next three subsections.

Four kinds of artificial data sets (DS1 - DS4) are produced in a 2-D region [0, 600] × [0, 600] by a program presented in Online Resource 3 of Supplementary Material of this paper. Other kinds of artificial data sets (DS5 - DS16) are produced in a interval [0, 600] in each dimension by a similar program. DS0 is produced in a 2-D region [0, 200] × [0, 200] by a similar program. Iris et al. (Frank et al., 2010) are eight UCI data sets that used in our experiments. Three bmp pictures (named as Picture1, Picture2, and Picture3) are obtained from Internet. **Table 4** is the description of the experimental data sets.

Table 4 The description of experimental data sets

(a) The description of sixteen kinds of artificial data sets

Data Sets	Number of Clusters	With Noise	Cluster Semidiameter	Dimension (*d*)
DS0	9	no	30	2
DS1	5	yes	40	2
DS2	5	no	50	2
DS3	9	yes	30	2
DS4	9	no	40	2
DS5	12	no	30	2
DS6	12	no	30	4
DS7	12	no	30	6
DS8	12	no	30	8
DS9	5	no	30	2
DS10	5	no	30	4
DS11	5	no	30	6
DS12	5	no	30	8
DS13	5	no	30	20
DS14	5	no	30	40
DS15	5	no	30	80
DS16	5	no	30	100

(b) The description of eight UCI data sets (Frank and Asuncion, 2010)

UCI Data Sets	Number of Points (*n*)	Dimension (*d*)	Number of Classes
Iris	150	4	3
Wine	178	13	3
Wdbc	569	30	2
Glass	214	9	6
Ionosphere	351	34	2
Letter-recognition	20000	16	26
Segmentation	210	19	7
Cloud	2048	10	2

(c) The description of three bmp picture data sets (obtained from Internet)

Picture Data Sets	Number of Pixels (*n*)	Dimension (*d*)
Picture1	200*200	3
Picture2	200*200	3
Picture3	200*200	3

In our simulated experiments, the maximum times of synchronization evolution in the while repetition of SynC algorithm, ESynC algorithm, SSynC algorithm, and MLSynC method is set as 50. MLSynC method is implemented using the two-level framework algorithm of MLSynC method.

The comparison results of these clustering algorithms are presented by six figures (**Figs. 3 - 8 of Supplementary Material**) and nine tables (**Tables 1 - 9 of Supplementary Material**). Because of the limited pages, we only select two figures from **Figs. 3 - 8 of Supplementary Material** in the manuscript. All six figures are presented in Online Resource 4 of Supplementary Material. All nine tables are presented in Online Resource 5 of Supplementary Material. And performance of

algorithms is measured by time cost (second). Clustering quality of algorithms is measured by display figures of clustering results and two robust information-theoretic measures, Adjusted Mutual Information (AMI) (Vinh et al., 2010) and Normalized Mutual Information (NMI) (Strehl et al., 2002). According to Vinh et al. (2010), the higher the value of the two measures gets, the better the clustering quality of algorithms is. In simulations, we use the Matlab code from Vinh et al. (2010) to compute the two clustering quality measures.

In section 5.2, MLSynC method will be compared with SynC algorithm, ESynC algorithm, SSynC algorithm, and some other classic clustering algorithms (K-Means (MacQueen, 1967), FCM (Bezdek, 1981), AP (Frey et al., 2007), DBSCAN (Ester et al., 1996), Mean Shift (Fukunaga et al., 1975; Comaniciu et al., 2002) in clustering quality and time cost using some artificial data sets.

In section 5.3, MLSynC method will be compared with SynC algorithm, ESynC algorithm, SSynC algorithm, and some other classic clustering algorithms in clustering quality and time cost using eight UCI data sets.

In section 5.4, MLSynC method will be compared with SynC algorithm, ESynC algorithm, SSynC algorithm, and some other classic clustering algorithms in compressed effect of clustering results, clustering quality, and time cost using three bmp pictures.

In the experiments, parameter δ used in SynC algorithm, ESynC algorithm, SSynC algorithm, MLSynC method, DBSCAN algorithm, and Mean Shift algorithm is the threshold of Definition 1. In DBSCAN algorithm, parameter $MinPts$ = 4, and parameter Eps is the same as parameter δ.

The detailed discussion on how to construct δ near neighbor point sets is described in Chen (2013). How to select a proper value for parameter δ in SynC algorithm is discussed in Böhm et al. (2010). ESynC algorithm, SSynC algorithm, and MLSynC method use Theorem 1 and Property 1 to select a proper value for parameter δ. In SSynC algorithm and MLSynC method. In SSynC algorithm and MLSynC method, parameter ε has trivial effect in time cost and clustering results.

To simplify our simulation of the two-level framework algorithm of MLSynC method, we set the same value for parameter δ_1, δ_2, $\cdots$, δ_m, and δ_{merge}. So they are also named as δ.

5.2 Experimental results of some artificial data sets (from DS1 - DS16)

5.2.1 The comparison of the clustering results among SynC algorithm, ESynC

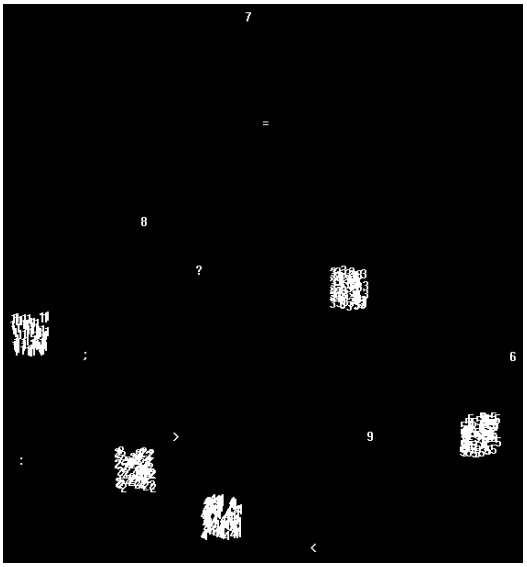
algorithm, SSynC algorithm, and MLSynC method (from DS1- DS4)

Table 1 of Supplementary Material presents the comparison results of four synchronization clustering algorithms (SynC, ESynC, SSynC, and MLSynC) by using four artificial data sets (from DS1 - DS4). In **Table 1 of Supplementary Material**, by intercomparing SynC, ESynC, SSynC, and MLSynC, we observe that MLSynC is the fastest clustering algorithm. At the same time, MLSynC, SSynC, and ESynC can get better local synchronization results than SynC in the four data sets.

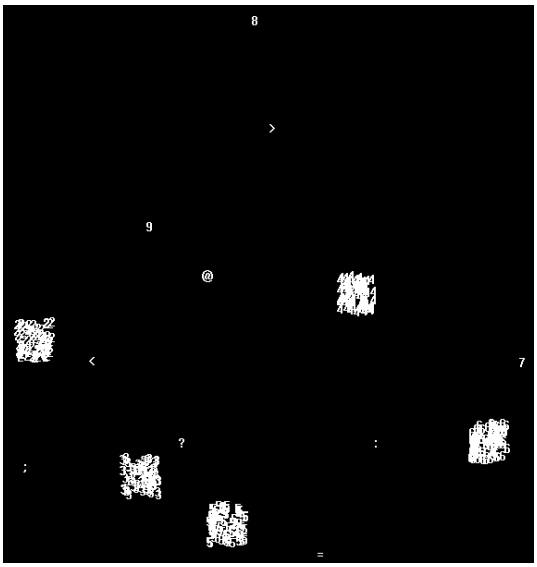
In the two-level framework of MLSynC method, parameter m is set as 10. If we use two different partitioning methods (a near unevenly partitioning method and a near evenly partitioning method) in the four data sets, the two sequences of the number of clusters of 10 subsets in each data set are different. The comparison results are presented in **Table 2 of Supplementary Material**. From **Table 2 of Supplementary Material**, we observe that two different partitioning methods result in different clustering distributions of 10 subsets in each data set.

5.2.2 The comparison of the clustering results among SynC algorithm, ESynC algorithm, SSynC algorithm, MLSynC method, and some classical clustering algorithms (from DS1 - DS8)

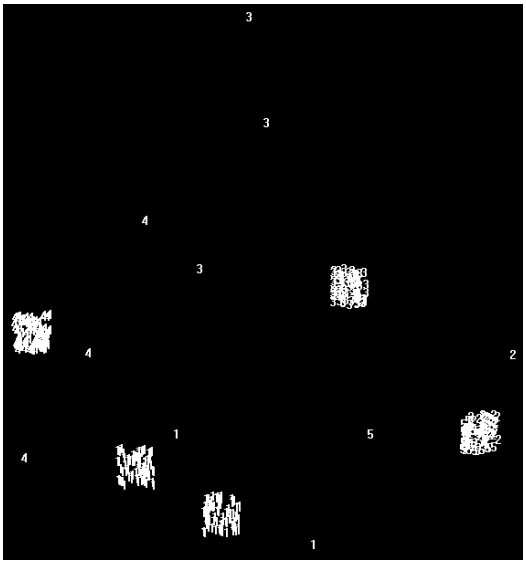
Table 3 of Supplementary Material presents the clustering quality of several clustering algorithms (SynC, ESynC, SSynC, MLSynC, and some classical clustering algorithms) by using six kinds of artificial data sets (DS2, DS4, DS5, DS6, DS7, and DS8). When computing the two information-theoretic measures (NMI and AMI), the predefined cluster labels of the eight artificial data sets are used in the true_mem that is an input file of the MATLAB code (Vinh et al., 2010). In **Table 3 of Supplementary Material**, by intercomparing MLSynC, SynC, ESynC, SSynC, and some classical clustering algorithms, we observe that MLSynC can get acceptable and similar clustering results with SSynC and ESynC in the eight data sets if the partitioning of the data set in MLSynC method is near evenly. Because the two data sets (DS4 (n = 1000) and DS5 (n = 12000)) have two connected clusters, MLSynC, SSynC, and ESynC do not get the largest values of NMI and AMI. We also observe that the partitioning method of data set can affect the clustering results of MLSynC method.



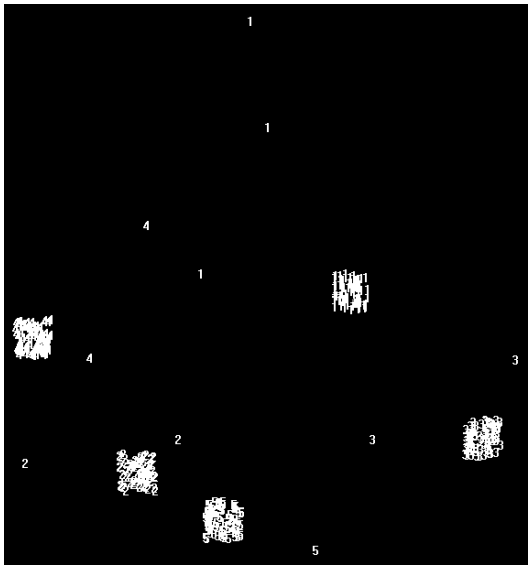
(a) Clusters identified by MLSynC (15 clusters or isolates)



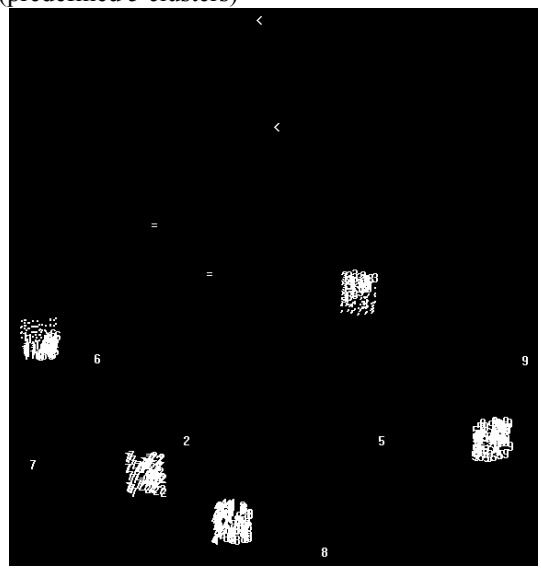
(b) Clusters identified by ESynC and SSynC (15 clusters or isolates)



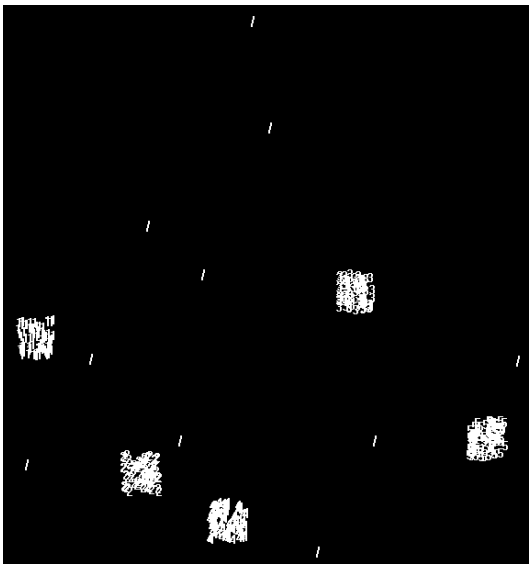
(c) Clusters identified by K-Means (predefined 5 clusters)



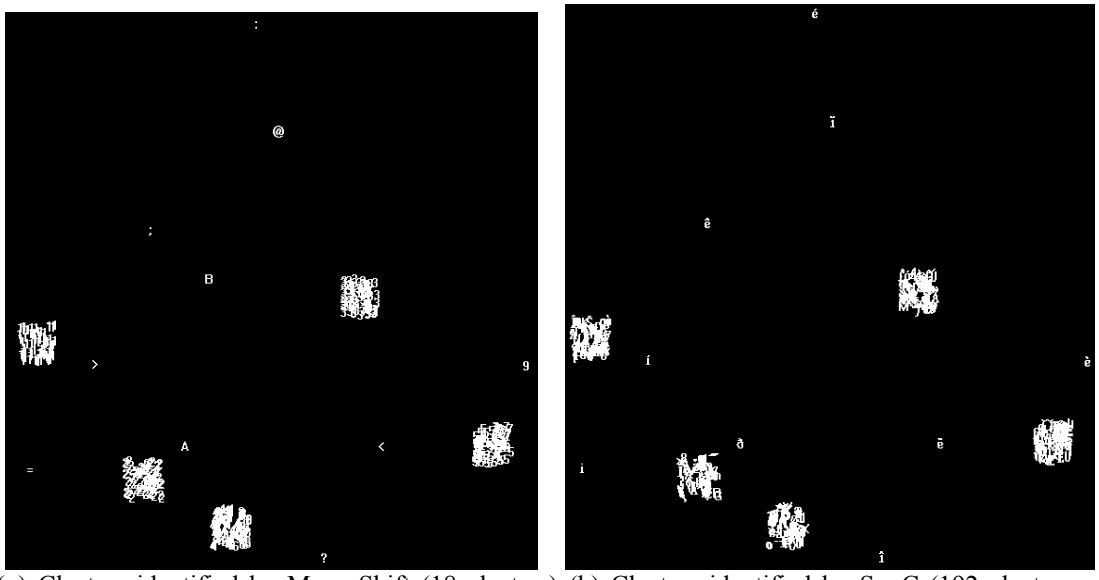
(d) Clusters identified by FCM (predefined 5 clusters)



(e) Clusters identified by AP (13 clusters)



(f) Clusters identified by DBSCAN (5 clusters)



(g) Clusters identified by Mean Shift (18 clusters) (h) Clusters identified by SynC (192 clusters or isolates)

Fig. 4. The comparison of the clustering results of several algorithms (DS1, n = 400). In Fig. 4, parameter δ = 18 in SynC, ESynC, SSynC, DBSCAN, Mean Shift, and MLSynC method; the number of data points n = 400; parameter ε = 1 in SSynC algorithm and MLSynC method. In MLSynC method, parameter m is set as 2, and the two-level framework algorithm is used.

Figure 4 and **Figs. 4 - 6 of Supplementary Material** present the comparison clustering results of several clustering algorithms by some display figures that reflect the clustering quality clearly. From **Figure 4** and **Figs. 4 - 6 of Supplementary Material**, we observe that MLSynC, SSynC and ESynC can get better clustering quality (obvious clusters or isolates displayed by figures) than SynC, AP, K-Means, and FCM in the four artificial data sets (from DS1 - DS4). Mean Shift, DBSCAN can obtain similar clustering quality (obvious clusters displayed by figures) with MLSynC, SSynC and ESynC in these artificial data sets. Especially, MLSynC, SynC, ESynC, and SSynC can all easily find some isolates if setting a proper value for parameter δ, and MLSynC can get the same clustering results (the same clusters displayed by figures) with SSynC, ESynC in these data sets.

Table 4 of Supplementary Material presents the comparison results of several clustering algorithms in time cost. In **Table 4 of Supplementary Material**, for DS1, parameter δ = 18 in SSynC, ESynC, SynC, and DBSCAN; parameter δ = 25 in MLSynC. For DS2, parameter δ = 25 in MLSynC, SSynC, ESynC, SynC, and DBSCAN. For DS3, parameter δ = 18 in MLSynC, SSynC, ESynC, SynC, and DBSCAN. For DS4, parameter δ = 18 in SSynC, ESynC, SynC, and DBSCAN;

parameter δ = 20 in MLSynC. Parameter ε = 1 in SSynC and MLSynC. In MLSynC, parameter m is set as 10, and the two-level framework algorithm is used.

In **Table 4 of Supplementary Material**, intercomparing MLSynC, SynC, ESynC, SSynC, DBSCAN, FCM, and K-Means, we observe that MLSynC is faster than SynC, ESynC, DBSCAN, and SSynC, and K-Means is the fastest clustering algorithm. Mean Shift and AP cannot run normally on a personal computer because the number of data points is set as 40000.

5.2.3 The comparison of the valid interval of parameter δ among MLSynC method, SynC algorithm, ESynC algorithm, SSynC algorithm, DBSCAN algorithm, and Mean Shift algorithm using some artificial data sets (from DS5 - DS16)

Here we compare the valid interval of parameter δ among MLSynC method, SynC algorithm, ESynC algorithm, SSynC algorithm, DBSCAN algorithm, and Mean Shift algorithm.

Table 5 of Supplementary Material presents the comparison results of the valid interval of parameter δ among MLSynC, SynC, ESynC, SSynC, DBSCAN, and Mean Shift. Here, $[e_k , e_{k+1}]$ can be obtained from Eq.(8) of Chen (2015). In **Table 5 of Supplementary Material**, intercomparing MLSynC, SynC, ESynC, SSynC, DBSCAN, and Mean Shift, we observe that although the valid interval of parameter δ in MLSynC is shorter than that in SSynC and ESynC, the valid interval of parameter δ in MLSynC is long enough in many kinds of data sets.

Table 6 of Supplementary Material compares the valid interval of parameter δ in MLSynC method for several different value of parameter ε using some artificial data sets with different dimensions. In **Table 6 of Supplementary Material**, intercomparing several different value of parameter ε, we observe that if parameter ε is less than parameter δ, the valid interval of parameter δ has very small difference for several different values of parameter ε.

5.3 Experimental results of eight UCI data sets

Because we do not know the true dissimilarity measure of these UCI data sets, all points of these UCI data sets are standardized into a interval [0, 600] in each dimension in the experiments. When computing the two information-theoretic measures (NMI and AMI), because we do not know the true cluster labels of these UCI data sets, the class labels of these UCI data sets are used in the true_mem that is an input file of the MATLAB code (Vinh et al., 2010).

5.3.1 The comparison of the clustering results among MLSynC method, SynC

algorithm, ESynC algorithm, and SSynC algorithm

Table 7 of Supplementary Material presents the comparison results of four synchronization clustering algorithms (MLSynC method, SynC algorithm, ESynC algorithm, and SSynC algorithm) by using eight UCI data sets. In **Table 7 of Supplementary Material**, intercomparing MLSynC method, SynC algorithm, ESynC algorithm, SSynC algorithm, we observe that MLSynC method, ESynC algorithm, and SSynC algorithm can get better local synchronization results than SynC algorithm in the eight UCI data sets, and MLSynC method is the fastest algorithm. From this simulated experiment, we also find that if the number of points in the data set is small and we use an uneven partition method, the difference of clustering effect between MLSynC method and SSynC algorithm is large.

5.3.2 The comparison of the clustering quality among MLSynC method, SynC algorithm, ESynC algorithm, SSynC algorithm, and some classical clustering algorithms

Table 8 of Supplementary Material presents the comparison clustering quality of several clustering algorithms (MLSynC method, SynC algorithm, ESynC algorithm, SSynC algorithm, and some classical clustering algorithms) using eight UCI data sets. In **Table 8 of Supplementary Material**, by intercomparing these clustering algorithms, we observe that MLSynC method gets the largest value of NMI and AMI in five UCI data sets (Iris, Wdbc, Glass, Segmentation, and Cloud) if it uses a sequential and uneven partitioning method. So we can say that MLSynC method often gets better clustering results than some clustering algorithms in some UCI data sets. From the final number of clusters in **Table 8 of Supplementary Material**, we observe that MLSynC method, SSynC algorithm, and ESynC algorithm can get better local synchronization results than SynC algorithm.

5.4 Experimental results of three bmp pictures

The value in RGB (Red, Green, and Blue) color space of pixel points is in a interval [0, 255] in each dimension. In **Table 9 of Supplementary Material**, **Fig. 7 of Supplementary Material**, and **Figure 5**, parameter $\varepsilon = 1$ in MLSynC method and SSynC algorithm. In MLSynC method of **Table 9 of Supplementary Material**, **Fig. 7 of Supplementary Material**, and **Figure 5**, parameter m is set as 10, and the two-level framework algorithm is used.

5.4.1 The comparison of the clustering results among SynC algorithm, ESynC

algorithm, SSynC algorithm, and MLSynC method

Table 9 of Supplementary Material presents the comparison results of four synchronization clustering algorithms (SynC, ESynC, SSynC, and MLSynC) using three pixel-point data sets from RGB color space of three bmp pictures. In **Table 9 of Supplementary Material**, by intercomparing SynC, ESynC, SSynC, and MLSynC, we observe that MLSynC is the fastest clustering algorithm. At the same time, MLSynC, SSynC, and ESynC can get better local synchronization results than SynC in these pixel-point data sets.

5.4.2 The comparison of the clustering compressed effect among MLSynC method, SynC algorithm, ESynC algorithm, SSynC algorithm, and some classical clustering algorithms

Fig. 7 of Supplementary Material lists Picture3 and its RGB space distribution of 200 * 200 pixel points.

MLSynC (final k = 45)

SSynC, ESynC (final k = 26)

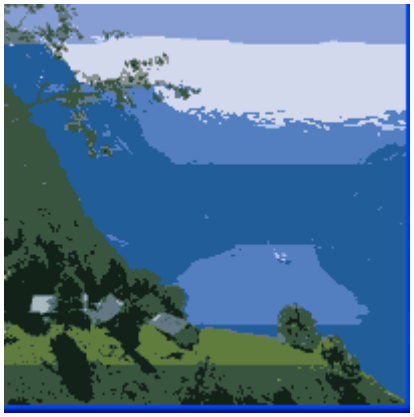
SynC (final k = 3375)

Mean Shift (final k = 19)

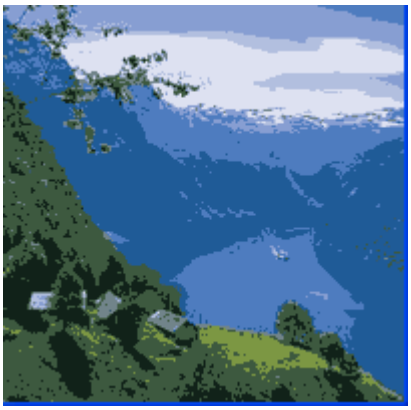
K-Means, FCM (final k = 1)

DBSCAN (final k = 148)

(a) δ = 18 in MLSynC, SynC, ESynC, SSynC, DBSCAN, and Mean Shift; predefined k (number of clusters) = 26 in K-Means and FCM.

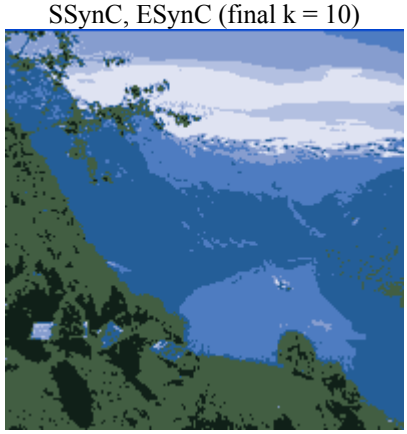


MLSynC (final k = 14)

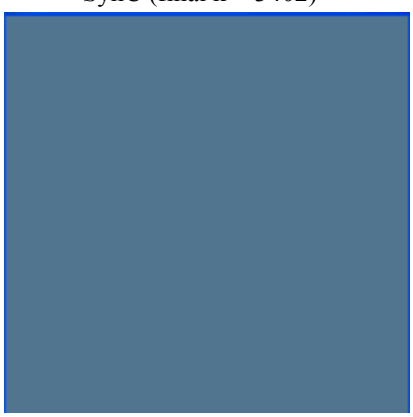


SSynC, ESynC (final k = 10)

SynC (final k = 3402)



Mean Shift (final k = 7)



K-Means, FCM (final k = 1)

DBSCAN (final k = 53)

(b) δ = 30 in MLSynC, SynC, ESynC, SSynC, DBSCAN, and Mean Shift; predefined k (number of clusters) = 10 in K-Means and FCM.

Fig. 5. The comparison of the original picture and several compressed pictures of Picture3. In Fig. 5, several compressed pictures based on different clustering algorithms are drawn by using the means of clusters that are obtained by clustering the 200 * 200 pixel points of Picture3 in RGB space.

Figure 5 lists the original picture and several compressed pictures of Picture3. In **Figure 5**, the several compressed pictures are drawn by using the means of clusters obtained by clustering the 200 * 200 pixel points of Picture3 in RGB color space using different algorithms. Because AP algorithm needs too much time and space for Picture3, this experiment does not use it. Although the queue of DBSCAN algorithm slops over, it still gets a compressed picture basing on the clustering results. From **Figure 5**, we observe that MLSynC method, ESynC algorithm, and SSynC algorithm can get multi-level clustering compressed effect for different values of parameter δ.

5.5 Analysis and conclusions of experimental results

From the comparison experimental results of these figures and tables (**Figs**. **3 - 8 of Supplementary Material** and **Tables 1 - 9 of Supplementary Material**), we observe that MLSynC method is faster than SSynC algorithm, ESynC algorithm, and SynC algorithm. We think that MLSynC method is superior to SSynC algorithm, ESynC algorithm, and SynC algorithm in time cost because of its framework of "divide and collect".

From the simulations of some artificial data sets (from DS5 - DS16), we observe that the effective interval of parameter δ in MLSynC method is enough long just like Mean Shift algorithm, DBSCAN algorithm, SSynC algorithm, and ESynC algorithm. In some cases, the effective interval of parameter δ in MLSynC method is longer than that in DBSCAN algorithm.

In some display figures, by intercomparing MLSynC method, SSynC algorithm, ESynC algorithm, and SynC algorithm, we observe that MLSynC method can explore the same clusters and isolates (displayed by some figures) with ESynC algorithm and SSynC algorithm if the partition in MLSynC method is near well-proportioned or unaided (each subset is independent with each other in the space). In many kinds of data sets, MLSynC method, SSynC algorithm, and ESynC algorithm can explore obvious clusters or isolates if selecting a proper value for parameter δ, and SynC algorithm cannot explore obvious clusters in many data sets.

From simulations of some data sets, we observe that the iterative times of SynC algorithm, AP algorithm, K-Means algorithm, and FCM algorithm is larger than that of MLSynC method, SSynC algorithm, and ESynC algorithm. In many data sets,

MLSynC method, ESynC algorithm, SSynC algorithm, Mean Shift algorithm, and DBSCAN algorithm have better ability than SynC algorithm, K-Means algorithm, FCM algorithm, and AP algorithm in exploring clusters and isolates. Specially, AP algorithm needs the longest time.

MLSynC method is an improved clustering algorithm with faster clustering speed than SSynC algorithm and ESynC algorithm almost in all cases. Usually, parameter ε has a long effective interval (for example, the effective interval of parameter ε is about (0, 10] if parameter $\delta > 15$). In simulations, we observe that if parameter ε gets some different values in its effective interval, the clustering results of MLSynC method is almost the same except the time cost.

Because the values in RGB space of the pixel points of Picture3 are almost continuous and have no obvious clusters. In this case, MLSynC method, SSynC algorithm, and ESynC algorithm can get more obvious multi-level compressed effect than some other clustering algorithms, such as K-Means algorithm and FCM algorithm. In simulations, we also observe that DBSCAN algorithm needs more space than MLSynC method, SSynC algorithm, and ESynC algorithm because of its recursive process.

Because of the limited page space, we only select some typical data sets (sixteen kinds of artificial data sets, eight UCI data sets, and three bmp picture data sets) used in our experiments. For all experimental data sets, we observe that MLSynC method improves ESynC algorithm in time cost. For other data sets, we think MLSynC method is still superior to SynC algorithm in time cost. We believe that the selection of experimental data sets is not biased.

From **Figure 1** and some other simulated experimental results, we conclude the application condition of MLSynC method. In any one of the following two cases, MLSynC method can get similar clustering effect with SSynC algorithm and ESynC algorithm.

Case 1: The spatial distribution of any partitioned data subset is almost the same as that of the original data set.

Case 2: Any two partitioned data subsets cannot be intersect, cannot be joined, or cannot be much near (less than or equal to parameter δ). In this case, any unabridged cluster of the original data set will not be partitioned into multiple near (larger than parameter δ) small clusters after the partition process of MLSynC method.

When the partition of the original data set does not satisfy any one case above,

MLSynC method will often get different clustering effect with SSynC algorithm and ESynC algorithm.

6. Conclusions

This paper presents an improved synchronization clustering method, MLSynC, which often gets better clustering results than the original synchronization clustering algorithm, SynC. From the experimental results, we observe that MLSynC method can often obtain less iterative times, faster clustering speed, and better clustering quality than SynC algorithm in many kinds of data sets.

The major contributions of this paper can be summarized as follows:

(1) It develops an effective framework of "divide and collect" in clustering field by using a linear weighted Vicsek model.

(2) It presents two concrete implementations of MLSynC method, a two-level framework algorithm and a recursive algorithm.

(3) It validates the improved effect of MLSynC method in time cost and clustering quality by some simulated experiments.

MLSynC method is also robust to outliers and can find obvious clusters with different shapes. The number of clusters does not have to be fixed before clustering. Usually, parameter δ has some valid interval that can be determined by using an exploring method listed in Chen (2015), the heuristic method described by Theorem 1 and Property 1 presented in Chen (2017), or using the MDL-based method presented in Böhm et al. (2010).

MLSynC method has some similarities with MapReduce framework. So we can also say it is an application example of MapReduce framework in clustering field.

This work opens some possibilities for further improvement and investigation. First, further improve MLSynC method in time cost. For example, designing similarity-preserving hashing function that needs $O(1)$ time complexity is valuable in the process of constructing δ near neighbor point set. Second, extend the applicability and explore the clustering effect of our algorithms in high-dimensional data. Third, implement MLSynC method on a cluster with a parallel programming model or on MapReduce framework.

Acknowledgments

This work was supported by Chongqing Cutting-edge and Applied Foundation Research Program of China (grant numbers cstc2016jcyjA0521, cstc2016jcyjA0063);

Chongqing Municipal Key Laboratory of Institutions of Higher Education (grant number [2017]3); Program of Chongqing Development and Reform Commission (grant number 2017[1007]); and Chongqing Three Gorges University of China (grant number 16PY08).

References

Agrawal, R., Gehrke, J., & Gunopolos, D., et al. (1998). Automatic subspace clustering of high dimensional data for data mining applications. In *Proceedings of ACM SIGMOD* (pp. 94-105).

Ankerst, M., Breunig, Markus M., Kriegel, Hans-Peter., & Sander, Jörg. (1999). OPTICS: Ordering points to identify the clustering structure. In *Proceedings of ACM SIGMOD* (pp. 49-60).

Bezdek, J. C. (1981). Pattern recognition with fuzzy objective function algorithms. New York, Plenum Press.

Bouguettaya, A., Yu, Q., & Liu, X., et al. (2015). Efficient agglomerative hierarchical clustering. *Expert Systems with Applications*, *42*(5), 2785-2797.

Böhm, C., Plant, C., & Shao, J., et al. (2010). Clustering by synchronization. In *Proceedings of ACM SIGKDD* (pp. 583-592).

Cao, F., Huang, J. Z., & Liang, J. (2017). A fuzzy SV-k-modes algorithm for clustering categorical data with set-valued attributes. *Applied Mathematics and Computation*, 297, 1-15.

Chen, X. (2013). Clustering based on a near neighbor graph and a grid cell graph. *Journal of Intelligent Information Systems*, *40*(3), 529-554.

Chen, X. (2014). Synchronization Clustering based on a Linearized Version of Vicsek model. arXiv: 1411.0189 [cs.LG]. http://arxiv.org/abs/1411.0189.

Chen, X. (2015). A new clustering algorithm based on near neighbor influence. *Expert Systems with Applications*, 42(21), 7746-7758.

Chen, X. (2017). An effective synchronization clustering algorithm. *Applied Intelligence*, 46(1): 135 - 157.

Chen, X. (2018). Fast synchronization clustering algorithms based on spatial index structures. *Expert Systems with Applications*, Accepted for publication in 2018.

Comaniciu, D., & Meer, P. (2002). Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *24*(5):603-619.

Dean, J., & Ghemawat, S. (2008). MapReduce: Simplified data processing on large

clusters. *Communication of the ACM*, *51*(1), 107-113.

Ester, M., Kriegel, H. P., Sander, J., & Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial data sets with noise. In *Proceedings of ACM SIGKDD* (pp. 226-231).

Frank, A., & Asuncion, A. (2010). UCI Machine Learning Repository Irvine, University of California.

Frey, B. J., & Dueck, D. (2007). Clustering by passing messages between data points. *Science*, *315*(16), 972-976.

Fukunaga, K., & Hostetler, L. (1975). The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Transactions on Information Theory*, *21*(1):32-40.

Grunwald, P. (2005). A tutorial introduction to the minimum description length principle. Cambridge, MIT Press.

Guha, S., Rastogi, R., & Shim, K. (1998). CURE: An efficient clustering algorithm for clustering large databases. In *Proceedings of ACM SIGMOD* (pp. 73-84).

Güngör, E., & Özmen, A. (2017). Distance and density based clustering algorithm using Gaussian kernel. *Expert Systems with Applications*, 69, 10-20.

Hang, W., Choi, K., & Wang, S. (2017). Synchronization clustering based on central force optimization and its extension for large-scale datasets. *Knowledge-Based Systems*, 118, 31-44.

Horn, D., & Gottlieb, A. (2002). Algorithm for data clustering in pattern recognition problems based on quantum mechanics. *Physical Review Letters*, *88*(1), 018702.

Huang, J. B., Kang, J. M., Qi, J. J., & Sun, H. L. (2013). A hierarchical clustering method based on a dynamic synchronization model. *Science in China Series F: Information Sciences*, *43*(5), 599-610.

Jadbabaie, A., Lin, J., & Morse, A.S. (2003). Coordination of groups of mobile autonomous agents using nearest neighbor rules. *IEEE Transactions on Automatic Control*, *48*(6):998-1001.

Jain, A. K., Murty, M. N., & Flynn, P. J. (1999). Data clustering: A review. *ACM Computing Surveys*, *31*(3), 264-323.

Karypis, G., Han, E. H., & Kumar, V. (1999). CHAMELEON: A hierarchical clustering algorithm using dynamic modeling. *IEEE Computer*, *32*(8), 68-75.

Luxburg, U. V. (2007). A tutorial on spectral clustering. *Statistics and Computing*,

17(4), 395-416.

MacQueen, J. B. (1967). Some methods for classification and analysis of multivariate observations. In *MSP* (pp. 281-297).

Roy, S., & Bhattacharyya, D. K. (2005). An approach to find embedded clusters using density based techniques. *Lecture Notes in Computer Science*, 3816:523-535.

Schölkopf, B., Smola, A., & Müller, K. R. (1998). Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, *10*(5): 1299-1319.

Shao, J., Hahn, K., & Yang, Q., et al. (2010). Hierarchical density-based clustering of white matter tracts in the human brain. *International Journal of Knowledge Discovery in Bioinformatics*, *1*(4):1-25.

Shao, J., Yang, Q., Böhm, C., & Plant, C. (2011). Detection of arbitrarily oriented synchronized clusters in high-dimensional data. *In Proceedings of ICDM* (pp. 607-616).

Shao, J., He, X., Plant, C., Yang, Q., & Böhm, C. (2013a). Robust synchronization-based graph clustering. In *Proceedings of PAKDD* (PP. 249-260).

Shao, J., He, X., Böhm, C., Yang, Q., & Plant, C. (2013b). Synchronization inspired partitioning and hierarchical clustering. *IEEE Transactions on Knowledge and Data Engineering*, *25*(4), 893-905.

Shao, J., Ahmadi, Z., & Kramer, S. (2014). Prototype-based learning on concept-drifting data streams. In *Proceedings of ACM SIGKDD* (pp. 412-421).

Spurek, P., Tabor, J., & Byrski, K. (2017). Active function Cross-Entropy Clustering. *Expert Systems with Applications*, 72, 49-66.

Tan, P.N., Steinbach, M., & Kumar, V. (2005). *Introduction to data mining*. Addison Wesley.

Theodoridis, S., & Koutroumbas, K. (2006). *Pattern recognition* (3rd edition). Academic Press.

Vicsek, T., Czirok, A., & Ben-Jacob, E., et al. (1995). Novel type of phase transitions in a system of self-driven particles. *Physics Review Letter*, *75*(6):1226-1229.

Wang, L., & Liu, Z. (2009). Robust consensus of multi-agent systems with noise. *Science in China Series F: Information Sciences*, *52*(5):824-834.

Wang, W., Yang, J., & Muntz, R. (1997). STING: A statistical information grid approach to spatial data mining. In *Proceedings of VLDB* (pp. 186-195).

White, D. A., & Jain, R. (1996). Similarity indexing with the SS-tree. In *IEEE ICDE* (pp. 516-523).

Zahn, C. T. (1971). Graph-theoretical methods for detecting and describing gestalt clusters. *IEEE Transactions on Computers*, C-20(1), 68-86.

Zhang, T., Ramakrishnan R., & Livny, M. (1996). BIRCH: An efficient data clustering method for very large databases. In *Proceedings of ACM SIGMOD* (pp. 103-114).