A layered parallel algorithm for data clustering via fuzzy c-means technique

Yashuang Mu\textsuperscript{a,b,*}, Wei Wei\textsuperscript{c}, Hongyue Guo\textsuperscript{d,e}, Lijun Sun\textsuperscript{a}

\textsuperscript{a}Key Laboratory of Grain Information Processing and Control (Henan University of Technology), Ministry of Education, Zhengzhou 450001, P.R. China.
\textsuperscript{b}School of Artificial Intelligence and Big Data, Henan University of Technology, Zhengzhou 450001, P.R. China.
\textsuperscript{c}College of Information Science and Engineering, Henan University of Technology, Zhengzhou 450001, P.R. China.
\textsuperscript{d}School of Maritime Economics and Management, Dalian Maritime University, Dalian 116026, China
\textsuperscript{e}Collaborative Innovation Center for Transport Studies, Dalian Maritime University, Dalian 116026, China

Abstract

In this study, a layered parallel algorithm via fuzzy c-means (FCM) technique, called LP-FCM, is proposed in the framework of Map-Reduce for data clustering problems. The LP-FCM mainly contains three layers. The first layer follows a parallel data partitioning method which is developed to randomly divide the original dataset into several subdatasets. The second layer uses a parallel cluster centers searching method based on Map-Reduce, where the classic FCM algorithm is applied to search the cluster centers for each subdataset in Map phases, and all the centers are gathered to the Reduce phase for the confirmation of the final cluster centers through FCM technique again. The third layer implements a parallel data clustering method based on the final cluster centers. After comparing with some famous classic random initialization sequential clustering algorithms which include K-means, K-medoids, FCM and MinMaxK-means on 20 benchmark datasets, the feasibility in terms of clustering accuracy is evaluated. Furthermore, the clustering time and the parallel performance are also tested on some generated large-scale datasets for the parallelism.

Keywords: Clustering, Fuzzy c-means, Map-Reduce, Parallel Computing

1. Introduction

Clustering problems have received a widespread concern as one of the most important data mining tasks in various areas. The underlying mechanism of clustering analysis is to partition a dataset into several sub-clusters according to some predefined distance measures \cite{1}. The data samples which belong to the same sub-cluster share the most similar feature values to each other, and the data samples coming to different sub-clusters are the most dissimilar \cite{2}. Numerous attempts have been made to establish a number of clustering algorithms. Considering from the view point of the statistics, the current clustering methods can be simply divided into two categories \cite{2}. The first category methods are not distribution-free approaches such as the expectation and maximization (EM) algorithm.

*Corresponding author

Email address: muyashuang324@126.com (Yashuang Mu)
[3, 4], and the fuzzy c-directions (FCD) algorithm [5]. The other category methods are
the distribution-free approaches such as K-means algorithm [6] and fuzzy c-means (FCM)
algorithm [7]. In view of the technical details of realizing clustering process [8], the exist-
ing clustering algorithms can be generally classified into five distinct categories including
partitioning-based clustering algorithms [6, 7, 9, 10, 11, 12], hierarchical-based clustering
algorithms [13, 14, 15, 16, 17], density-based clustering algorithms [18, 19, 20], grid-based
clustering algorithms [21, 22] and model-based clustering algorithms [3, 4, 23, 24]. In ad-
dition, there is also another category of fashionable approaches named ensemble clustering
[25, 26, 27].

In partitioning-based clustering algorithms, FCM is a classic and famous clustering
method based on dynamic partition process. Unlike other partitioning-based methods like
K-means [6], FCM is a soft clustering solution which applies fuzzy set theory to describe the
belonging grades of each sample allows the samples belonging to each class with different
grades. As FCM can be easily implemented, and can produce a reasonable result, it is
always a hot topic in data mining fields [28]. However, with quickly increasing data size
involved in many businesses, plenty of traditional clustering models including the famous
FCM algorithm cannot directly deal with large-scale datasets because of some issues like
memory restriction, time complexity and data’s complexity [29].

The FCM is one of the most powerful and well-known clustering algorithm. It is in-
evitably applying FCM to large-scale data clustering problems. To address the issues that
occur in many traditional machine learning methods, parallel or distributed computing
becomes a frequently-used and trustworthy mechanism [30]. The Map-Reduce based dis-
tributed platforms [31] such as Hadoop [32], Spark [33], etc., provide an effective parallel
computing solution for many data mining operations [34, 35]. Some proposed parallel clus-
tering approaches about FCM can be found in the following literatures. In 2007, [36] pre-
sented a parallel implementation for FCM cluster analysis by using a validity index named
PBM [37] to evaluate the quality of the partition. [38] investigated the parallelization of
FCM algorithm in the framework of Map-Reduce (MR-FCM), where two Map-Reduce jobs
are utilized in each iteration. Furthermore, [39] parallelized a fast fuzzy c-means method
(MR-FFCM) and focused on underwater image segmentation. The MR-FFCM developed
a two-layer distribution model to group large-scale images and employed an iterative Map-
Reduce process to optimize the clustering process. In 2017, [40] proposed a FCM model
to classify the sentiments of millions of English documents in Cloudera (a parallel network
environment).

The above mentioned parallel FCM approaches not only exhibit a good effectiveness
on large-scale data clustering, but also make an extensive and far-reaching influence in
real world applications. Nevertheless, there is a similar characteristic for these methods,
that is, applying the parallel computing technique to the local parts of the standard FCM
algorithm or each iterative process. Obviously, these parallel computing solutions need
to be performed on a real distribution computing platform which is usually constructed
based on a computer cluster. Although the computer cluster has become an efficient
method for most engineering and scientific computations, there are some communication
operations among those computers in the cluster. Thus, the communication cost cannot
be avoided in a real distribution computing platform. As the standard FCM algorithm is
an iterative procedure, the previous technique applying parallel computing to local parts
or each iterative process may lead to a lot of communication costs. Motivated by this, in
this study, a simple layered parallel clustering algorithm via FCM technique (LP-FCM)
is developed in the framework of Map-Reduce. Unlike applying parallel computing to
local parts or each iterative process, the proposed LP-FCM algorithm applies Map-Reduce
strategy to the whole of the standard FCM algorithm.

In the proposed LP-FCM algorithm, there are only three parallel computing operations
via Map-Reduce model and each operation can be simply treated as a data processing
layer. First, we develop a parallel data partitioning method at the first layer to randomly
partition an original dataset into several subdatasets. Then, the second layer designs a
cluster centers searching method (the processing procedure is somewhat similar to ensemble
clustering approach), where the classic FCM algorithm is employed to search cluster centers
in each Map phase. Each subdataset can produce a set of centers, and these centers are
gathered to a new dataset in Reduce phase. Meanwhile, the classic FCM algorithm is
applied to the new dataset again to confirm the final cluster centers. Finally, based on the
final cluster centers, a parallel data clustering method is organized as the third layer. In
this stage, the original samples are clustered in a parallel manner on the basis of the Euclid
distance between each of the samples and each of the final cluster centers.

The major contributions of the designed LP-FCM method can be summarized as two
aspects: (1) the proposed LP-FCM can achieve a good competitive clustering result by
comparing with other famous sequential traditional clustering algorithms; (2) the estab-
lished LP-FCM is a simple and effective method in addressing the memory restriction and
time-consuming problems that arise in large-scale data clustering problems.

This study is structured as follows. Section 2 provides some introductions about FCM
algorithm and Map-Reduce. In Section 3, the proposed LP-FCM algorithm is established.
Section 4 shows the experimental results. Finally, the conclusions are made in Section 5.

2. FCM algorithm and the framework of Map-Reduce

In this section, we provide a brief review about the basic knowledge of FCM algorithm
and Map-Reduce architecture.

2.1. FCM algorithm

FCM algorithm is one of the most popular clustering methods, which realizes the final
partitions through optimizing the basic c-means objective function. By using the simple
Picard iteration in the first-order conditions for stationary points, the nonlinear minimiza-
tion problem is resolved in FCM algorithm. Bezdek [7] has proven the convergence of FCM
algorithm. The FCM algorithm is subject to the principle that each data point belongs to
more than one cluster with different membership values ranging from $[0, 1]$. Additionally, the sum of the membership values for each data point must be equal to one $[2, 38]$.

Let $X = \{x_i\}_{i=1}^N$ be a collection of instances in $n$-dimensional vector space, and $c$ $(2 \leq c \leq N)$ denotes the number of clusters. An optimal $c$ partition is realized iteratively by minimizing the weighted sum of squared error objective function:

$$J = \sum_{i=1}^N \sum_{j=1}^c (u_{ij})^w d^2(x_i, c_j),$$

where $u_{ij} = u(x_i, c_j) \in [0, 1]$ is the degree of membership of $x_i$ belonging to the $j^{th}$ cluster, $w$ $(1 < w < \infty)$ is a fuzzification coefficient on each fuzzy membership, $c_j$ is the center of the $j^{th}$ cluster, and $d^2(x_i, c_j)$ is a square distance measure between the instance $x_i$ and the center $c_j$. The detailed iterative process can be obtained in Algorithm 1.

### Algorithm 1: The FCM algorithm.

**Input:** Weighted exponent of fuzzy membership $w$; threshold $\varepsilon$ used as a stopping criterion; dataset $X = \{x_i\}_{i=1}^N$.

**Output:** Updated centers $C$ and fuzzy partition matrix $U$.

1. Randomly initialize the fuzzy partition matrix $U = [u_{ij}(k)]$, where $k$ is the iterative times, $i = 1, 2, ..., N$, $j = 1, 2, ..., c$.

2. **while** true **do**

   3. Calculate the cluster centers with $U^k$:
      $$c_j = \frac{\sum_{i=1}^N u_{ij}(k)x_i}{\sum_{i=1}^N u_{ij}(k)}.$$  

   4. Update the membership matrix $U^{k+1}$ using:
      $$u_{ij}(k + 1) = \frac{1}{\sum_{k=1}^c \left( \frac{d(x_i, c_k)}{d(x_i, c_j)} \right)^{\frac{2}{w-1}}}.$$  

   5. if $\max_{ij} ||u_{ij}(k + 1) - u_{ij}(k)|| < \varepsilon$ **then**

   6. break.

7. **end**

8. **return** $C = \{c_1, c_2, ..., c_c\}$ and $U = [u_{ij}]$.

### 2.2. The framework of Map-Reduce

Map-Reduce is a very effective parallel computing model $[41, 31]$. The simply intent is to apply some parallel or distributed algorithm for some large-scale data mining operations in a cluster. The cluster may contains several computers or thousands of computers, where the Map-Reduce can be realized on different distributed computing platforms to resolve various real world problems $[38, 42, 43]$. There are many Map-Reduce based computing platforms. Hadoop is a famous one among this kinds of platforms, which is an open-source software for parallel computing or distributed computing with the ability of high reliability, scalability and fault tolerance $[32]$.

Hadoop can directly provide a very simple programming framework through applying Map-Reduce model to deal with large-scale datasets in a distributed computing way. As
an open-source software running in a cluster of computers, the primary principle is to complete a whole task in the manner of several parallel single subtasks. There are mainly two processing phases in a Map-Reduce job: Map and Reduce [44, 45]. The Map phase involves a Mapper function, and the Reduce phase involves a Reducer function, where these two functions need to be flexibly implemented by users according to their actual situations. Meanwhile, these two functions take some \(<key, value>\) pairs as the inputs and outputs. A brief illustration about Map-Reduce workflow can be found in Fig. 1.

![Figure 1: The detailed workflow of Map-Reduce.](image)

In general, the Mapper function processes the input data and produces some intermediate results in the Mapper phase. The Mapper function generates a list of intermediate \(<key, value>\) pairs from a stream of single \(<key, value>\) pair. The concrete form can be expressed as follows:

\[
\text{map} \ <key_1, value_1> \rightarrow \text{list} \ <key_2, value_2> . \tag{4}
\]

All the output intermediate \(<key, value>\) pairs are grouped by key before delivering to the Reducer function. Then, these intermediate results are fused as the inputs of the Reducer function during the Reducer phase. The Reducer function exports the final results (the form is also \(<key, value>\) pair) by virtue of aggregation computation. The processing procedure can be represented by the following formula:

\[
\text{reduce} \ <key_2, \text{list} (value_2)> \rightarrow <key_3, value_3> . \tag{5}
\]

3. The layered parallel clustering algorithm

We design a layered parallel algorithm via FCM technique, that is LP-FCM, for data clustering in this section. The proposed LP-FCM algorithm mainly contains three layers: the first layer applies a parallel data partitioning method to randomly divide the original
dataset into several subdatasets; the second layer presents a parallel cluster centers searching method based on the classic FCM algorithm; the third layer uses a parallel clustering data method in terms of the final cluster centers. All these three layers are parallel computing solutions in the framework of Map-Reduce. The main structure of LP-FCM method can be illustrated in Fig. 2. In what follows, we will elaborate the layers one by one.

Figure 2: The main structure of LP-FCM algorithm. (1) Layer 1 contains two Map-Reduce jobs. The first job divides the original data (large hollow cylinder) into several sub-datasets (medium hollow cylinders) and shuffles each sub-dataset. Finally, the randomized data (large light red cylinder) are formed. The second job divides the randomized data into several sub-datasets (medium light red cylinders) and split each sub-dataset into several small data sets (small blue cylinders). Finally, the small data sets with same key are combined into one sub-dataset (medium yellow cylinder); (2) Layer 2 applies FCM to each sub-dataset (medium yellow cylinder) to confirm the centers (green purple/red/green small circle). The centers are merged into a new dataset for applying FCM to determine the final centers. (3) Layer 3 labels each sample with the nearest center through calculating the distance between the sample and each center. Finally, the samples (small purple/red/green cylinder) belonging to the same category are gathered to a cluster (medium purple/red/green cylinder).

3.1. Layer 1: randomly partitioning data

In this layer, we develop a parallel data partitioning method to randomly divide a dataset into several subdatasets through applying Map-Reduce technique. As shown in
Fig. 2, the developed method mainly contains two Map-Reduce jobs. The first Map-Reduce job is a parallel data randomizing method, which is used to randomize the data samples, and the second Map-Reduce job is a parallel data partitioning method aiming to divide the randomized dataset into several subdatasets.

Firstly, in order to randomize a large-scale dataset in a parallel processing way, the first Map-Reduce job used in this layer considers two parameters in advance: the number of used Mappers and the number of random values in each Mapper. The first parameter determines the number of subsets by applying a distributed computing platform to the original dataset, and the other parameter determines the number of keys generated in each Mapper. Suppose the dataset $X$ can be divided into $m$ subsets $X_1, X_2, \ldots, X_m$, the number of random values used in each Mapper is $p$. In the Mapper phases, each sample in subset $X_j$ will be assigned a random integer value as the output key, and the output value is the sample. Then, the samples with the same key are gathered in the Reducer phase. Finally, depending on the assigned key, the samples are stored into some different places. The detailed data randomizing process can be found in the following Algorithm 2.

### Algorithm 2: A parallel randomizing data method.

**Input:** Dataset $X$; the number of random values $p$.

**Output:** A randomized data $\tilde{X}$.

1. **Initialize a Hadoop Job RandomizedDataJob:**
   - Set RandomizedDataMapper as the Mapper class.
   - Set RandomizedDataReducer as the Reducer class.

2. Suppose dataset $X$ can be partitioned into $m$ subsets: $X_1, X_2, \ldots, X_m$.

3. **In the $j$th RandomizedDataMapper:**
   - **Input:** Subset $X_j = \{x_i\}_{i=1}^N$, where $x_i$ is the $i^{th}$ instance.
   - **Output:** $(key, value) = (n_s, x_i)$, where $n_s (n_s \in [1, p])$ is a random integer value.
   - for each $x_i \in X_j$ do
     - Randomly generate an integer value $n_s (n_s \in [1, p])$.
     - **Mapper Output:** $(key, value) = (n_s, x_i)$.
   - end

4. **In the RandomizedDataReducer:**
   - **Input:** $(key, value) = (n_s, list(x_i(j)))$, where $j = 1, 2, \ldots, m$, $list(x_i(j))$ denotes the instances with the same key $n_s$.
   - **Output:** $(key, value) = (null, list(x_i(j)))$.
   - Store the instances $list(x_i(j))$ into $\tilde{X}$.
   - **Reducer Output:** $(key, value) = (null, list(x_i(j)))$.

5. return A randomized data $\tilde{X}$.

Then, the second Map-Reduce job in this layer is aimed to divide the randomized dataset into several subdatasets. There are also two parameters that need to be determined in advance. The first parameter is the number of used Mapper. The second parameter is the number of the partitioned subdatasets. If the number of used Mapper is $m$ and the number of the partitioned subdatasets is $\beta$, then this Map-Reduce job can divide a dataset into $\beta$ subdatasets through $m$ Mappers. The following Algorithm 3 illustrates the partitioning process in detail.
Algorithm 3: A parallel partitioning data method.

Input: Dataset $X$; the number of partitioned subdatasets $N$.
Output: Subsets $X^1, X^2, ..., X^N$.

1. Initialize a Hadoop Job $PartitionedDataJob$:
   Set $PartitionedDataMapper$ as the Mapper class.
   Set $PartitionedReducer$ as the Reducer class.

2. Suppose dataset $X$ can be partitioned into $m$ subsets: $X_1, X_2, ..., X_m$.

3. In the $j^{th}$ $PartitionedDataMapper$:
   Input: Subset $X_j = \{x_i\}_{i=1}^N$, where $x_i$ is the $i^{th}$ instance.
   Output: $(key, value) = (n_s, x_i)$, where $n_s = 1, 2, ..., N$.

4. for each $x_i \in X_j$ do
   Compute the remainder $n_s$ of $\frac{i}{X_j}$.
   Mapper Output: $(key, value) = (n_s + 1, x_i)$.
end

5. In the $PartitionedDataReducer$:
   Input: $(key, value) = (n_s, \text{list}(x_i(j)))$, where $j = 1, 2, ..., m$, $\text{list}(x_i(j))$ denotes the instances with the same key $n_s$.
   Output: $(key, value) = (null, \text{list}(x_i(j)))$.

6. Store the instances $\text{list}(x_i(j))$ into $X^{n_s}$.

7. Reducer Output: $(key, value) = (null, \text{list}(x_i(j)))$.

8. return Subdatasets $X^1, X^2, ..., X^N$.

3.2. Layer 2: searching cluster centers

Suppose dataset $X$ can be randomly partitioned into $m$ subdatasets through the first layer (Layer 1), in Layer 2, a parallel method based on FCM algorithm is established to search the cluster centers of dataset $X$ in the framework of Map-Reduce. As there are $m$ subdatasets, thus the needed number of Mappers is $m$. If the target number of clusters is $c$, then we firstly apply the classic FCM clustering method to each subdataset to confirm its $c$ cluster centers in each Mapper phase. All the cluster centers of subdatasets (the total number of centers is $m*c$) are delivered to the Reducer phase. These centers are gathered to produce a new small dataset $\hat{X}$. Furthermore, the classic FCM algorithm is applied to the new dataset $\hat{X}$ again and search the $c$ cluster centers, where these $c$ cluster centers are approximately regarded as the final cluster centers of dataset $X$.

The following Algorithm 4 shows the searching process of the final cluster centers in detail. It is easy to find that the data cluster centers of dataset $X$ are not directly generated by the classic FCM algorithm, but are produced through applying FCM to several subdatasets. The classic FCM algorithm, which is treated as a technique, is fused into the different processing phases of Map-Reduce model. This solution provides a simple way to endow the classic FCM with the ability to deal with large-scale data clustering problems.

3.3. Layer 3: clustering data

When the cluster centers $c_1, c_2, ..., c_c$ are confirmed, the dataset still needs to be partitioned into $c$ clusters in a parallel computing way following the mechanism of Map-Reduce. The number of Mappers is equal to the number of subsets of dataset $X$. In each Mapper
Algorithm 4: A parallel searching centers method.

Input: Dataset \( X \); the number of clusters \( c \).

Output: The final cluster centers \( c_1, c_2, \ldots, c_c \).

1. Initialize a Hadoop Job SearchCentersJob:
   1. Set SearchCentersMapper as the Mapper class.
   2. Set SearchCentersReducer as the Reducer class.

2. Suppose dataset \( X \) can be partitioned into \( m \) subsets: \( X_1, X_2, \ldots, X_m \).

3. In the \( j \)th SearchCentersMapper:
   1. Input: Subset \( X_j = \{x_i\}_{i=1}^N \), where \( x_i \) is the \( i \)th instance.
   2. Output: \((key, value) = (null, [c_1(X_j), c_2(X_j), \ldots, c_c(X_j)])\).

4. Apply the classic FCM method (Algorithm 1) on dataset \( X_j \) to confirm cluster centers \( c_1(X_j), c_2(X_j), \ldots, c_c(X_j) \).

5. Mapper Output: \((key, value) = (null, [c_1(X_j), c_2(X_j), \ldots, c_c(X_j)])\).

6. In the SearchCentersReducer:
   1. Input: \((key, value) = (null, list([c_1(X_j), c_2(X_j), \ldots, c_c(X_j)])), \) where \( j = 1, 2, \ldots, m \).
   2. Output: \((key, value) = (null, [c_1, c_2, \ldots, c_c])\).

7. Gather all centers and form a new dataset \( \hat{X} \).

8. Apply the classic FCM method (Algorithm 1) on dataset \( \hat{X} \) to confirm cluster centers \( c_1, c_2, \ldots, c_c \).

9. Reducer Output: \((key, value) = (null, [c_1, c_2, \ldots, c_c])\).

10. return \( c_1, c_2, \ldots, c_c \).

Algorithm 5: A parallel clustering data method.

Input: Dataset \( X \); the final cluster centers \( c_1, c_2, \ldots, c_c \).

Output: The final clusters \( X_1, X_2, \ldots, X_c \).

1. Initialize a Hadoop Job ClusterDataJob:
   1. Set ClusterDataMapper as the Mapper class.
   2. Set ClusterDataReducer as the Reducer class.

2. Suppose dataset \( X \) can be partitioned into \( m \) subsets: \( X_1, X_2, \ldots, X_m \).

3. In the \( j \)th ClusterDataMapper:
   1. Input: Subset \( X_j = \{x_i\}_{i=1}^N \), where \( x_i \) is the \( i \)th instance.
   2. Output: \((key, value) = (null, [X_1, X_2, \ldots, X_c])\).

3. for each \( x_i \in X_j \) do
   1. Compute Euclidean distance \( d(x_i, c_k) \), where \( k = 1, 2, \ldots, c \).
   2. Select \( k^* = \arg\min_k d(x_i, c_k) \) and store \( x_i \) to \( X_{k^*} \).
   3. Mapper Output: \((key, value) = (k^*, x_i)\).

4. end

5. return \( X_1, X_2, \ldots, X_c \).
3.4. The proposed LP-FCM clustering algorithm

On basis of the previous work, the layered parallel clustering algorithm via FCM technique (LP-FCM) is established in this section. The detailed pseudocode is given in Algorithm 6.

**Algorithm 6:** The proposed LP-FCM algorithm.

**Input:** Dataset $X$; the number of centers $c$.

**Output:** The final clusters $X_1, X_2, ..., X_c$.

1. Randomize the dataset $X$ by Algorithm 2 (Layer 1).
2. Partition $X$ into $m$ subdatasets $\{X_j\}_{j=1}^m$ by Algorithm 3 (Layer 1).
3. Search the final cluster centers $c_1, c_2, ..., c_c$ by Algorithm 4 (Layer 2).
4. Divide dataset $X$ into $c$ clusters $X_1, X_2, ..., X_c$ by Algorithm 5 (Layer 3).
5. return $X_1, X_2, ..., X_c$.

As shown in Algorithm 6, there are three layers or four steps in the pseudocode. First, the dataset $X$ is randomized and partitioned by Algorithm 2 and Algorithm 3, respectively. Then, we employ Algorithm 4 to search the final cluster centers. Finally, the dataset $X$ is divided into $c$ clusters $X_1, X_2, ..., X_c$ by Algorithm 5. All these steps are some parallel implementations in the framework of Map-Reduce. The proposed LP-FCM approach incorporates the classical FCM method to the procedure of Map-Reduce in a simple and specific way, which can effectively resolve some issues in dealing with large-scale data clustering for the traditional FCM algorithm.

3.5. Analysis of the time and space complexity

Suppose there are $n$ samples and each sample has $f$ attributes, the number of iterations is $k$, the number of clusters is $c$, it can be analyzed that the time and space complexity of the traditional FCM algorithm are $O(k \times c \times n \times f)$ and $O(n \times f)$, respectively. For the proposed LP-FCM algorithm, the time and space complexity is mainly related to the two phases of Map-Reduce. If the original dataset can be partitioned into $m$ subsets in Map phase, then the time complexity is $O(\frac{k \times c \times n \times f}{m})$ and the space complexity is $O(\frac{n \times f}{m})$. In the Reduce phase, as the number of gathered centers is $c \times m$, the time complexity is $O(k \times c \times c \times m \times f)$ and the space complexity is $O(k \times m \times f)$. Thus, the total time and space complexity of the proposed LP-FCM are $O(\frac{k \times c \times n \times f}{m}) + O(k \times c \times c \times m \times f)$ and $O(\frac{n \times f}{m}) + O(k \times m \times f)$, respectively. However, in practice, due to the reasons like a cluster system with heterogeneous hardware, slightly different data partition sizes, network latencies, etc, there are usually some communication costs in a cluster of computers, which is a challenging problem in parallel computing or distributed computing system.

4. Experimental studies

To assess the performance of the proposed LP-FCM approach, several numerical experimental studies are described. First, Section 4.1 covers some descriptions about the computing environment and the datasets used in the experiments. Then, in order to verify
the feasibility of the proposed LP-FCM algorithm, a comparative analysis on clustering accuracy with some famous random initialization clustering methods is offered in Section 4.2. Furthermore, Section 4.3 and Section 4.5 include some analysis about the parallelism in terms of the execution time and some evaluation indices on some generated datasets.

4.1. Computing environment and data sets

In this study, we implement the proposed LP-FCM algorithm with Java language based on the open source Hadoop distribution computing platform [32] (the version of software is 2.6.0), meanwhile, some data structures in WEKA [46] (the version of software is 3.6.9) are also utilized. All sequential methods are implemented on base of the statistics and machine learning toolbox in the platform of MATLAB (the version of software is R2015b). All the experiments related to the sequential modes are carried out on a machine with Intel Core i5-4590 3.30 GHZ, 8 GB RAM and Ubuntu 14.04.1 LTS (64 Bit) OS, and all experiments related to parallel modes are conducted on a small distribution computing cluster, where it mainly contains one host machine with Intel Core i5-4590 3.30 GHZ, 8 GB RAM, Ubuntu 14.04.1 LTS (64 Bit) OS and six servant machines with Intel Core i3 2.93 GHZ, 4 GB RAM, Ubuntu 14.04.1 LTS (64 Bit) OS.

The experiments totally employ 21 UCI benchmark datasets [47] to evaluate the sequential clustering algorithms and the proposed parallel clustering algorithm in this study. Specially, the samples with missing values are deleted from the original data sets. The detailed compositions related to these datasets are exhibited in Table 1. The first 20 datasets are mainly used to compare with some sequential clustering algorithms for the feasibility analysis, and the last dataset is aimed to test the parallelism.

| No. | Data sets    | Attributes | Class | Instances |
|-----|--------------|------------|-------|-----------|
| 1   | Banknote     | 4          | 2     | 1,372     |
| 2   | Blood        | 4          | 2     | 748       |
| 3   | Breast-cancer-w-o | 9      | 2     | 699       |
| 4   | Breast-cancer-w-d | 30     | 2     | 569       |
| 5   | Ecoli        | 7          | 8     | 366       |
| 6   | Fertility    | 9          | 2     | 100       |
| 7   | Glass        | 9          | 7     | 214       |
| 8   | Hayes-roth   | 4          | 3     | 160       |
| 9   | Heart-statlog| 13         | 2     | 270       |
| 10  | Iris         | 4          | 3     | 150       |
| 11  | Lenses       | 4          | 3     | 24        |
| 12  | Magic        | 10         | 2     | 19,020    |
| 13  | Messidor     | 19         | 2     | 1,151     |
| 14  | Skin         | 3          | 2     | 245,057   |
| 15  | Seeds        | 7          | 3     | 210       |
| 16  | Segment      | 19         | 2     | 2,310     |
| 17  | Sonar        | 60         | 2     | 208       |
| 18  | Waveform     | 40         | 3     | 5,000     |
| 19  | Wine         | 13         | 3     | 178       |
| 20  | Wilt         | 5          | 2     | 4,839     |
| 21  | Covertype    | 54         | 7     | 581,012   |
4.2. Comparisons with K-means, K-medoids, FCM and MinMaxK-means

In order to test the feasibility of the proposed LP-FCM algorithm, we compare it with several famous random initialization clustering methods: K-means [6], K-medoids [9], FCM [7] and MinMaxK-means [48]. The classic K-means, K-medoids and FCM are implemented by using the related functions in MATLAB toolboxes, where the parameter ‘Start’ is set as ‘sample’ (if exits) and others parameters are set as their defaults. The implementation of MinMaxK-means is also developed by Matlab codes, in which the parameters are used as their defaults.

During the experiments, all these clustering methods are executed 50 times for each benchmark dataset, and the real number of class labels is taken as the initial number of clusters. The clustering accuracies [28, 49] on all benchmark datasets are recorded in Table 2, where the values are the means values of 50 independent runs. Meanwhile, the number of processors used in the proposed LP-FCM approach is set as 3. The results marked with bold face denote that the current method is the best one among all methods on the corresponding dataset.

Table 2: The comparison of clustering accuracies on 20 data sets.

| Data sets    | K-means | K-medoids | FCM     | MinMaxK-means | LP-FCM |
|--------------|---------|-----------|---------|---------------|--------|
| Banknote     | 0.6122  | 0.5951    | 0.6093  | 0.6028        | 0.5994 |
| Blood        | 0.7247  | 0.7259    | 0.7072  | 0.7262        | 0.7005 |
| Breast-cancer-w-o | 0.9610 | 0.9580    | 0.9561  | 0.9213        | 0.9458 |
| Breast-cancer-w-d | 0.8541 | 0.8541    | 0.8541  | 0.8313        | 0.8660 |
| Ecoli        | 0.5397  | 0.5932    | 0.5107  | 0.5483        | 0.6393 |
| Fertility    | 0.6064  | 0.5312    | 0.5514  | 0.5876        | 0.6160 |
| Glass        | 0.4995  | 0.5349    | 0.4907  | 0.5015        | 0.5083 |
| Hayes-roth   | 0.4648  | 0.4294    | 0.3453  | 0.4606        | 0.4613 |
| Heart-statlog| 0.5899  | 0.6074    | 0.5926  | 0.5926        | 0.5984 |
| Iris         | 0.8340  | 0.8939    | 0.8933  | 0.8867        | 0.9335 |
| Lenses       | 0.4825  | 0.4975    | 0.4167  | 0.4725        | 0.4992 |
| Magic        | 0.6491  | 0.6168    | 0.5780  | 0.6392        | 0.5969 |
| Messidor     | 0.5065  | 0.5155    | 0.5325  | 0.5039        | 0.5585 |
| Skin         | 0.5513  | 0.5514    | 0.5511  | 0.5399        | 0.5382 |
| Seeds        | 0.8930  | 0.8952    | 0.8952  | 0.8952        | 0.8734 |
| Segment      | 0.6956  | 0.6935    | 0.6420  | 0.6645        | 0.5695 |
| Sonar        | 0.5450  | 0.5203    | 0.5529  | 0.5481        | 0.5704 |
| Waveform     | 0.5142  | 0.5768    | 0.5248  | 0.5162        | 0.5747 |
| Wine         | 0.6637  | 0.6820    | 0.6854  | 0.6685        | 0.6942 |
| Wilt         | 0.5200  | 0.5211    | 0.5015  | 0.5226        | 0.5012 |
| Avg.         | 0.6354  | 0.6397    | 0.6197  | 0.6315        | 0.6426 |
| (5/20)       | (5/20)  | (1/22)    | (3/20)  | (8/20)        |

From the values recorded in Table 2, we can intuitively find that, among these algorithms, the classic K-means, K-medoids, FCM, and MinMaxK-means methods achieve better performance on 5, 5, 1 and 3 datasets among 20 datasets, while the proposed LP-FCM algorithm can achieve 8 datasets among 20 datasets. According to the average of clustering accuracies on all these benchmark datasets, the proposed LP-FCM approach also obtains a similar result with other clustering algorithms.

Furthermore, in order to check whether there is a significant difference among these
clustering methods, we provide a statistical discussion on these computational results by
employing Friedman test [50]. In the Friedman test, the null-hypothesis is that the tested
indices are equivalent. At the significance level \( \alpha = 0.05 \), the critical value with 4 and
76 degrees of freedom is 2.4920, but the \( F_F \) derived from the Table 2 is 1.2748, which
is smaller than 2.4920. Thus, there is not a significant difference among these clustering
methods.

From the above discussions, the results deduce that the proposed LP-FCM method
can keep the same level of performance on clustering accuracy with some classic clustering
methods, which validates the feasibility of the proposed LP-FCM algorithm.

4.3. The execution time analysis

As the proposed LP-FCM algorithm is a parallel solution for clustering problems, it is
necessary to study the execution time. In this experiment, we examine the execution time
on Covertype dataset. The dataset contains 54 conditional attributes and one decision
attribute with 7 categories. The original Covertype dataset includes 581,012 instances,
however, to be able to clearly present the parallelism, we generate 6 datasets from the
original dataset by bootstrap technique [51] to test the execution time for data clustering.

In Table 3, the generated 6 datasets are described in detail.

| No. | Data sets        | Attributes | Class | Instances  |
|-----|------------------|------------|-------|------------|
| 1   | Covertype (25MB) | 54         | 7     | 202,621    |
| 2   | Covertype (50MB) | 54         | 7     | 405,242    |
| 3   | Covertype (75MB) | 54         | 7     | 607,863    |
| 4   | Covertype (100MB)| 54         | 7     | 810,484    |
| 5   | Covertype (125MB)| 54         | 7     | 1,013,105  |
| 6   | Covertype (150MB)| 54         | 7     | 1,215,726  |

The execution time of Algorithm 6 at six different processors is summarized in Table 4,
where the results with mark “—” denote that there is a memory warning in the experiment.
According to the values, it is obvious that, in most cases, the execution time of LP-
FCM algorithm presents a downtrend with the increasing number of the processors, which
reflects the efficiency of parallel solution in reducing execution time. Especially, for the
case of Covertype (25MB), the value with six processes is larger than the value with five
processes, which is unnatural comparing with other cases. The reason is that there is a
communication time existing among these processors. When the amount of data is smaller
and the number of processors is bigger, the communication time takes a large proportion
in the total execution time. In additional, when the amount of data is increased, the less
processors cannot deal with the dataset because of some memory restrictions, but this
issue can be resolved through utilizing more processors. Taking the case of Covertype
(100MB), when two processors are employed to deal with the dataset, a memory warning
is appeared. However, we can observe that the memory warning is disappeared when the
number of processors is raised to three or more.
### Table 4: The execution time (seconds) of LP-FCM.

| Data sets       | One       | Two       | Three      | Four       | Five       | Six        |
|-----------------|-----------|-----------|------------|------------|------------|------------|
| Covertype (25MB)| 947.278   | 530.862   | 392.053    | 343.638    | 323.866    | 346.952    |
| Covertype (50MB)| 1746.253  | 1020.711  | 788.154    | 672.986    | 568.792    |            |
| Covertype (75MB)| —         | 1503.130  | 1302.679   | 921.933    | 805.442    |            |
| Covertype (100MB)| —        | 2311.940  | 1735.194   | 1431.983   | 1193.799   |            |
| Covertype (125MB)| —       | —         | 2454.673   | 2037.295   | 1485.984   |            |
| Covertype (150MB)| —       | —         | —          | 2559.292   | 1776.333   |            |

#### 4.4. The comparison on reducing time between LP-FCM and MR-FCM

In order to further present the effectiveness of the proposed LP-FCM algorithm on reducing execution time, we make a comparison on execution time between LP-FCM and the parallel clustering algorithm MR-FCM [38]. The main similarity of the two methods is that both of them are the parallelization techniques of the classic FCM algorithm. The main difference is that the proposed LP-FCM algorithm applies parallel computing to the whole of the classic FCM approach, while the parallel MR-FCM algorithm applies parallel computing to local parts or each iterative process. The used datasets in the experiments are selected from Table 3.

Firstly, we fix the number of processors and employ the datasets Covertype (50MB), Covertype (50MB), Covertype (75MB), Covertype (100MB) in Table 3 to test the execution time. During the experiments, both of the two algorithms employ the same parameters, and are executed on the same cluster where the number of processors are set as 3. Through conducting the experiments under the same conditions, the values are obtained and displayed in Fig. 3. We can easily find that the proposed LP-FCM algorithm can evidently save the execution time comparing with the parallel MR-FCM algorithm.

![Figure 3: The execution time on different datasets.](image)

Then, we fix the data size (where the used dataset is Covertype (50MB)) and adjust the number of processors from 2 to 5. From the comparison of results as shown in Fig. 4, two conclusions can be easily summarized. (1) The execution time presents a downward trend with the increase in the number of processors. (2) The proposed LP-FCM parallel solution
has more advantages on execution time than the parallel MR-FCM clustering algorithm.

![Figure 4: The execution time on different processors.](image)

4.5. The parallel performance analysis

In addition, we also compute some evaluation indices such as Speedup, Scaleup and Sizeup [52] to further verify the parallel performance of the proposed LP-FCM algorithm. Speedup represents how much a parallel system with \( m \) processors is faster in time than a system with one processor. Scaleup means the ability of a parallel system with \( m \) processors to process \( m \) times data during the same original computing time. Sizeup measures how much longer time is consumed while dealing with \( m \) times original data than dealing with the original data in a given parallel system.

Here, we treat the dataset Covertype (25MB) as the original dataset in Table 3, and the datasets Covertype (50MB), Covertype (75MB), Covertype (100MB), Covertype (125MB) and Covertype (150MB) are respectively regarded as the corresponding times datasets of Covertype (25MB). The execution time has been recorded in Table 4. The definitions of Speedup and Sizeup require that the missing values must be known. Thus, in order to calculate smoothly, we only use dataset Covertype (25MB) to depict the performance of Speedup. From the tendency of the curve, we can obviously find that the Speedup is highly correlated to the number of processors. With the increase in number of processors, the Speedup is gradually increasing. However, the Speedup does not exhibit an increasing tendency. When the number of processor is five, the Speedup reaches the maximum. That reason is that there are more communication costs when the number of processors increases, and the communication costs take a bigger proportion in the total execution time.
Then, for the Scaleup performance, the following Fig. 6 depicts the tendency. According to the definition, the used datasets are 1 to 6 times of original data Covertype (25MB), and the number of processors is equal to the times for each case. In an ideal parallel system, Scaleup should be equal to 1. However, as the reason of communication cost among the processors, the Scaleup exhibits a downward tendency. Meanwhile, when the data and the processors increase, the Scaleup reduces slowly, or shows a fluctuation trend.

Furthermore, we also study the Sizeup performance of the proposed LP-FCM algorithm. During the experiment, the processors are fixed as 5 and 6, respectively. Meanwhile, the used datasets are 1 to 6 times of the original data Covertype (25MB), respectively. The detailed tendencies are summarized in Fig. 7. From the curves, the proposed approach shows a good Sizeup performance in the given parallel system.

5. Conclusions

In this study, a parallel algorithm named LP-FCM is proposed for data clustering problems. The LP-FCM clustering approach mainly contains three layers: randomly partitioning data, searching cluster centers and clustering data. All the layers are implemented in the framework of Map-Reduce. Especially, in the second layer, the famous FCM technique is fused to the different phases of Map-Reduce model. During the experimental studies, the
feasibility is evaluated by comparing with some traditional clustering algorithms, and the parallelism is examined through analyzing the execution time and some evaluation indices for parallel performance.

At last, it deserves to point out three aspects of this paper: (1) there are three layers in the proposed approach, where each layer can be treated as a processing step for other large-scale data mining algorithms. For example, the first layer can be regarded as a parallel data pre-processing step, and the second layer can be applied to confirm the parameters of fuzzy sets in a large-scale fuzzy system; (2) there are also other kinds of clustering algorithms such as density-based methods. The idea of effectively parallelize other cluster algorithms is also a meaningful theme.

Acknowledgments

This work is supported by the Research Foundation for Advanced Talents (2019BS007, 31401204) of Henan University of Technology. This work is supported by the Research Foundation for Advanced Talents of Henan University of Technology (No. 2019BS007), the Open Fund of Key Laboratory of Grain Information Processing and Control (Henan University of Technology), Ministry of Education (No. KFJJ-2020-112), and the National Natural Science Foundation of China under Grants (No. 62006071).

Ethical approval

The authors declare that there is no conflict of interests regarding the publication of this paper.

Funding

This study is not supported by any source or any organizations.

Conflict of interest

The authors declare that there is no conflict of interests regarding the publication of this paper.
Informed Consent

Informed Consent was not required as no human or animals were involved.

Authors’ Contributions

Yashuang Mu: writing-original draft and model visualization; Wei Wei: methodology and data analyses; Hongyue Guo: writing-review and data analyses; Lijun Sun: writing-review and supervision.

References

[1] E. Zhu and R. Ma, “An effective partitional clustering algorithm based on new clustering validity index,” *Applied Soft Computing*, vol. 71, no. C, pp. 608–621, 2018.

[2] O. Kesemen, Ö. Tezel, and E. Özkul, “Fuzzy c-means clustering algorithm for directional data (FCM4dd),” *Expert Systems with Applications*, vol. 58, pp. 76–82, 2016.

[3] A. P. Dempster, N. M. Laird, and D. B. Rubin, “Maximum likelihood from incomplete data via the EM algorithm,” *Journal of the Royal Statistical Society*, vol. 39, no. 1, pp. 1–38, 1977.

[4] G. J. McLachlan and K. E. Basford, “Mixture models: Inference and applications to clustering,” *Inference & Applications to Clustering*, vol. 38, no. 2, 1988.

[5] M. S. Yang and J. A. Pan, “On fuzzy clustering of directional data,” *Fuzzy Sets and Systems*, vol. 91, no. 3, pp. 319–326, 1997.

[6] J. Macqueen, “Some methods for classification and analysis of multivariate observations,” in *Proc. of Berkeley Symposium on Mathematical Statistics and Probability*, 1965, pp. 281–297.

[7] J. C. Bezdek, R. Ehrlich, and W. Full, “FCM: The fuzzy c-means clustering algorithm,” *Computers & Geosciences*, vol. 10, no. 2, pp. 191–203, 1984.

[8] A. Fahad, N. Alshatri, Z. Tari, and A. Alamri, “A survey of clustering algorithms for big data: Taxonomy and empirical analysis,” *Emerging Topics in Computing IEEE Transactions on*, vol. 2, no. 3, pp. 267–279, 2014.

[9] H. S. Park and C. H. Jun, “A simple and fast algorithm for K-medoids clustering,” *Expert Systems with Applications*, vol. 36, no. 2, pp. 3336–3341, 2009.

[10] R. T. Ng and J. Han, “Efficient and effective clustering methods for spatial data mining,” in *VLDB Conference*, 1994, pp. 144–155.

[11] L. Kaufman and P. J. Roussieeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*. DBLP, 1990.
Ng, T. Raymond, Han, and Jiawei, “CLARANS: A method for clustering objects for spatial data mining,” *IEEE Transactions on Knowledge & Data Engineering*, vol. 14, no. 5, pp. 1003–1016, 2002.

T. Zhang, R. Ramakrishnan, and M. Livny, “BIRCH: An efficient data clustering method for very large databases,” in *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, vol. 25, no. 2, 1994, pp. 103–114.

S. Guha, R. Rastogi, and K. Shim, “CURE: An efficient clustering algorithm for large databases,” *Information Systems*, vol. 26, no. 1, pp. 35–58, 1998.

S. Guha, R. Rastogi, and K. Shim, “ROCK: A robust clustering algorithm for categorical attributes,” *Information Systems*, vol. 25, no. 5, pp. 345–366, 1999.

G. Karypis, E. H. Han, and V. Kumar, *Chameleon: Hierarchical Clustering Using Dynamic Modeling*. IEEE Computer Society Press, 1999.

A. N. Mahmood, C. Leckie, and P. Udaya, “An efficient clustering scheme to exploit hierarchical data in network traffic analysis,” *IEEE Transactions on Knowledge & Data Engineering*, vol. 20, no. 6, pp. 752–767, 2008.

M. Ester, H. P. Kriegel, and X. Xu, “A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise,” in *International Conference on Knowledge Discovery and Data Mining*, 1996, pp. 226–231.

M. Ankerst, M. M. Breunig, H. P. Kriegel, and J. Sander, “OPTICS: Ordering points to identify the clustering structure,” in *International Conference on Management of Data and Symposium on Principles of Database Systems*, 1999, pp. 49–60.

X. Xu, M. Ester, and H. P. Kriegel, “A distribution-based clustering algorithm for mining in large spatial databases,” in *International Conference on Data Engineering, 1998, Proceedings*, 1998, pp. 324–331.

G. Sheikholeslami, S. Chatterjee, and A. Zhang, “Wavecluster: A multi-resolution clustering approach for very large spatial databases,” in *International Conference on Very Large Data Bases*, 1998, pp. 428–439.

W. Wang, J. Yang, and R. R. Muntz, “STING: A statistical information grid approach to spatial data mining,” pp. 186–195, 1997.

Fisher and H. Douglas, “Knowledge acquisition via incremental conceptual clustering,” *Machine Learning*, vol. 2, no. 2, pp. 139–172, 1987.

J. H. Gennari, P. Langley, and D. Fisher, “Models of incremental concept formation,” *Artificial Intelligence*, vol. 40, no. 1, pp. 11–61, 1989.

A. Strehl, J. Ghosh, “Cluster ensembles: A knowledge reuse framework for combining partitionings,” *Journal of Machine Learning Research*, vol. 3, no. 3, pp. 583–617, 2002.
[26] S. Vega-Pons, J. Ruiz-Shulcloper, “A survey of clustering ensemble algorithms,” *Journal of Pattern Recognition and Artificial Intelligence*, vol. 25, no. 03, pp. 337–372, 2011.

[27] M. Mojarad, S. Nejatian, H. Parvin, and M. Majid, “A fuzzy clustering ensemble based on cluster clustering and iterative Fusion of base clusters,” *Applied Intelligence*, vol. 47, no. 7, pp. 2567–2581, 2019.

[28] H. X. Pei, Z. R. Zheng, C. Wang, C. N. Li, and Y. H. Shao, “D-FCM: Density based fuzzy c-means clustering algorithm with application in medical image segmentation,” *Procedia Computer Science*, vol. 122, pp. 407–414, 2017.

[29] Y. Mu, X. Liu, and L. Wang, “A pearson’s correlation coefficient based decision tree and its parallel implementation,” *Information Sciences*, vol. 435, pp. 40–58, 2018.

[30] Y. Mu, L. Wang, and X. Liu, “A fast rank mutual information based decision tree and its implementation via map-reduce,” *Concurrency and Computation: Practice and Experience*, vol. 30, no. 10, 2018.

[31] J. Dean and S. Ghemawat, “MapReduce: Simplified data processing on large clusters,” *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[32] Apache Hadoop. Available online: http://hadoop.apache.org/.

[33] Apache Spark. Available online: https://spark.apache.org/.

[34] J. Dean and S. Ghemawat, “MapReduce: A flexible data processing tool,” *Communications of the ACM*, vol. 53, no. 1, pp. 72–77, 2010.

[35] L. Zhang, K. Li, Y. Xu, J. Mei, F. Zhang, and K. Li, “Maximizing reliability with energy conservation for parallel task scheduling in a heterogeneous cluster,” *Information Sciences*, vol. 319, no. C, pp. 113–131, 2015.

[36] M. V. Modenesi, M. C. A. Costa, A. G. Evsukoff, and N. F. F. Ebecken, “Parallel fuzzy c-means cluster analysis,” in *International Conference on High Performance Computing for Computational Science*, pp. 52–65.

[37] M. K. Pakhira, S. Bandyopadhyay, and U. Maulik, “Validity index for crisp and fuzzy clusters,” *Pattern Recognition*, vol. 37, no. 3, pp. 487–501, 2004.

[38] S. A. Ludwig, “MapReduce-based fuzzy c-means clustering algorithm: implementation and scalability,” *Journal of Machine Learning and Cybernetics*, vol. 6, no. 6, pp. 923–934, 2015.

[39] L. Xiu, J. Song, Z. Fan, X. Ouyang, and S. U. Khan, “MapReduce-based fast fuzzy c-means algorithm for large-scale underwater image segmentation,” *Future Generation Computer Systems*, vol. 65, no. C, pp. 90–101, 2016.
[40] V. N. Phu, N. D. Dat, V. T. N. Tran, V. T. N. Chau, and T. A. Nguyen, “Fuzzy C-means for english sentiment classification in a distributed system,” Applied Intelligence, vol. 46, no. 3, pp. 717–738, 2017.

[41] S. Shankland, “Google spotlights data center inner workings,” CNet News Blog, posted on May, vol. 30, p. 2008, 2008.

[42] J. Chen, K. Li, Z. Tang, K. Bilal, S. Yu, C. Weng, and K. Li, “A parallel random forest algorithm for big data in a spark cloud computing environment,” IEEE Transactions on Parallel & Distributed Systems, vol. 28, no. 4, pp. 919–33, 2017.

[43] G. Tang, W. Yang, K. Li, Y. Ye, G. Xiao, and K. Li, “An iteration-based hybrid parallel algorithm for tridiagonal systems of equations on multi-core architectures,” Concurrency and Computation: Practice and Experience, vol. 27, no. 17, pp. 5076–5095, 2015.

[44] T. White, Hadoop: The Definitive Guide, 4th Edition. O’Reilly Media, 2015.

[45] I. Triguero, D. Peralta, J. Bacardit, S. Garca, and F. Herrera, “MRPR: A MapReduce solution for prototype reduction in big data classification,” Neurocomputing, vol. 150, no. 150, pp. 331–345, 2015.

[46] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, “The WEKA data mining software: An update,” ACM SIGKDD Explorations Newsletter, vol. 11, no. 1, pp. 10–18, 2009.

[47] D. Dheeru and E. K. Taniskidou, 2017, UCI Machine Learning Repository. Available online: http://archive.ics.uci.edu/ml.

[48] G. Tzortzis, A. Likas, and G. Tzortzis, “The MinMax K-means clustering algorithm,” Pattern Recognition, vol. 47, no. 7, pp. 2505–2516, 2014.

[49] M. Halkidi, Y. Batistakis and M. Vazirgiannis, “On clustering validation techniques,” Journal of Intelligent Information Systems Integrating Artificial Intelligence & Database Technologies, vol. 17, no. 2, pp. 107–145, 2001.

[50] J. Demšar, “Statistical comparisons of classifiers over multiple data sets,” Journal of Machine Learning Research, vol. 7, no. 1, pp. 1–30, 2006.

[51] R. W. Johnson, “An introduction to the bootstrap,” Teaching Statistics, vol. 23, no. 2, pp. 49–54, 2001.

[52] Q. He, T. Shang, F. Zhuang, and Z. Shi, “Parallel extreme learning machine for regression based on mapreduce,” Neurocomputing, vol. 102, pp. 52–58, 2013.