MapReduce-based Fast Fuzzy C-means Algorithm for Large-scale Underwater Image Segmentation

Xiu Li¹*, Jingdong Song¹, Fan Zhang², Xiaogang Ouyang¹, Samee U. Khan³
¹Division of Information Science & Technology, Graduate School at Shenzhen, Tsinghua University, Shenzhen, China, 518055
²Kavli Institute for Astrophysics and Space Research, Massachusetts Institute of Technology, Cambridge, MA 02139
³Department of Electrical and Computer Engineering, North Dakota State University, Fargo, ND 58108-6050
*Corresponding Author: li.xiu@sz.tsinghua.edu.cn

Abstract — The research on underwater image segmentation is faced with the problem of rapidly increasing number of images and videos. To solve this issue, parallel computing paradigms, such as the MapReduce framework seems a viable solution. Consequently, in this paper, we proposed a MapReduce-based fast fuzzy c-means algorithm (MRFFCM) to provide the capability of parallelly segmenting the increasing number of images. In our work, we use a two-layer distribution model to group the large-scale images and adopt an iterative MapReduce process to parallelize the FFCM algorithm. A combination of segmentation way is also used to improve algorithm’s efficiency. To evaluate the performance of our algorithm, we develop a small Hadoop cluster to execute the MRFFCM algorithm. The experiment results demonstrate that our proposed method is effective and efficient on large-scale images. When compared to the traditional non-parallel methods, our algorithm can be expected to provide a more efficient segmentation on images with at least 13% improvement. Meanwhile, with the growth of cluster size, further improvement of the algorithm performance was also achieved. Consequently, such a scalability can enable our proposed method to be used effectively in oceanic research, such as in underwater data processing systems.

Index Terms — fast fuzzy c-means algorithm; image segmentation; MapReduce; two-layer distribution model

I. INTRODUCTION

With the development of information technology frameworks and solutions, and with the increasing demand from the scientific and engineering research community, the ability to process large-scale data is becoming an emerging need in almost every discipline, and oceanic research is no exception. In oceanic research, massive data can be generated in multiple ways, such as from the large amount of in situ sensors, from process modeling and simulation, and from institutional historical data archives. Such data contains not only observations that represent changes in underwater environment, but also the oceanographic scientists’ ideas, efforts and achievements. Therefore, effective and efficient processing of such data is in urgent demand, which could accelerate invaluable scientific findings. For example, to the oceanographic scientists, who may lack time or professional training to deal with massive data, large-scale data processing, especially steps such as data cleaning and data pre-processing become a tedious task. Therefore, the assist from other scientists of fields such as information, computer and oceanographic sciences, seems to be required.

In recent years, research on large-scale data processing has become a hot topic. Parallel computing paradigms, such as the MapReduce framework, has been considered as a plausible way to process ‘big’ data. Therefore, it has attracted sufficient attention and research, both in industry and academia. Since initially processed in 2004 [1], MapReduce has been used widely in many areas, such as log file analysis [2], machine translation [3], and computation in meteorology [4]. In oceanic research, MapReduce has also been applied in areas such as sensor data compression [5]. As an inherent attribute, the MapReduce is particularly useful in numerical data processing. However, when it comes to image processing, the MapReduce framework has not received sufficient attention, especially in emerging area of underwater image processing.

Similar to the numerical data, underwater images and videos are captured by underwater instruments, such as cameras. Because the processing of videos can be reduced to an operation of processing video frames, the research on underwater videos can be translated to research on images. Therefore, besides numerical data processing, the underwater image processing is also a significant part of large-scale oceanic data processing research. For the oceanographic scientists, the real interest is in the information pertaining to the benthos and anomalous of the underwater events. For instance, many scientists have focused on underwater moving object detection [6], and a key step of such work is segmenting the objects of interest of a very large data set. Therefore, it is necessary to processing such images efficiently, and possibly by exploring the usage of the MapReduce framework.

In this paper, we focus on underwater image segmentation with MapReduce framework. Because image segmentation is the fundamental step in image processing, we will use a fast fuzzy c-means algorithm (FFCM) to achieve that. In our work, we
propose a parallel FFCM algorithm based on MapReduce, and used the algorithm to segment interesting benthos objects, such as fish and jellyfish from large-scale underwater images. A two-layer distribution model was used to group large-scale images and an iterative MapReduce process was used to optimize the process. To evaluate our proposed algorithm's performance, experiments have been conducted on an small Hadoop [7] cluster. The results show that our proposed algorithm is effective and efficient on large-scale image dataset. Also when compared to traditional non-parallel segmentation methods or other researchers' work, our proposed algorithm will be more efficient on large-scale image dataset, and be more scalable in the algorithm's performance.

The rest of paper is organized as follows. Section II will introduce the related work of fuzzy c-means algorithm, include its application on image segmentation. Moreover, research pertaining to the application of MapReduce in oceanographic science will also be briefly reviewed. Thereafter Section III will introduce the large-scale data set problem in underwater image segmentation research. In Section IV, our proposed algorithm will be detailed. The two-layer distribution model, MapReduce-based FFCM algorithm, and the analysis of the algorithm on a Hadoop cluster will be elaborated in this section, as well. Section V details the performance evaluation and experimental results of algorithm. Lastly, Section VI concludes this research.

II. RELATED WORK

The fuzzy c-means algorithm (FCM) is an unsupervised classification technique. Unlike the hard c-means methods who usually classify a data point into just one cluster exclusively, the FCM allows a data point belonging to multiple categories with its degree of membership [8]. Therefore, by introducing the fuzzy mathematics theory, the FCM is considered to be an improved and more general algorithm than hard c-means [9]. Therefore, FCM has been widely applied to feature analysis, clustering, and the classifier design [10]. Moreover, the disciplines in which FCM has been utilized are also multiple: astronomy, bioinformatics [11], electrical power system [12], network structures [13], and image segmentation.

Although image segmentation algorithms are diverse, such as the edge-based segmentation, the watershed method, and the adaptive thresholding method are all adopted in this area, the FCM is still an useful and better performed algorithm in some cases [14]. Compared to other methods like hard c-means, the FCM can provide a more flexible classification result and retain more information from original images [15]. Such a quality makes the FCM a more attractive method in image segmentation. Especially in the Magnetic Resonance Images (MRI) research, many improved FCM methods have been used by the researchers. They either modified the objective function [16] or distance metric [15], or took the spatial information into account, to overcome the affection from noises, and improve the performance and robustness of segmentation procedure. Except that, researchers in other disciplines have also achieved some results, such as remote sensing image segmentation [17].

Among the various improved FCM algorithms, a method that using the image gray histogram to replace the pixel gray value in the parametric update process, has been used widely. Compared to the standard FCM, the improved FCM algorithm will classify the pixels based on gray histogram. Although the method requires that different regions within an image must have relative difference in the gray value, the improved FCM can still transfer the operation on pixels to the operation on gray histogram vector. Therefore, the improved FCM may reduce large amount of data storage, which is especially important when the image size is large in an iterative process, and accelerate the convergent speed [18]. Therefore, the improved FCM is usually named the fast fuzzy c-mean method (FFCM), which we also will use in our work.

The FFCM has been utilized in many fields, include the underwater image segmentation. Though other methods such as thresholding [19] and neural network [20] are sometimes used in image segmentation, Padmavathi et al. [14] found that FCM outperformed other segmentation methods in underwater image processing, which makes it attract growing attention in recent years. Also Padmavathi et al. [21] studied the FCM in underwater image segmentation and estimated the quality of segmented results. Shah et al. [22] used the FCM too, and also concluded that the FCM can retain more information but cost more process time. Even both works above were simple; they still demonstrated the superiority of the FCM. At the same time, Singh et al. [23] proposed a method combining contrast limited adaptive histogram equalization image enhancement with thresholding with 3-class fuzzy c-means clustering. Their work improved the contrast of underwater images and showed effectiveness in image segmentation. Wang et al. [18] provided a particle swarm optimization-improved histogram weighted fuzzy c-means algorithm in underwater object segmentation, for purpose of designing a more powerful autonomous underwater vehicle. However, all works described above are pertain to the segmentation performance on a single image; but we find no major work that details the segmentation speedup when encountering large-scale image datasets. The accuracy of segmentation is definitely important, but increasingly severity of large-scale data problem must not be ignored. Therefore, more attention must be put on the research of processing large scale underwater images. However, as far as we know, the relevant works are rather few.

To solve such problem, we considered parallel computing paradigms such as MapReduce framework. As its superiority in processing large-scale data, MapReduce is growingly popular in both industry and academia. Oceanographic research is also one such area that needs computational capability to process large-scale information, such as sensor data compressing and Web-based sensor data processing [24]. However, when it comes to image processing and particularly segmentation, to the best of our knowledge, there are only a handful of reported work.
In this paper, we will combine the MapReduce framework and the FFCM algorithm, to tackle the large-scale underwater image dataset segmentation problem. The work reported here, will not only advance the oceanic science, but also increase our understanding of combining the MapReduce framework with the modified FFCM algorithm.

III. THE PROBLEM IN UNDERWATER IMAGE SEGMENTATION

Underwater image segmentation is an important step in the whole image processing cycle. It is usually a pre-process step to image processes, such as moving object detection, object tracking and automated object recognition. As depicted in Figure 1, the underwater image segmentation process will take the original images or videos as input, and will output the segmented images that contain objects that the scientists may be interested in, such as new species of benthos, interesting animal behaviors or anomalous events occurring underwater.

![Image of the underwater image processing cycle](image)

Figure 1. The underwater image processing cycle

As mentioned earlier, traditional research on underwater image segmentation focused more on a single image. The research either sought to improve the accuracy and robustness of the segmentation, or tried to overcome the affection from the underwater environment, such as illumination conditions and other noises. Although some improved algorithms (e.g., the FFCM) were proposed to accelerate the convergent speed, the methodology still focused on a single image. However, nowadays, as more and more cameras have been installed underwater, the number of captured underwater images have substantially increased. As a result, the image segmentation faces the issue of the large-scale dataset problem. In a nutshell, it requires the segmentation process to be efficient, to fulfill the requirements of AUV designs or underwater data processing systems. In some cases, the requirements proposed by the scientists might be even more stringent: for example, real-time or near real-time capabilities may be proposed. Consider an underwater camera that has captured a video containing the fierce environment changes, which may indicate a climate disaster. Therefore, the timely processing of the video stream is extremely important, and in this case, it seems that the fast processing must have precedence over accurate processing.

On the other hand, because the underwater cameras have improved their performance, clearer images or videos can be expected. Moreover, the affection from image noises caused by devices may also be reduced. In this case, we must provide more attention and effort to improve the processing speeds. To realize this goal, we must make a tradeoff between efficiency and accuracy. In this paper, we focus on improving the speed of image segmentation. We use a cluster and MapReduce to parallelize the segmentation process. A two-layer distribution model and a FFCM algorithm are used. By providing a more efficient segmentation, image processes based on segmented results can also be executed efficiently.

IV. MAPREDUCE-BASED FAST FUZZY C-MEANS ALGORITHM

In this section, our proposed algorithm will be introduced in detail. It contains a two-layer distribution model and a parallel FFCM algorithm based on MapReduce. Moreover, in order to make the proposed algorithm more efficient, modifications on objective function and center matrix have been taken into account. Subsequently, performance analysis of our proposed algorithm on Hadoop cluster will be done, to guide us selecting suitable cluster size in image processing.

A. Fast Fuzzy C-means Algorithm

The FFCM is an improved version of standard FCM. As a clustering method, the FCM can segment the underwater image into several parts through clustering the pixels, based on their colors or gray values. Since in underwater images, there is difference in the light reflection or absorption, in most cases, the target objects, such as fishes will have color or gray scale difference with the corresponding backgrounds. Therefore, we can segment image into two parts: foreground containing target objects and background. The image segmentation process can be regarded as a two-class clustering problem.

In the standard FCM , the image segmentation process will cluster the pixels \( X = \{x_1, x_2, \ldots, x_n\} \) into two clusters, using an iterative optimal procedure. And as defined, the FCM uses the membership to measure similarity between pixels and cluster
centres. For instance, \( u_{ji} \in [0, 1] \) represents the membership of pixel \( x_i \) to cluster \( v_j \). Therefore, both final clustering results and the objective function of iterative optimal process can be expressed using the membership \( U = \{u_{ji}\} \in \mathbb{R}^{m 	imes n} : 
abla \)

\[
F_n(U, V) = \sum_{j=1}^{m} \sum_{i=1}^{n} \left( u_{ji} \right)^m d^2_{ji} \left( x_i, v_j \right)
\]

(1)

In equation (1), \( m \) means the fuzzy weighted index and \( d^2_{ji} \left( x_i, v_j \right) \) means distance between pixel \( x_i \) and centre \( v_j \). As related research indicates, \( m \in [1.5, 2.5] \) will be considerable choice [25]. And in the iterative optimal process of algorithm, membership and centres matrix will be updated in each round. Through using the Lagrangian multiplier method, the update formulas of memberships and cluster centres can be derived as:

\[
u_j = \left( \frac{\sum_{i=1}^{n} \left( u_{ji} \right)^m x_i}{\left( \sum_{i=1}^{n} \left( u_{ji} \right)^m \right)} \right) \left( \frac{\sum_{j=1}^{m} v_j}{\sum_{j=1}^{m} \left( u_{ji} \right)^m} \right)^{2/(m-1)}
\]

(3)

Up to now, the iterative optimal procedure can be executed based on above equations, until the objective function is convergent. But as mentioned earlier, each time we update the cluster centres, all of pixels must be used. Obviously it is a disadvantage when the method is used in image segmentation, since it brings large amount of data storage and computation burden to processors. More importantly, the convergent speed is slower. Hence, to solve this problem, the FFCM algorithm was used in this research, instead of FCM algorithm.

The FFCM algorithm is based on image gray histogram. For an image whose size is \( W \times H \), the gray value of pixel in position \( (x, y) \) is defined as \( f(x, y) \), which ranges from 0 to 255 in 8-bit gray level. Therefore, the gray histogram of the gray image can be defined as:

\[
h(i) = \frac{1}{H} \sum_{y=0}^{H-1} \sum_{x=0}^{W-1} \delta_{y}(i), i = 0, 1, ..., 255
\]

(4)

The \( \delta_{y}(i) \) in equation (4) is defined as following:

\[
\delta_{y}(i) = \begin{cases} 1, & f(x, y) = i \\ 0, & \text{else} \end{cases}
\]

(5)

In the FFCM algorithm, the image segmentation process is based on such histogram. Through thresholding or clustering, the histogram can be partitioned into two parts, refer to image foreground and background. After that, pixels can be classified according to their position in the histogram. In this way, FFCM can replace the operation on pixels with the operation on gray histogram. Therefore, the parameters can be modified as:

\[
F_n'(U, V) = \sum_{i=0}^{255} \sum_{j=1}^{m} \left( u_{ji} \right)^m \cdot h(i) \cdot d^2_{ji} \left( i, v_j \right)
\]

(6)

\[
u_{ji} = 1/ \sum_{i=1}^{255} \left( d^2_{ji} \left( i, v_j \right) \right)^{2/(m-1)}
\]

(7)

\[
v_j = \left( \frac{\sum_{i=0}^{255} \left( u_{ji} \right)^m h(i) \cdot i}{\sum_{i=0}^{255} \left( u_{ji} \right)^m h(i)} \right)
\]

(8)

The \( d^2_{ji} \left( i, v_j \right) \) means distance between each gray level and each cluster centre. Similar to the standard FCM, the iterative optimal procedure can be executed based on equation (7) and (8). Steps of the FFCM algorithm can be expressed as follows:

Step1: Initialize the cluster centres \( V = \{v_1, v_2\} \), which are random values at first;

Step2: Calculate or update the distance \( d^2_{ji} \left( i, v_j \right) \);

Step3: According to the equation (7) and (8), calculate or update the memberships and cluster centres;

Step4: Update the objective function according to equation (6), and judge that whether it is convergent or not;

If objective function is convergent, the algorithm is end; otherwise the Step2 to 4 need to be repeated multiple times.

In this algorithm, the pixels can be classified into the foreground cluster or the background cluster, according to their gray values and memberships. Then some initial and rough segmented results can be obtained, based on which, the other image processes or analysis can be executed.

B. The Two-Layer Distribution Model
In our proposed algorithm, the two-layer distribution model is a major part. In this model, the large-scale underwater images are grouped according to their gray distribution or similarity. And then they are distributed across the Hadoop cluster. In this way, segmentation process can be executed parallelly in Hadoop cluster; and also we can leverage the cluster’s capability and MapReduce framework to improve the efficiency of our proposed image segmentation algorithm.

We all know that a MapReduce program usually contains two parts: the map function and reduce function. The former takes split data chunks as input and generate intermediate K-V pairs; while the later collects and combines such pairs and outputs the final results. Therefore, in order to segment images in MapReduce way, the large-scale images must be firstly split into ‘data chunks’, images need to be grouped and distributed. The framework of distribution model is illustrated in Figure 2.

![Figure 2. The framework of two-layer distribution model](image)

In the first layer, the original underwater images are firstly gathered to a dataset, which is also the source dataset of our algorithm. After that, the large-scale images are grouped and added into the compressed files named HIB files. The HIB file, also named HIPI [26] Image Bundle file, is a special compressed image file type proposed in HIPI, a Hadoop-oriented open source image processing interface. By using HIPI, images can be stored on HDFS more easily, and be used by MapReduce-based algorithms more directly.

At the same time, in order to group underwater images properly, the Perceptual Hash Algorithm [27] is used. In this algorithm, the underwater image is transformed into a $8 \times 8$ gray image firstly; then an average gray value of such image will be calculated. To obtain the gray value distribution, or the ‘fingerprint’ of the image, gray value of each pixel is compared with the average gray value: if larger, it is set as 1; otherwise it is set as 0. Then to measure the similarity of two images, the Hamming distance of their ‘fingerprints’ is calculated. By using Hamming distance and the ‘fingerprints’ of images, we can group images into several clusters. Therefore, in first layer, the large-scale underwater images can be reduced into several clusters, and then compressed into HIB files. The HIB files are distributed into different nodes of Hadoop cluster then. As a result, the ‘image groups’ will be processed in parallel in each virtual machine or virtual cluster, which is relative independent cluster that almost exchange no data with others in the MapReduce procedure, then the subsequent second layer can use the MapReduce-based FFDM algorithm to segment these underwater images, based on the ‘image groups’.

The second layer is the core of our work. The distributed ‘image groups’ will be segmented in this layer. To make the segmentation process more suitable processing the grouped images, in our work, we did not blindly consider the ‘distribute’ operation, but also used the ‘combination’ operation:

Since in a HIB file, the images are similar, which means their gray distributions or gray histograms are similar as well. Hence, to further improve the efficiency of our proposed algorithm, the images with similar gray histograms would be segmented all at once, using the same parameters, such as cluster centres and objective function. As to the cluster centres, since they refer to image foreground and background, they would be similar in a HIB file, as well; therefore, their update formula in equation (8) would be replaced by an mean value of some sampled images:

$$v'_j = \bar{v}'_{sample} = \frac{1}{t} \sum_{i=1}^{t} v'_i, j = 1, 2$$

(9)

The $v'_j$ refers to each image’s cluster centre and $t$ means the number of some sampled images, which is much less that the total images number in a HIB file. Similarly, the objective function can also be modified. In this way, since calculation for all images can be reduced to calculation for several sample images, a large amount of data storage and computation can be reduced,
which makes our proposed algorithm more efficient. The experiments in Section V will demonstrate effectiveness of the ‘combination’ segmentation way. The detailed process of the second layer and the design of the ‘combination’ operation are illustrated in Figure 3.

![Diagram of the second layer process and combination design](image)

**Figure 3.** The detailed process of the second layer and the design of the ‘combination’ segmentation way (dashed boxes show)

Finally, by combining the first layer’s group and distribute algorithm with the second layer’s MapReduce-based FFCM algorithm (with the ‘combination’ segmentation way), we can obtain a efficient, parallel underwater image segmentation algorithm to process the large-scale underwater image dataset.

**C. The MapReduce-based FFCM Algorithm**

The most important part of the distribution model is the MapReduce-based FFCM algorithm. In this algorithm, the HIB files would be taken as input; and the segmented images would be output as results. The flow chart of the MapReduce-based FFCM algorithm is illustrated in Figure 4.

![Flow chart of the MapReduce-based FFCM algorithm](image)

**Figure 4.** The flow chart of the MapReduce-based FFCM algorithm
From Figure 4, we can see that the centre file is also an input of the algorithm. It contains the centre cluster values that indicate clustering result of each iterative round. The algorithm starts with setting initial parameters, include cluster number, fuzzy weighted index, convergent threshold, etc. And images processed in the method are gray images as mentioned earlier. Then the MapReduce-based algorithm’s two functions: map and reduce, will process these images.

Map function: in this function, we took the default K-V pairs ruled in HIPI as the input K-V pairs: the ImageHeader was used as key while the FloatImage was used as value.

ImageHeader is a data structure that contains image basic information, such as width, height, bands and exif information. To some extent, it can be used to identify an image uniquely.

FloatImage is a one-dimension float array that contains the gray values of pixels. In the algorithm, it will become the only dataset that represents image gray value information.

As shown in Figure 4, in map function, the gray histogram \( h \) of image is calculated firstly. Then the distances \( d_{\delta}(i, v_j) \) and membership matrix \( U \) are updated. Since in this paper, the \( d_{\delta}(i, v_j) \) refers to distance between gray level and cluster centre, and both of them are one-dimension vector, so distance \( d_{\delta}(i, v_j) \) in this situation is a one-norm Euclidean distance as the following, where \( i \) is the gray level.

\[
d_{\delta}(i, v_j) = |i - v_j|, \quad i = 0, 1, ..., 255, j = 1, 2, ... \tag{10}
\]

With the distance expressed above, the initial random memberships can be updated according to the equation (7). As the cluster number is 2, the size of membership matrix in the FFCM is \( 2 \times 256 \), which is an obvious less size than standard FCM algorithm, the \( 2 \times n \), \( n \) is the image pixels number.

The memberships and distances are updated in mappers parallelly, because they are various in each image. But as mentioned, the ‘combination’ segmentation way requires the parameters such as cluster centres and objective function to be processed together. Therefore, memberships and distances must be put into the intermediate K-V pairs. We redesign the K-V pairs, in which cluster indexes, such as number ‘1’ means foreground and ‘2’ means background, are set as key, and a joint parameters string containing gray level, gray histogram, memberships, distances, etc., are set as value:

\[
\langle \text{key}: 1 \quad \text{value}: "i_k, h(i_k), u_{ik}, d_{ik}, F^{u_{ik}}\rangle \langle \text{key}: 2 \quad \text{value}: "i_k, h(i_k), u_{ik}, d_{ik}, F^{u_{ik}}\rangle
\]

Here ‘1’ or ‘2’ means cluster index; and in value, the parameters included are gray level, histogram, memberships, distances and old objective function in sequence. They are transformed from double data type into string data type firstly, and then spliced together with a special symbol, such as ‘,’. In this way, they can be easily transferred from map to reduce, and be correctly parsed in the reduce function.

Another important part of map function is the segmented images creation function. The function won’t be triggered until the objective function is convergent as follows:

\[
F_n^{u}(U, V) = F_n^{u}(U, V) < \text{Threshold}_{obj} \tag{11}
\]

The \( F_n^{u}(U, V) \) and \( F_{n+1}^{u}(U, V) \) are objective functions of last and current round respectively; \( \text{Threshold}_{obj} \) is set as an initial parameter. When equation (11) is satisfied, last round’s result will be regarded as the final clustering result, so we can classify the pixels into foreground or background according to their memberships. To create the segmented images, the image creation function provided by HIPI will be used. In which, the classified pixels will be stored in two FloatImages; and then be transformed to JPEG files and be written to file system. Name of each JPEG file is the hash code that generated by the function as well, which is used to avoid duplicate file name. Of course, if the convergent condition is not satisfied, the image creation function won’t be triggered, the algorithm will continue its next stage, the reduce function.

Reduce function: in this function, the cluster centres and the objective function will be updated. Since in our proposed algorithm, the cluster number is 2, so the number of reducers, where intermediate K-V pairs are sent to, is also 2. One refers to the foreground and the other refers to the background. The intermediate K-V pairs generated in map function will be directed to one of reducers according to their key, the cluster index. In common methods, each image’s K-V pairs need to be processed one by one, but in the ‘combination’ segmentation way, only several sample images’ K-V pairs will participate in calculation. Results of which are used to replace the results of all images. Therefore, in reduce function, the update formula of cluster centres is shown as the equation (9), not equation (8). Here the selection of sample images is random at first, we are still looking for a more reasonable way. Similarly, the ‘part’ of objective function can be updated, as well. The ‘part’ means that each item in equation (12) is calculated in each reducer; and then they are finally added up to form the total objective function, after the processing of reduce function:

\[
F_n^{u}(U, V) = \sum_{0}^{255} (u_{i}^{u}) \cdot h(i) \cdot d_{\delta}^{u}(i, v_j) + \sum_{0}^{255} (u_{i}^{u}) \cdot h(i) \cdot d_{\delta}^{u}(i, v_j) \tag{12}
\]
The equation (12) is just a deformation of equation (6). After being updated, the centres and objective function parts need to be output to the centre file. Therefore, the output K-V pairs of the reduce function will be set as the following:

\[
\begin{align*}
\text{key: } \text{null} & \quad \text{value: } "v_i^t F_{m, \text{part } i}^t" \\
\text{key: } \text{null} & \quad \text{value: } "v_i^t F_{m, \text{part } i}^t"
\end{align*}
\]

The \( F_{m, \text{part } i}^t \) means the objective function part, the \( \text{value} \) in the pairs contains both the cluster centres and \( F_{m, \text{part } i}^t \); while the \( \text{key} \) is set as \( \text{null} \), which is a common choice. The K-V pairs have the same form as the content in initial centre file, and indicates the clustering result of this iterative round. Therefore, we can use them to monitor the execution process of our proposed algorithm.

Until here, an iterative round is completed. In algorithm’s main function, the round will be invoked multiple times, until the convergent condition is satisfied. At last, as described in map function, two segmented images are output. Based on the images, other image processes can be conducted.

D. Algorithm’s Performance Prediction on Hadoop Cluster

Since the large-scale data problem caused by the growing underwater images has emerged in oceanic research, the tasks that processing images need to be more efficient. However, the traditional non-parallel segmentation methods seemingly can’t achieve this goal, since their near-linear way in large-scale image processing. As to our proposed algorithm, the large-scale data problem can be expected to be solved, because we leverage both parallel computing paradigms and capability of computer cluster. In this section, performance of traditional methods and our proposed algorithm is analyzed respectively, to demonstrate that our algorithm has theoretical superiority on large-scale image dataset than traditional methods.

Traditional non-parallel methods: In these methods, images will be processed in sequence, which makes the total processing time a near-linear function of images number. We supposed that one image processing time is \( t_i \) and the images number is \( m \); then total processing time is:

\[
t_{\text{traditional}} = m \cdot t_i
\]  

(13)

The \( t_i \) we thought can be divided into three parts: an iterative optimal procedure’s time \( t_i \), a final classification time \( t_z \), and a histogram calculation time \( t_s \). For time \( t_i \), since the iterative optimal procedure is based on equation (6) to (8), so \( t_i \) is a function related to the gray level \( I \), the cluster number \( c \), and the iteration time \( I \); \( t_i = I \cdot O(c \cdot I) \). For time \( t_z \), every pixel needs to be used in the final classification procedure, so processing time \( t_z \) is a function related to cluster number \( c \) and pixels number \( n \); \( t_z = O(c \cdot n) \). The similar analysis can be done on time \( t_s \) as well. Therefore, with the ignoring of image reading time, one image processing time \( t_i \) can be estimated:

\[
t_i = I \cdot O(c \cdot I) + O(c \cdot n) + O(n)
\]  

(14)

Combined with the equation (13), the total processing time of \( m \) images is expressed as the following:

\[
t_{\text{traditional}} = m \left( I \cdot O(c \cdot I) + O(c \cdot n) + O(n) \right)
\]  

(15)

From which, we can see that if the \( m \) is a big number, the image segmentation process will cost a large amount of time. And with the increasing of \( m \), the increasing pattern of the processing time will be rather rapid. But when it comes to our proposed algorithm, the situation is different.

MapReduce-based FFCM algorithm: This algorithm will contain multiple iterative rounds. In each round, we thought that the processing time can be divided into four parts: \( t_1 \), time spent on reading HIB and centre file; \( t_2 \), map function time; \( t_3 \), reduce function time; and \( t_4 \), time spent on covering the old centre file. In such for times, \( t_2 \) and \( t_4 \) will be both fixed values as we observed in experiments. Total number of such two times is about 1 second as we recorded, and it has no relationship with images number. Other two times in the algorithm that have no relationship with images number are startup time and clean-up time of MapReduce process. For convenience of following analysis, in our work, we use a fixed number \( t_{\text{start}} \) to represent the total value of the four constant times.

As to time \( t_2 \), since the map function in one iterative round can be divided into four parts, i.e., the histogram calculation, the distances & membership calculation, the K-V pairs writing, and segmented images creation; it consists of four times, as well, time \( t_{21} \) to \( t_{24} \). Subsequently, in one image situation, for time \( t_{21} \), as same analysis done on \( t_i \), it can be: \( t_{21} = O(n) \); for time \( t_{22} \) and \( t_{23} \), both of they have the similar form as time \( t_i \); \( t_{22} + t_{23} = O(c \cdot I) + O(c \cdot I) = 2O(c \cdot I) \); and finally for time \( t_{24} \), it will only be added into \( t_2 \) when algorithm is convergent, so the time \( t_2 \) has two forms in the iterative rounds. Also we suppose that number of iteration times is \( I \), then the time \( t_2 \) in multiple iterative round would be like:
\[ t'_i = I \cdot \left[ O(n)+2 \cdot O(c \cdot l) \right] + O(c \cdot n) \] (16)

As to time \( t'_i \), since the existence of the ‘combination’ segmentation way, it has the form as the following:

\[ t'_i = I \cdot O(t_{\text{map}}, l) \] (17)

Therefore, in one image situation, total processing time of our proposed MapReduce-based algorithm will be like:

\[ t'_i = I \cdot \left[ O(n)+O(c \cdot l) \right] + O(c \cdot n) + t'_{\text{final}} \] (18)

Here we have considered the meaning of symbol \( O \). When compared to the equation (14), our algorithm spent a little more time than tradition non-parallel methods, which may be caused by the repeat calculation of histogram and the fixed time \( t'_{\text{final}} \).

The analysis above is in one image situation, when it comes to large-scale images dataset, the situation is different. For example, when \( m \) exceeds one mapper’s process capability, the cluster would set up more mappers to parallelly process them, makes performance of our algorithm improved. Suppose one mapper can process \( k \) images to the maximum; and cluster is powerful enough to provide sufficient mappers to process data. And then estimation of time \( t'_2 \) and \( t'_1 \) must be modified based on these assumptions:

\[ t'_2 = I \cdot \left[ O(k \cdot n)+2 \cdot O(k' \cdot c \cdot l) \right] + O(k \cdot c \cdot n) \] (19)

\[ t'_1 = I \cdot O(k' \cdot l) \] (20)

The \( k' \) has the same meaning as \( k \), indicating maximum possible number of K-V pairs that one mapper can process. Therefore, when \( m \) is big enough, the total processing time of our proposed algorithm will be changed into:

\[ t'_{\text{dir}} = I \cdot \left[ O(k \cdot n)+O(k' \cdot c \cdot l) \right] + O(k \cdot c \cdot n) + t'_{\text{final}} \] (21)

Here condition \( I \cdot O(k' \cdot c \cdot l) < I \cdot m \cdot O(c \cdot l) \) is always true, since in most cases the condition \( k' < m \) is satisfied. So when compared to the equation (15), \( t'_{\text{dir}} \) in equation (21) is less than \( t'_{\text{traditional}} \) in traditional methods. More importantly, the bigger the \( m \) is, the more performance improvement that our proposed algorithm can provide. And when compared to the equation (18), we can see that the growth pattern of processing time is no longer the near-linear way. When cluster is powerful enough, a rather slow or flat processing time increasing can be gotten, makes it more suitable for massive images processing.

However, as mentioned earlier, analysis above is based on assumption that a ideal situation. We didn’t take the problem of lacking enough mappers into account. In fact, such problem often happens in real word, and the estimation of processing time will be different, which we will analysis in next section.

V. Performance Evaluation

In this section, we will evaluate our method’s performance. The traditional methods are realized using Matlab [28] (version 7.11.0 R2010b), while our proposed algorithm is realized using Java language. The HIPI is used to provide image processing interface. In this paper, the experimental environment is a small Hadoop cluster consisting of five nodes: the master node is equipped with a quad-core 3.3GHz Intel Xeon CPU; 8GB main memory and 500GB disk storage; while four slaves are both equipped with quad-core 3.1GHz Intel i5 CPU; 4GB main memory and 250GB disk storage. Operation systems are all 64-bit Centos 6.4; and all nodes are in a 100Mbps LAN.

As to the data source, we use underwater images from a project named fishknowledge [29]. Images of which contain various kinds of fishes; and for each kind, several continuous, similar images or video frames are provided.

A. The Effectiveness Experiments’ Results

The first experiment compares the segmented results of our proposed algorithm with traditional methods such as standard FCM and FFCM. Figure 5 shows the results. In which, (a) to (c) refer to different kinds of underwater environments. In each situation, from left to right are original image, the FCM’s result, the FFCM’s result and our algorithm’s result in turn.
In Figure 5, the segmented results of our proposed method show similar accuracy with two traditional methods, even some simplification techniques such as ‘combination’ way has been used. It means that our algorithm can retain the acceptable accuracy while being efficient in processing large-scale images. Some deficiency can be seen in Figure 5 as well, especially in (c). In which, underwater environment has been affected by illumination disturbance, making segmented results are not as good as we want. But if environment is ideal or even affected by turbid seawater, our algorithm can output accurate results, as long as fished have different gray distributions with the background, as shown in Figure 5(a) and (b).

The second experiment compares our proposed algorithm with works of other researchers. Intuitive segmented results comparisons are shown in Figure 6 to Figure 8. Then discrete entropy of such results would be used as an assessment to measure such results quantitatively. Table I shows the values.

**Table I. The Discrete Entropy of the Segmented Results**

<table>
<thead>
<tr>
<th>Names of researchers</th>
<th>Results of other researchers work</th>
<th>Results of our proposed algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Padmavathi et al.</td>
<td>0.1032</td>
<td>0.1025</td>
</tr>
<tr>
<td>Wang et al.</td>
<td>1.9888</td>
<td>1.5359</td>
</tr>
<tr>
<td>Singh et al.</td>
<td>3.6116</td>
<td>4.0739</td>
</tr>
</tbody>
</table>

![Images](image1.png)

**Figure 6. The results comparison with work of Padmavathi et al. [21]**

From Figure 6 and Table I, we can see the segmented result of our proposed algorithm can retain more image details, and has sizeable or even larger discrete entropy, means that more information has been retained.
From Figure 7 and Table I, we can find that the comparable accuracy can be obtained as well; but the discrete entropy is smaller than Wang et al., indicating that more noise's affection has been reduced in our proposed algorithm.

And when compared with the work of Singh et al. [23], our proposed algorithm generates a more reasonable result as well: more image details, such as the edge of coral reef and the fish, can be correctly segmented by our method, as Figure 8 shows. The larger discrete entropy also proves that. In general, the comparison of segmented results above shows that accuracy of our algorithm can be at least as good as such works and in some cases, our proposed algorithm even performed better.

Whereafter, the third experiment tests the effectiveness of our 'combination' way. In this experiment, we firstly group the images from a same period of video into a same HIB file, which usually contains a same kind of fish, but different postures in different images. The segmented results are shown in Figure 9.

In Figure 9, we can see that original images that come from a same period of video have similar gray distributions, which leads to the similar cluster centers. Therefore, in our proposed 'combination' segmentation way, since we use such similar centers, correct results can be obtained, as Figure 9(b) shows.

Moreover, even if such images are from different videos, 'combination' way can be applied as well, as long as images have similar gray distributions. So we then picked some images coming from different periods of videos, but are quite similar with each other, the segmented results are shown in Figure 10.
The first three original images come from one video, while another two are from a different one. The segmented results indicate that even images contain different kinds of fishes, the correct results can also be obtained, since they have similar gray distributions with each other.

However, when it comes to other situations, such as the images are not similar at all, the 'combination' way will be no longer applicable. Figure 11 illustrate this situation.

![Figure 11](image)

In the situation of Figure 11 shows, though we can get the final cluster centers, they can’t represent all foregrounds and backgrounds of all images correctly. Therefore, the segmented results shown in (b) are not so good as we want. But if we separately process such two kinds of images, the much better results we can obtain, as (c) shows. It means that our proposed ‘combination’ way is highly depend on the gray distributions of images, which we have considered in the first layer of model. Therefore, our design of grouping images according to their gray distributions is proven to be reasonable and essential.

### B. The Efficiency Experiments' Results

In this research, the main goal and advantage is to provide a more efficient segmentation process than the traditional methods, on the large-scale image dataset. Therefore, in this paper, to demonstrate the efficiency of our proposed algorithm, some experiments would be conducted on a small Hadoop cluster, which contains five nodes. Here, we firstly used three of them, since they have been built before, the five-node one would be used later for comparison.

The first experiment is based on some underwater images that larger than images from fish4knowledge. Based on them, we artificially generated some big HIB files. We want the HIB files to be large, at least larger than 64MB, the default file split size of Hadoop. In the experiment, sizes of HIB files range from 1.1MB to 7.1GB, and numbers of images range from 5 to 32,000. Then we use Matlab that installed on master node to process images in traditional ways, while use Java and HPI to realize our proposed algorithm. Running times of four algorithms are recorded: the standard FCM, the FFCM, MapReduce-based FCM and our proposed algorithm. The former two are executed on single computing node, and the later two are running on the three-node cluster. Running times results of the experiment are reported in Figure 12 as the following.
Figure 12. Running times of four segmentation algorithms

In Figure 12, both the standard FCM and the MapReduce-based FCM consumed too much time on algorithm’s execution when file size is too large. So in the experiment, we would not record their running times if file size exceeds a large threshold, 2GB for example, because we think it makes no sense. Also we can find a huge running times difference exists among the four algorithm: the FFCM-like algorithms are more efficient that the other two. Even though we made some simplifications in the MapReduce-based FCM, it still cost much more time. So we can say that in image segmentation, using histogram is a good choice to improve algorithm efficiency. As Table II shows as well, data volume of MapReduce-based FFCM has been reduced much than the MapReduce-based FCM. Map Output Records here means the number of intermediate K-V pairs, and in the table we can see that MapReduce-based FCM’s Map Output Records is related to image size, while the other is only related to gray level, makes the calculation is much less.

<table>
<thead>
<tr>
<th>Images Number</th>
<th>MapReduce-based FCM (records)</th>
<th>MapReduce-based FFCM (records)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>46,335,890</td>
<td>2,560</td>
</tr>
<tr>
<td>200</td>
<td>185,343,560</td>
<td>102,400</td>
</tr>
</tbody>
</table>

In Figure 12, though both the traditional FFCM and the MapReduce-based FFCM are efficient, there is difference between them, detailed comparison is shown in Figure 13. From (a) we can see that both running times are increasing with a near-linear manner, but MapReduce-based FFCM’s time is much less than the traditional one. However, in more detailed local figure, (b), we can see that in fact, the growing pattern of our algorithm isn’t a near-linear way, but more like a logarithm one. Also Table III shows the efficiency improvement (at least 13%) of our algorithm when compared to traditional methods, and the larger the file size is, the higher improvement our proposed algorithm can achieved.
Figure 13 Running times of the two FFCM algorithms

We list several groups of running time comparisons in Table III, from which, we can see that at relative large file size, our proposed algorithm provides a more efficient segmentation than traditional non-parallel one; but if the image dataset is small, our proposed algorithm would consume more time. This phenomenon may be caused by the start-up and clean-up time of the MapReduce process, the two MapReduce stages would cost about 3 to 4 seconds in once iteration. The more detailed analysis of running times of the two methods is based on the experimental results. To the traditional non-parallel FFCM, its processing time is linear function of images number (HIB file size). But as to our proposed algorithm, there is a difference situation, and it deserves to be analysed as follows.

<table>
<thead>
<tr>
<th>File Size (GB)</th>
<th>FFCM (Seconds)</th>
<th>MR FFCM (Seconds)</th>
<th>Efficiency Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.45</td>
<td>532.863</td>
<td>621</td>
<td>-16.5%</td>
</tr>
<tr>
<td>1.78</td>
<td>2137.072</td>
<td>1852</td>
<td>13.3%</td>
</tr>
<tr>
<td>3.57</td>
<td>4175.288</td>
<td>3307</td>
<td>20.8%</td>
</tr>
<tr>
<td>5.36</td>
<td>6412.488</td>
<td>5006</td>
<td>21.9%</td>
</tr>
<tr>
<td>7.15</td>
<td>8422.957</td>
<td>6488</td>
<td>22.9%</td>
</tr>
</tbody>
</table>

The linear increasing manner of our proposed algorithm that shown in Figure 13(a) is caused by lacking sufficient computing resource, means the mappers, as our cluster is just a three-node one. Therefore, as we can see in (b) that when the required mappers number is below 6, the maximum number our cluster can provide, the time increasing pattern isn’t linear anymore; but if it requires more than 6 mappers, our cluster can not provide so much, so some map tasks need to wait for the re-allocation of compute resource, which makes the processing time increasing again, as shown as the situation in (b) when file size is between 223MB to 446MB. As a result, when file size increases further, cluster will keep running on its maximum capability, so the running time will again be near-linear.

In Section IV, we analyzed our algorithm’s processing time based on an assumption that the cluster is large enough. In the experiment, such assumption can only be satisfied when the required mapper is below 6, means file size is below 384MB. Here we select a file whose size is 178MB and images number is 800, so required mappers number is 3. In this situation, we can calculate when our algorithm will outperform the traditional methods in ideal case. The traditional method’s running time is 212 seconds as we recorded, so based on equation (14), one image’s processing time is about 0.265 seconds. As to our proposed algorithm, map and reduce functions time in one iteration is 22 and 8 seconds in average, so based on equation (19) and (20), we can obtain that:

\[ t'_1 \text{\_oneRound} = O(k \cdot n) + 2 \cdot O(k' \cdot c \cdot l) \approx 22 \]

\[ t'_i \text{\_oneRound} = O(k' \cdot l) \approx 8 \]  

Therefore, \( O(k \cdot n) \approx 6 \). So if iteration time is still 12, the processing times of map function and reduce function are \( t'_i = 12 \cdot 22 + 6 = 270 \) and \( t'_i = 12 \cdot 8 = 96 \). Combined with time \( t'_{\text{fixed}} \approx 4 \cdot 12 = 48 \), the processing time of our algorithm when
cluster reaches its maximum capability is $t_{\text{MR}} = 414$, which is close to the value 495 obtained from experiment. In this way, our algorithm will outperform the traditional methods when:

$$t_{\text{traditional}} = m \cdot 0.265 > t_{\text{MR}} \approx 414 \Rightarrow m > 1563$$  \hspace{1cm} (24)

In equation (24), when images number exceeds a threshold, and the assumption mentioned earlier can be satisfied, our proposed algorithm will provide a more efficient segmentation than traditional methods. However, the problem is that such analysis is just a theoretical estimation. In real world, or in the experiment, since lacking sufficient mappers, the threshold will be pulled to a higher value, and execution of algorithm will be different from theoretical situation. Since the lacking of mapper will lead to the longer of map tasks waiting queue, the running time of map function will then increase with the file size. The map and reduce function’s running times growth curves are shown in Figure 14. In which, we defined the running time of reduce function is from the start time to the finish time, so it contains the time spent on waiting for the completion of map function, which occupies a large part in the processing time of reduce function, also makes the map function’s running time occupies a large part in entire job’s running time.

![Figure 14. Running times growth curve of our algorithm and its map function and reduce function](image)

In map stage, when the problem of lacking enough mappers happens, some tasks need to wait for the re-allocation of the computing resource, so they can’t be executed immediately, making the map function’s running time increase. The situation also makes reduce function stay in wait state, so its time is also increasing, and has a similar manner as the map stage. But at the beginning of the curve, we find some constant points. They appears when file size is small, means all reduce tasks are located in same rack with map ones, there is no time delay between them, so reduce stage’s time keeps fixed. Therefore, in fact, reduce stage’s time is a small part in the whole time, and the bottleneck of our proposed algorithm is map function. To solve this problem, we can add more nodes into cluster, to reduce time spent on map stage and improve the efficiency.

Another experiment has been done then, to test the scalability of our algorithm, on different cluster size, as Figure 15 shows. In the experiment, we change our cluster size to two-node, three-node and five-node one. As we can see in Figure 15(a), the larger the cluster size is, the less running time it will be, at same file size, means with adding more nodes, performance of our proposed algorithm will be improved. Also when file size is growing, the performance difference is more obvious, means that the larger size cluster is more powerful.

![Figure 15. Scalability test](image)

(a) File sizes range from 1.1MB to about 7.1GB
Hence, in the future, if more images or videos are produced, to provide a more efficient segmentation or a more powerful processing capability, we just need to scale out the cluster size by adding more nodes, which is a easy work.

On the other hand, when file size is small, running times of the three clusters have no difference with each other. Because in this situation, all clusters are powerful enough to process the images; so there will be no tasks waiting in map function, makes the running time has no relationship with cluster size. But if the cluster is becoming less powerful, running times will increase again and are related to cluster size, as in (b) when file size is between 223MB to 892MB. In this range, the two-node cluster’ time firstly begins to increase, and then is three-node one, the five-node cluster is the last.

So in general, the algorithm’s running time increasing on different cluster size can be divided into three parts: when all clusters have enough mappers, there is no obvious difference in running times; when the required mappers’ number exceeds the capability of some clusters, times of such clusters begin to increase; when all clusters can’t provide enough mappers, the larger the cluster size is, the better performance it can obtain.

And when compared with the work of Wang et al. [18], our proposed algorithm shows its superiority on large-scale image dataset. Image size in their experiment is $768 \times 576$, so we use images at the same size. Based on them, we generate three HIB files, containing 8,000, 12,000 and 16,000 images respectively. Experiment is conducted on the five-node cluster, the running times results are reported in Table IV. In which, one image’s running time is an average value that dividing the total time by the number of images.

<table>
<thead>
<tr>
<th>File size (MB)</th>
<th>Total running time (Seconds)</th>
<th>One image’s running time (Seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>487.6</td>
<td>584</td>
<td>0.073</td>
</tr>
<tr>
<td>731.5</td>
<td>679</td>
<td>0.057</td>
</tr>
<tr>
<td>975.3</td>
<td>767</td>
<td>0.048</td>
</tr>
</tbody>
</table>

In Table IV, we can find that the larger the file size is, the less mean value we can get, proving the algorithm’s advantage on processing larger-scale image dataset. Then compared with times results from work of Wang et al., our results are still comparable and even better, as Table V shows.

<table>
<thead>
<tr>
<th>Processing time of our algorithm (Seconds)</th>
<th>Processing time of the work of Wang et al. (Seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.073 (8,000 images)</td>
<td>0.140 (Three-prism)</td>
</tr>
<tr>
<td>0.057 (12,000 images)</td>
<td>0.126 (Sphere)</td>
</tr>
<tr>
<td>0.048 (16,000 images)</td>
<td>0.132 (Four-prism)</td>
</tr>
<tr>
<td>Average 0.059</td>
<td>Average 0.133</td>
</tr>
</tbody>
</table>

In Table V, our algorithm’s result is much less than results from work of Wang et al.. Processing time improvement in this situation is at about 55%. It demonstrates that our algorithm is particularly useful on large-scale images processing. And since the experiment is just on a five-node cluster, if we add more nodes into it, further performance improvement can be gotten.
VI. CONCLUSIONS

In this paper, to solve the large-scale images segmentation problem in ocean research, a MapReduce-based FFMC method was proposed. Unlike previous works that focused more on single image and accuracy, we made a tradeoff between the segmentation accuracy and efficiency. To achieve that, we focused on the parallel processing of large-scale images. A two-layer distribution model was proposed, in which the first layer would group the images into HIB files and the second layer, the core of our work, would use an iterative MapReduce process to realize the FFMC segmentation algorithm. Besides developing the map and reduce functions, we also proposed a combined segmentation of similar images, may substantially save running time and improve efficiency. Experiments were performed to evaluate the performance of our algorithm. Both the effectiveness and efficiency of our proposed algorithm were demonstrated; for instance on a three-node cluster, our proposed algorithm provided a 13% improvement over the traditional non-parallel segmentation methods.

Although our proposed algorithm shows its advantage on processing large-scale images, some issues pertaining to underwater image segmentation must be still considered. For example, reducing the affection from planktons or aquatic plants on segmentation accuracy, as mentioned earlier. The affection will exists in areas, such as detection an recognition of objects of interest. The illumination and deformation of the body attitude will affect the segmentation process and the further analysis, as well. Therefore, identifying more effective features and retaining more information in the resultant images is very important. And our future work will focus on the aforementioned issues, along with the further improvement of our proposed algorithm.

ACKNOWLEDGMENTS

This research was partly supported by National Natural Science Foundation of China (NSFC, Project No.: 71171121/61033005) and National ‘863’ High Technology Research and Development Program of China. (863 Project No.: 2012AA09A408).

REFERENCES


