Hindawi
Scientific Programming
Volume 2017, Article ID 5081526, 10 pages
https://doi.org/10.1155/2017/5081526

Research Article

Parallelizing Gene Expression Programming Algorithm in Enabling Large-Scale Classification

Lixiong Xu, Yuan Huang, Xiaodong Shen, and Yang Liu

School of Electrical Engineering and Information, Sichuan University, Chengdu 610065, China

Correspondence should be addressed to Yang Liu; yang.liu@scu.edu.cn

Received 19 October 2016; Accepted 23 January 2017; Published 20 February 2017

Academic Editor: Alex M. Kuo

As one of the most effective function mining algorithms, Gene Expression Programming (GEP) algorithm has been widely used in classification, pattern recognition, prediction, and other research fields. Based on the self-evolution, GEP is able to mine an optimal function for dealing with further complicated tasks. However, in big data researches, GEP encounters low efficiency issue due to its long time mining processes. To improve the efficiency of GEP in big data researches especially for processing large-scale classification tasks, this paper presents a parallelized GEP algorithm using MapReduce computing model. The experimental results show that the presented algorithm is scalable and efficient for processing large-scale classification tasks.

1. Introduction

In recent years, Gene Expression Programming (GEP) [1] algorithm has been widely studied due to its significant function mining ability. Comparing to the other machine learning algorithms such as support vector machine and neural networks, the most remarkable characteristic of GEP is that it can explicitly mine the mathematical equation f of the dependent variable $\mathbf{y}$ and independent variables $\mathbf{x}$ from dataset. As a result, the equation $\mathbf{y} = f(\mathbf{x})$ can be easily stored and employed in future study of the data. Similar to the Genetic Algorithm, GEP algorithm also simulates the processes of biological evolution to mine a function with the best fitness to represent the data relations. During the evolution, the algorithm employs selection, crossover, and mutation operations to generate offspring. Each individual of the offspring is assessed by a fitness function. The individual that has a better fitness has a higher chance to be selected to produce a next generation. The evolution keeps evolving until a satisfied function that can describe the data relations is found.

As an effective data analyzing approach, classification has been researched a lot. The classification algorithms, especially supervised classification algorithms, for example, artificial neural networks (ANNs), show remarkable classification abilities. However, ANNs are also function-fitting algorithms fundamentally, although the algorithms cannot output the functions explicitly as GEP does. This point motivates us that GEP can also be employed to deal with the supervised classification tasks using the following idea:

(1) Let the training data be $\mathbf{x}$ and the encoded classes be $\mathbf{y}$.

(2) Train the GEP algorithm using $\mathbf{x}$ and $\mathbf{y}$ to mine a function f.

(3) Input the to-be-classified data $\mathbf{x_t}$ to f; observe the output $\mathbf{y}_c$, which can represent the classes that $\mathbf{x_t}$ should belong to. Therefore, $\mathbf{x_t}$ is classified.

It also motivates us that, based on the works [2, 3], GEP-based classification has great potential to be further applied into large-scale classification tasks.

Unfortunately several works [4–6] pointed out that to process large-scale tasks using GEP may encounter the low efficiency issue. The reason is that, as a heuristic algorithm, GEP needs an extremely long time to mine the best-fitted function for large volume of data. Therefore to improve the large-scale classification efficiency using GEP is also focused by this paper. As a result, this paper presents a parallelized GEP algorithm in enabling large-scale classification. The algorithm is designed and implemented in the MapReduce distributed computing environment. Following a number of tests based on standard benchmark datasets have been carried out. The experimental results reveal that the parallelized GEP algorithm shows advantages in dealing with large-scale classification tasks.

The rest of the paper is organized as follows: Section 2 reviews the related work; Section 3 presents the parallelization of GEP; Section 4 discusses the experimental results; and Section 5 concludes the paper.

2. Related Work

As an effective function mining algorithm, GEP has been widely applied in numbers of researches. Sabar et al. [7] employed GEP to design a hyperheuristic framework in order to solve the combinatorial optimization problems. Their experimental results show that the proposed framework has great potential to solve the problems. Hwang et al. [8] employed GEP to predict the Qos (Quality of Service) traffic in the Ethernet passive optical network. The authors combined GEP algorithm to tackle the queue variation during waiting times as well as reducing the high priority packet delay. Deng et al. [9] also employed GEP and rough set to assess the security risk in cyber physical power system. Based on their studies, security risk levels of cyber physical power system can be accurately predicted.

However, several works [4–6] pointed out that GEP has low efficiency issue for processing complicated tasks. To solve the issue, several researchers focused on improving the algorithm parameters of GEP. Xue and Wu [10] proposed Symbiotic Gene Expression Programming (SGEP) based on symbiotic algorithm, estimation of distribution algorithm, and evolution processes improved GEP. The experimental results indicate that SGEP outperforms GEP in terms of efficiency. Chen et al. [11] pointed out that the most computationally expensive computation of GEP is the evolution in the expression tree. Therefore they proposed Reduced-GEP algorithm which is based on the chromosome reduction. The experimental results show that the algorithm is effective and efficient in calculating the fitness and reducing the size of chromosome. In research [12], inspired by the diversity of chromosome arrangements in biology, an unconstrained encoded Gene Expression Programming was proposed. The approach can enlarge the function searching space, which enhances the parallelism and the adaptability of the standard GEP algorithm.

Another effective way of solving the efficiency issue of GEP is to use parallel computing or distributed computing. Du et al. [13] proposed an asynchronous distributed parallel GEP algorithm. They aimed at speeding up the convergence of finding the optimal solution using MPI (Message Passing Interface) [14]. In each processor, a standalone GEP algorithm is running. And then the processors exchange their best individuals and continue evolving. Until a termination message is sent to the processors, the algorithm stops. Based on the experimental results, the authors claimed that the proposed algorithm can greatly speed up the algorithm convergence. However, they have not evaluated the algorithm using large volume of data. And also MPI is highly depending on the homogeneous hardware environment, which limits the algorithm adaption. Du et al. also proposed a MapReduce [15] based distributed GEP algorithm to process large populations and datasets. Similar to [13], each Map computes the fitness and in each Reduce the selection, mutation, and crossover operations are executed. The output is output into the distributed file system where the exchanges of the best individuals occur. Although the authors claimed that they achieved algorithm speedup, two issues should be discussed. Firstly the algorithm needs a large number of Reduces, which generates system overhead because of IO operations [2] in reducers. Secondly their algorithm needs a large number of iterations. However, MapReduce does not support iteration originally. Instead, the algorithm has to submit a number of MapReduce jobs to the cluster, which generates tremendously large overhead [3]. Browne and dos Santos [16] also discussed the parallelization of GEP using the island model. However, their algorithm does not focus on the parallelization. And the algorithm performance for processing the large volume of dataset has not been evaluated.

The improvement of GEP presented in this paper mainly focuses on parallelizing the GEP algorithm for executing large-scale classification. Our algorithm first employs the Hadoop framework [17] as the underlying infrastructure. And secondly combining with the ensemble techniques, the algorithm is able to supply efficiency, scalability, and accuracy.

3. Algorithm Design

3.1. Classification Using GEP. Based on the selection, crossover, mutation, and fitness, GEP is able to mine a function from the given dataset. Therefore, let T denote the training dataset; t_i denote an instance in T; l denote the length of t_i; j denote the jth class of T; T_e denote the testing dataset; t_{e_i} denote an instance in T_e; c_j denote the coded identifier of class; σ denote a threshold; r represent the number of correctly classified instances; and n represent the number of training instances. The classification using GEP is shown in Algorithm 1.

3.2. MapReduce and Hadoop. MapReduce computing model contributes two main functions Map and Reduce to facilitate the development of the distributed computing applications. Map function executes main computation and Reduce function collects the intermediate output of Maps and generates the final output. Each Map computes the data instances one by one in the form of key-value pair $\{K1, \text{Value1}\}$. And then the computed result is output as an intermediate output $\{K2, \text{Value2}\}$. The Reduces collects the intermediate output of all Maps. Afterwards each Reduce merges the inputs having the same key and generates the final output.

Hadoop framework [17, 18] is a Java based implementation of MapReduce. Two types of nodes including one NameNode and several DataNodes consist of a typical Hadoop cluster. The NameNode manages the metadata, whilst the DataNode executes a number of Map (mappers) and Reduce (reducers) operations in parallel. Both the NameNode and DataNodes contribute their resources including processors, memory, hard disks, and network adaptors to form the Hadoop

```
(1) For each $t_i \in T$
    Encode $c_j$ representing the class of $t_i$
(2) For each $t_i$ in $T$
        Input $t_i$ and $c_j$ into GEP
(3) Let $c_j$ be $\mathbf{y}$ and $t_i$ be $\mathbf{x}$.
(4) Initiate GEP components: function set, link function, selection
    mutation, crossover and fitness $r/n$.
(5) In each generation, GEP mines $c = f(t_i)$,
    if $|c - c_j| \le \sigma$
        $r = r + 1$
(6) GEP keeps running
    until terminating condition (determined generation or fitness) met
(7) Output $f$ which represents $c_j = f(t_i)$
(8) Let $\sigma$ denote a threshold
    For each $t_{e_i}$ in $T_e$
        Compute $y_e = f(t_{e_i})$
        if $|y_e - c_j| \le \sigma$
            $t_{e_i} \in c_j$
        else
            $t_{e_i}$ is an outlier
(9) Classification terminates
```

Algorithm 1: The pseudocode of classification using GEP.

Distributed File System (HDFS) [17]. HDFS not only is responsible for high performance data storage but also manages data processing for the mappers and reducers, which supplies fault tolerant, load balancing, scalability, and heterogeneous hardware support. Figure 1 shows the structure of Hadoop framework.

3.3. Parallelization of GEP in Enabling Large-Scale Classification. In the training phase, let m denote the number of mappers. Therefore, the training dataset T can be divided into a number of m data chunks, in which each chunk is represented by T_i $i \in m$. T_i satisfies

$$\bigcup_{i=1}^{m} T_i = T. \tag{1}$$

Firstly m mappers input m data chunks in parallel. And then each mapper initiates one sub-GEP starting the function mining according to the input training data. As long as a function (a classifier) has been mined in each mapper, totally a number of m classifiers are generated.

In the classification phase, the testing dataset T_e is also divided into m chunks. Each previously trained classifier inputs one testing data chunk and executes the classification. Therefore, the classification using GEP can be parallelized. However, one problem should be mentioned that, due to the data separation of the training dataset, each sub-GEP in each mapper is only trained by a subset of the original dataset. The insufficient training may lead to the loss of classification accuracy. In order to parallelize GEP classification avoiding accuracy loss, ensemble techniques including bootstrapping and majority voting are employed.

Bootstrapping is based on the idea of controlling the number of times that the training instances appear in the bootstrapping samples, so that, in the number of B bootstrapping samples, each instance appears the same number of times [19]. To create balanced bootstrapping samples, the following steps could be followed:

(1) Construct a string of the instances $X_1, X_2, X_3, \ldots, X_n$ repeating B times so that a sequence $Y_1, Y_2, Y_3, \ldots, Y_{Bn}$ is achieved.

(2) A random permutation p of the integers from 1 to B_n is taken. Therefore the first bootstrapping sample can be created from $Y_p(1), Y_p(2), Y_p(3), \ldots, Y_p(n)$, moreover the second bootstrapping sample from $Y_p(n+1), Y_p(n+2), Y_p(n+3), \ldots, Y_p(2n)$, and so on.

(3) Repeat step (2) until $Y_p((B-1)n+1), Y_p((B-1)n+2), Y_p((B-1)n+3), \ldots, Y_p(Bn)$, which is the Bth bootstrapping sample.

Based on the bootstrapping, the instance distributions of the original dataset can be simulated in the samples, in which the original data information can be kept more. The majority voting is a commonly used combination technique. The ensemble classifier predicts a class for a test instance that is predicted by the majority of the base classifiers [20].

By employing the bootstrapping and majority voting, the parallelized GEP classification algorithm works as follows.

In the training phase:

(1) The algorithm firstly generates a number of m bootstrapped sample sets using the training dataset T. Each set T_i is saved in one data chunk stored in the HDFS.

(2) Each mapper initiates a sub-GEP and inputs one data chunk from HDFS.

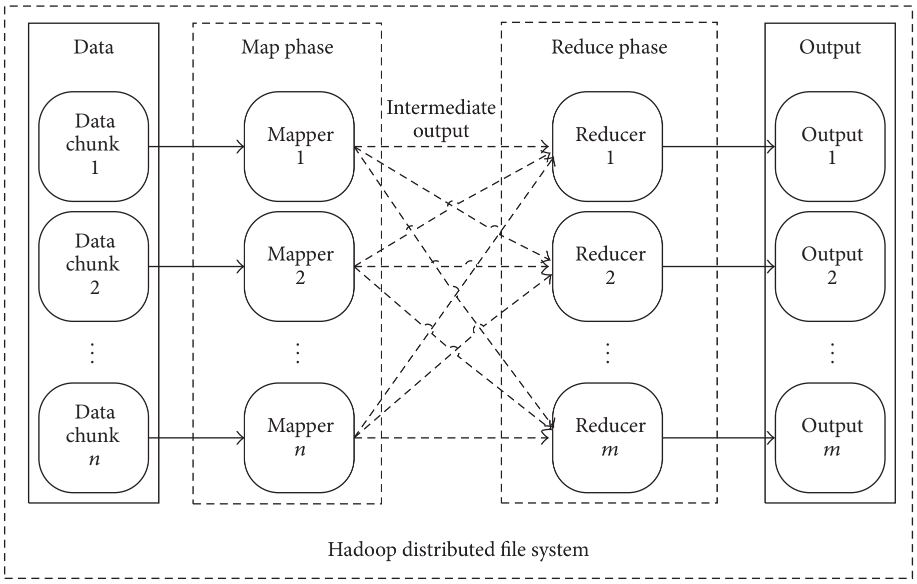


Figure 1: The structure of Hadoop framework.

(3) Each mapper trains its sub-GEP according to step (1) to (6) in Algorithm 1. As long as the training terminates, the mined function f_i is collected by reducer and saved into HDFS in the value pairs $\langle mapperId, f_i \rangle$.

(4) As in each mapper its sub-GEP mines the function individually, therefore, finally a number of m different functions can be saved, which means a number of m weak classifiers are created.

Figure 2 shows the training phase of the algorithm.

In the classification phase:

(1) Each mapper retrieves one function f_i from HDFS so that the mapper becomes one weak classifier.

(2) And then all the mappers input the same one instance t_{e_i} from the testing dataset T_e.

(3) In each mapper, when t_{e_i} is input, a value of c can be computed according to f_i. Comparing the values of c_j, σ, and c according to step (8) in Algorithm 1, t_{e_i} can be classified.

(4) The gth mapper outputs its intermediate output $\langle t_{e_i}, result_g \rangle$.

(5) One reducer collects all the intermediate outputs from all the m mappers. And then, it merges the outputs having the same key into one group. In the group, the majority voting is executed to vote the final classification result.

Figure 3 shows the classification phase.

4. Algorithm Evaluation

4.1. Experimental Environment. In order to evaluate the algorithm performance, a physical Hadoop cluster constituted by one NameNode and four DataNodes is established. The details of the cluster are listed in Table 1.

Table 1: The cluster details.

NameNode	CPU: Core i7@3 GHz Memory: 8 GB SSD: 750 GB OS: Fedora
DataNodes	CPU: Core i7@3.8 GHz Memory: 32 GB SSD: 250 GB OS: Fedora.
Network bandwidth	1 Gbps
Hadoop version	2.3.0

Table 2: The dataset details.

Type	Iris	Wine
Dataset characteristics	Multivariate	Multivariate
Instance number	150	178
Attribute number	4	13
Class number	3	3

The datasets employed in the experiments are Iris dataset [21] and Wine dataset [22]. The details of the datasets are listed in Table 2.

The parameters of GEP used in the experiments are listed in Table 3.

4.2. Accuracy of the Classification. Let *rightNum* represent the number of the correctly classified instances and *wrongNum*

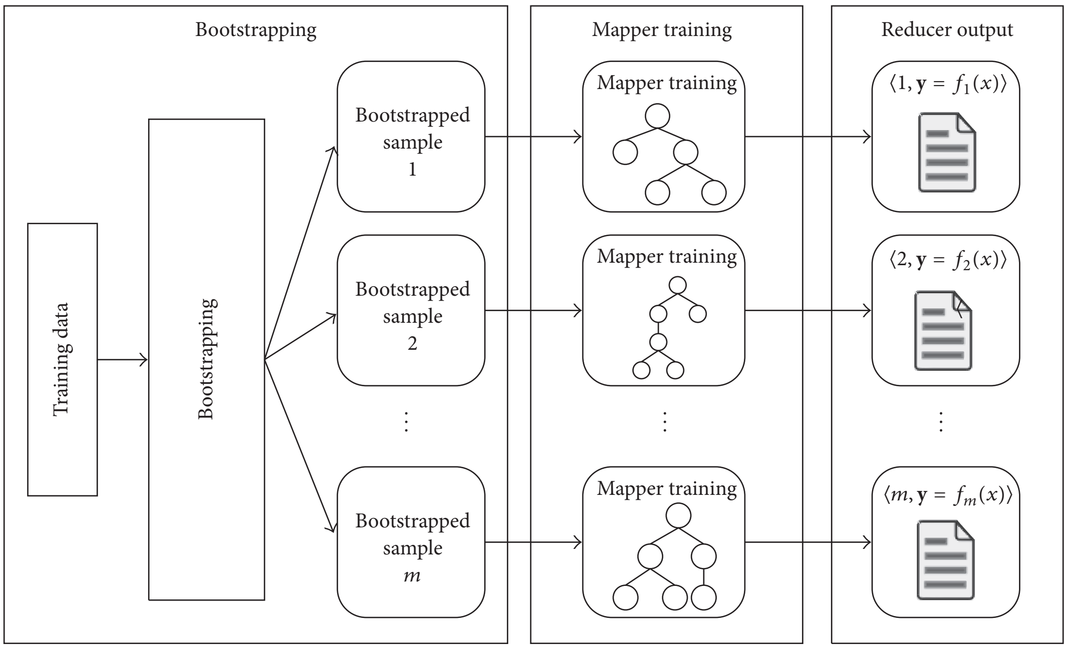


Figure 2: Training phase in the classification.

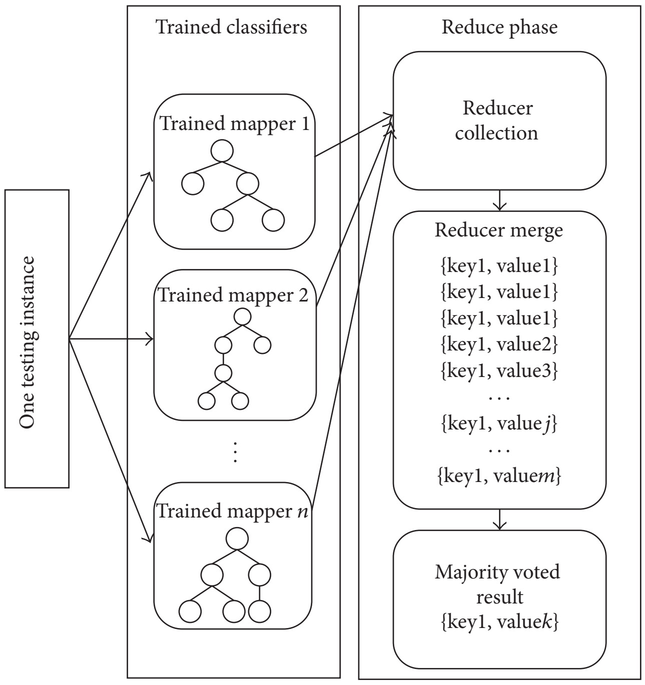


Figure 3: The classification of the presented algorithm.

Table 3: The details of the parameters of GEP.

Number of genes	2 (iris)/4 (wine)
Linking function	+
Head length	6
Function set	+ − ∗/cos sin tan exp log sqrt abs
Fitness	r/n
Terminal set	*abcd* (iris)/*abcdefghijklm* (wine)
Population size	100
Mutation rate	0.044
IS transposition rate	0.1
RIS transposition rate	0.1
Gene transposition rate	0.1
One-point recombination rate	0.4
Two-point recombination rate	0.2
Gene recombination rate	0.1
Threshold σ	0.5
Coded classes	1, 2, and 3

represent the number of the wrongly classified instances. Therefore the classification accuracy p is defined as

$$p = \frac{rightNum}{rightNum + wrongNum} \times 100\%. \qquad (2)$$

In the following tests, increasing numbers of instances are selected from the two datasets as the training instances, whilst the rest instances are the testing instances. The bootstrapping number is four and the algorithm starts eleven mappers for executing the parallelized GEP classification. The experimental results are shown in Figures 4–14 and *The Functions in Each Sub-GEP for Classifying Iris Dataset* and *The Functions in Each Sub-GEP for Classifying Wine Dataset* in the appendix.

Figure 4 shows the classification accuracy of the Iris dataset with an increasing number of the training instances from 12 to 105. It can be observed that the parallel GEP algorithm performs highly stable and outperforms the standalone GEP algorithm. The visualizations of the classification results

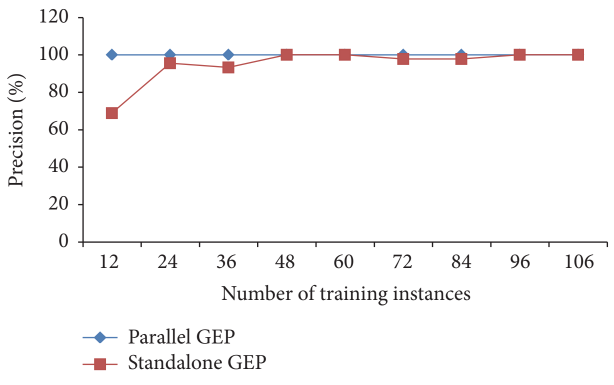


FIGURE 4: The precision comparison for classifying Iris dataset.

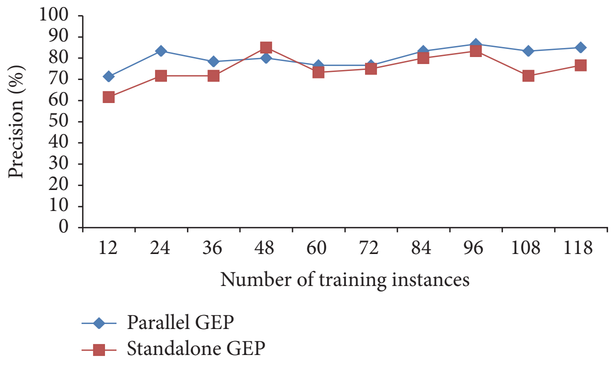


FIGURE 5: The precision comparison for classifying Wine dataset.

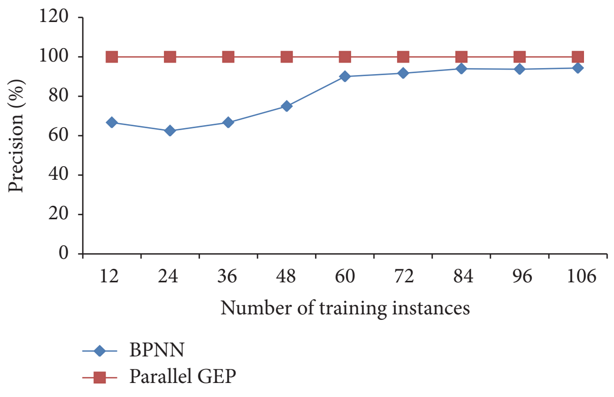


FIGURE 6: The precision comparison for classifying Iris dataset using parallel GEP and BPNN.

are shown by Figures 11, 12, and *The Functions in Each Sub-GEP for Classifying Iris Dataset* in the appendix.

Wine dataset has also been employed to evaluate the classification accuracy. Comparing to Iris dataset, each instance of Wine dataset has 13 attributes, which may impact the classification accuracy. The experimental result is shown in Figure 5.

Figure 5 shows the classification accuracy of the Wine dataset with an increasing number of the training instances from 12 to 118. The result shows that, due to more attributions of the instances, the accuracy of the parallel GEP starts fluctuating. However, in most of the tests the parallel GEP still outperforms the standalone GEP in terms of accuracy. It further indicates the ensemble techniques can help to improve the classification accuracy. The visualizations of the classification results are shown by Figures 13, 14, and *The Functions in Each Sub-GEP for Classifying Wine Dataset* in the appendix.

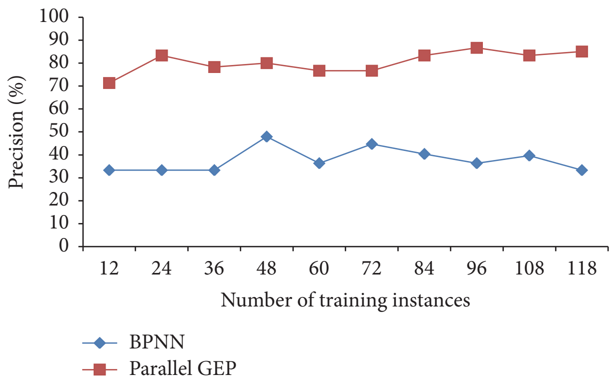


FIGURE 7: The precision comparison for classifying Wine dataset using parallel GEP and BPNN.

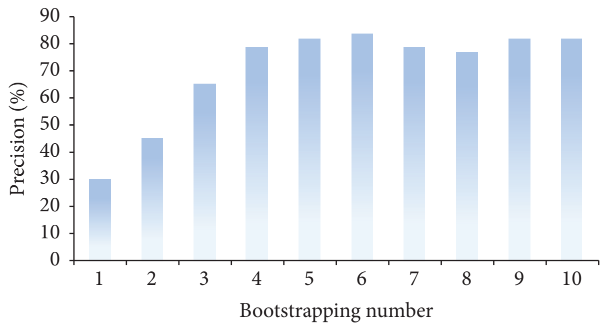


FIGURE 8: Precision of the classification with increasing bootstrapping numbers.

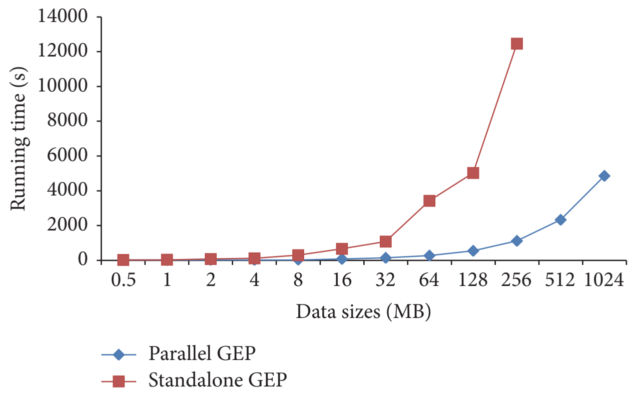


FIGURE 9: Algorithm efficiency comparison of increasing training data sizes.

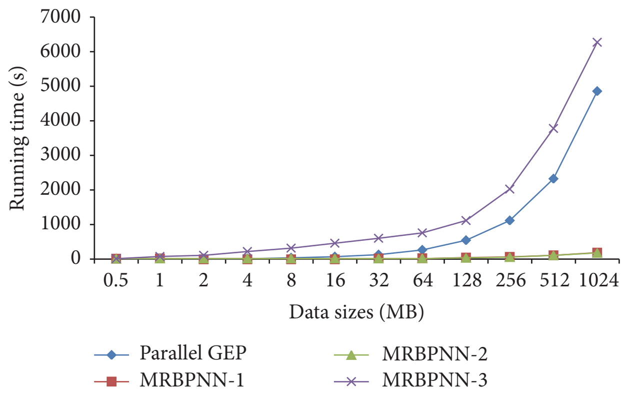


FIGURE 10: Comparison of the processing time of the parallel BPNNs and the parallel GEP with increasing training data sizes.

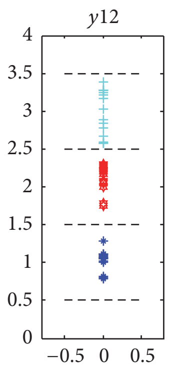


FIGURE 11: The classification result of the standalone GEP using Iris dataset.

For further evaluating the effectiveness of the proposed algorithm, we also implemented backpropagation neural network (BPNN). The comparisons of the classification accuracy are shown in Figure 6.

Figure 6 indicates that, in terms of Iris dataset classification accuracy, the parallel GEP algorithm outperforms BPNN. Although the neural network also performs well in the classification, when the number of the training instances is small, it gives lower classification accuracy.

Figure 7 indicates that, in terms of Wine dataset classification accuracy, the parallel GEP algorithm greatly outperforms BPNN. Due to more attributions in the dataset, it is difficult for BPNN to correctly classify most of the testing instances. Contrarily, parallel GEP can still keep a higher accuracy.

It should be noticed that the bootstrapping number which represents the number of times the training instances appear in the bootstrapping samples also impacts the algorithm accuracy. Therefore Figure 8 is generated to show the classification results with increasing bootstrapping numbers. Wine dataset is selected as the experimental dataset, in which a number of 118 instances are the training instances and the remaining 60 instances are the testing instances.

In Figure 8, it can be observed that when the bootstrapping number is less than 6, the classification precision keeps increasing. And then, the precision varies slightly. Figure 8 significantly tells that initially enlarging the bootstrapping number improves the classification. However, when the bootstrapping number reaches a certain value, the performance in terms of classification precision cannot be further improved.

4.3. Running Time of the Classification. In this section, Wine dataset is selected as the experimental dataset. The algorithm processing time for the increasing training data sizes has been evaluated. In the following tests, firstly the bootstrapping number is 4, which means each training instance appears 4 times. The number of the training instances is 118 whilst the testing instances remain 60. And then the training data size is duplicated from approximately 0.5 MB to 1024 MB. It should be pointed out that, because of the duplication, the bootstrapping number will change from 4 to 4n, where n represents the duplicated times. However, this section only focuses on the algorithm efficiency. Therefore although the varying bootstrapping numbers may affect the classification precision slightly according to Figure 8, the algorithm processing time with increasing training data sizes is highlighted in Figure 9.

Figure 9 shows that when the training data size is small, the performances of the standalone GEP and the parallel GEP are nearly the same. However, when the data size becomes larger, the parallel GEP outperforms the standalone GEP. When the data size increases more than 256 MB, the standalone GEP cannot finish the classification due to memory limitation. Contrarily, the parallel GEP still works fine even if the data size increases to 1024 MB.

To further compare the classification efficiency to the other classification algorithms, the MapReduce based parallel bac propagation neural network algorithms (MRBPNN 1, 2, and 3) [2] are also implemented. The comparisons are shown in Figure 10.

Figure 10 shows that in terms of the algorithm running time, the parallel BPNN algorithms MRBPNN 1 and 2 outperform the parallel GEP. The main reason is that GEP needs longer time to evolve. Contrarily, MRBPNN 1 and 2 need shorter time to train the neurons. Although parallel GEP performs slower than MRBPNN 1 and 2 do, it can supply higher classification accuracy according to Figures 6 and 7.

5. Conclusion

This paper presents a MapReduce and ensemble techniques based parallel Gene Expression Programming algorithm in enabling large-scale classification. The parallelization of GEP mainly focuses on paralleling the training phase (function mining phase) which is the most time consuming and computational intensive process. The experimental results show that the presented algorithm outperforms the standalone GEP and BPNN in terms classification accuracy. In the algorithm executing time evaluations, the presented parallel GEP also shows remarkable performance comparing to the standalone GEP. Although the parallel GEP works slower than MRBPNN 1 and 2, it can supply higher classification accuracy, which enables the presented parallel GEP to be one of the effective tools dealing with large-scale classification.

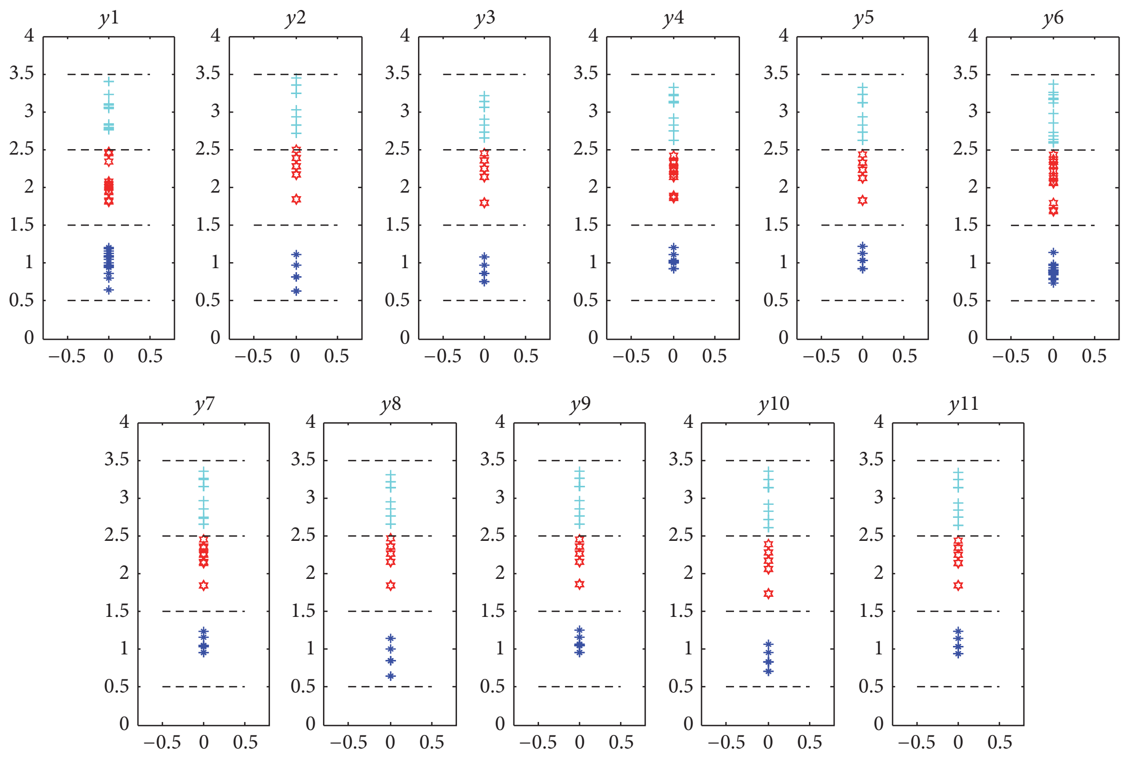


FIGURE 12: The classification results of eleven sub-GEPs.

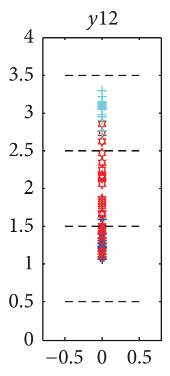


FIGURE 13: The classification result of the standalone GEP using Wine dataset.

Appendix

In the appendix, the details of classifying Iris and Wine dataset have been listed. Figure 11 visualizes the classification result of the standalone GEP for classifying Iris dataset.

Figure 11 indicates that GEP has ability to mine a function f to classify instances into three classes. In this case, the mined function f is represented by $y12$:

$$y12 = (d) + (\cos(\sin(\cos(\text{abs}(((b) * (d)) * (c)))))) \quad (A.1)$$

The Iris dataset classification results of the eleven sub-GEPs employed by parallel GEP are shown in Figure 12.

Figure 12 shows that the eleven sub-GEPs have different classification results due to differently mined functions f. However, because of the majority voting in the reducer, parallel GEP is able to output a correct classification result. The eleven mined functions are listed as follows.

The Functions in Each Sub-GEP for Classifying Iris Dataset

$$y1 = (d) + (\cos(\sin((\tan(\exp(d))) * ((a)/(c)))));$$

$$y2 = (d) + (\sin(\text{sqrt}(\text{sqrt}(d))));$$

$$y3 = (\cos(\cos(\text{abs}(\cos(\cos(-(d))))))) + (d);$$

$$y4 = (\cos(\sin((\log(b))/(\text{sqrt}(b))))) + (d);$$

$$y5 = (d) + (\cos(\sin(\exp(\tan(\exp((b)/(b)))))));$$

$$y6 = (d) + (\text{abs}(\cos(-(-(\sin(\text{sqrt}(a)))))));$$

$$y7 = (-(-(d))) + (\cos((\cos(\sin(b))) * (\cos(\cos(b)))));$$

$$y8 = (\text{sqrt}(\sin(\sin(\text{sqrt}(\sin(d)))))) + (d);$$

$$y9 = (\text{abs}(d)) + (\cos(\cos(\exp(\exp((-(a)) - (c))))));$$

$$y10 = (d) + (\cos(\text{abs}(\text{sqrt}(\cos(\text{sqrt}(\text{sqrt}(d)))))));$$

$$y11 = (\text{abs}(\sin(\exp((d) + (-(d)))))) + (d);$$

Figure 13 visualizes the classification result of the standalone GEP for classifying Wine dataset.

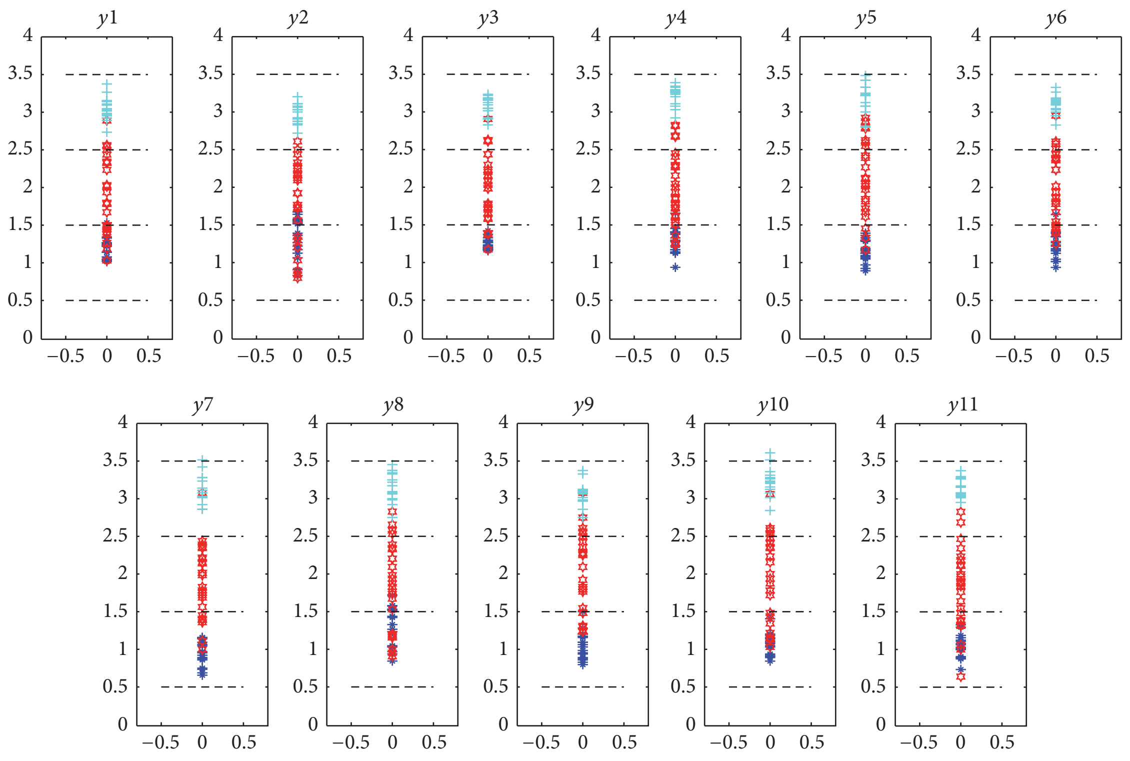


FIGURE 14: The classification results of eleven sub-GEPs.

In this case, the mined function f is represented by $y12$:

$$y12 = \left(\left(\left(\cos\left(\cos\left(\exp\left(\cos\left(\frac{(l)}{(g)}\right)\right)\right)\right)\right) + \left(\cos\left(\cos\left(\cos\left((\cos(k)) - \left(\frac{(j)}{(g)}\right)\right)\right)\right)\right) + (\cos(g))\right) + (\cos(\cos(\exp(k)))) \quad \text{(A.2)}$$

The Wine dataset classification results of the eleven sub-GEPs employed by parallel GEP are shown in Figure 14.

The eleven mined functions are listed as follows.

The Functions in Each Sub-GEP for Classifying Wine Dataset

$y1 = (((\cos(\cos(\mathrm{sqrt}((\mathrm{sqrt}(m)) + (b))))) + (\cos(k))) + (\sin(\sin(\cos(g))))) + (\mathrm{sqrt}(\sin(\mathrm{abs}(\cos(\sin(\tan(k)))))))$

$y2 = (((\cos(\cos(\mathrm{sqrt}(\cos(h))))) + ((\cos(\cos(j))) \ast ((\cos(h)) \ast (\cos(g))))) + (\mathrm{abs}(\sin(\exp(k))))) + (\cos(\cos(\cos(\cos(\cos(\cos(g)))))))$

$y3 = (((\cos(\sin(\cos(\sin(-((a) + (g))))))) + (\sin(\cos(\sin(a))))) + (\sin(\sin(\cos(g))))) + (\cos(\cos((\cos(\mathrm{abs}(g))) + (\sin(a)))))$

$y4 = (((\cos(\cos(\log(((c) \ast (l)) \ast (m))))) + (\cos(g))) + (\cos(\sin(f)))) + (\cos(\cos(\exp(k))))$

$y5 = (((\cos(h)) + (-(\sin(-((\mathrm{sqrt}(d)) \ast ((e)/(m))))))) + (\cos(g))) + (\cos(\cos((\cos(j))/k)))$

$y6 = (((\cos(\mathrm{abs}(\cos(\exp(k))))) + (\cos(\cos(\cos(\cos(\sin(\log(d)))))))) + (\sin(\sin(\cos(g))))) + (\cos(\log((m) - (((k) - (e)) - (e)))))$

$y7 = (((\cos(\mathrm{sqrt}(k))) + (\cos(\sin(\mathrm{abs}((\log(k)) + ((g) - (a))))))) + (\cos(\sin(\mathrm{abs}((\log(m)) + (h)))))) + (\cos(g))$

$y8 = (((\cos(\sin(\sin((j) - ((l)/(a)))))) + (\mathrm{abs}(\sin(\exp(\mathrm{abs}(k)))))) + (\cos(\sin(\sin(\cos((c) + (g))))))) + (\sin(\cos(g)))$

$y9 = (((\mathrm{abs}(\cos(\cos(\log(((m) - (d))/(f)))))) + (\cos(\cos(\exp(k))))) + (\cos(\tan(-(\tan(-(\mathrm{sqrt}(a)))))))) + (\cos(g))$

$y10 = (((\cos(\sin(\cos(\sin(\cos(\sin(i))))))) + (\cos(g))) + (\cos(\sin(\log(((m) + (m)) + ((j) + (j))))))) + (\tan(\sin(\cos(\cos(\exp(k))))))$

$y11 = (((\cos(k)) + (\cos(g))) + (\sin(\exp(\cos((\cos(g)) + (\sin(a))))))) + (\sin(\mathrm{sqrt}(g)))$

Competing Interests

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

Acknowledgments

The authors would like to appreciate the support from National Natural Science Foundation of China (no. 51437003).

References

[1] C. Ferreira, "Gene expression programming in problem solving," in *Soft Computing and Industry*, pp. 635–653, Springer, London, UK, 2002.

[2] Y. Liu, J. Yang, Y. Huang, L. Xu, S. Li, and M. Qi, "MapReduce based parallel neural networks in enabling large scale machine learning," *Computational Intelligence and Neuroscience*, vol. 2015, Article ID 297672, 13 pages, 2015.

[3] Y. Liu, L. Xu, and M. Li, "The Parallelization of back propagation neural network in MapReduce and Spark," *International Journal of Parallel Programming*, pp. 1–20, 2016.

[4] C. Zhou, W. Xiao, T. M. Tirpak, and P. C. Nelson, "Evolving accurate and compact classification rules with gene expression programming," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 6, pp. 519–531, 2003.

[5] S. Dehuri and S.-B. Cho, "Multi-objective classification rule mining using gene expression programming," in *Proceedings of the 3rd International Conference on Convergence and Hybrid Information Technology (ICCIT '08)*, pp. 754–760, November 2008.

[6] J. Wu, T. Li, B. Fang, Y. Jiang, Z. Li, and Y. Liu, "Parallel niche gene expression programming based on general multi-core processor," in *Proceedings of the International Conference on Artificial Intelligence and Computational Intelligence (AICI '10)*, pp. 75–79, October 2010.

[7] N. R. Sabar, M. Ayob, G. Kendall, and R. Qu, "Automatic design of a hyper-heuristic framework with gene expression programming for combinatorial optimization problems," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 3, pp. 309–325, 2015.

[8] I.-S. Hwang, J.-Y. Lee, and A.-T. Liem, "Genetic expression programming: a new approach for QoS traffic prediction in EPONs," in *Proceedings of the 4th International Conference on Ubiquitous and Future Networks (ICUFN '12)*, pp. 249–254, July 2012.

[9] S. Deng, D. Yue, X. Fu, and A. Zhou, "Security risk assessment of cyber physical power system based on rough set and gene expression programming," *IEEE/CAA Journal of Automatica Sinica*, vol. 2, no. 4, pp. 431–439, 2015.

[10] S. Xue and J. Wu, "Gene expression programming based on symbiotic evolutionary algorithm," in *Proceedings of the 2nd International Conference on Artificial Intelligence, Management Science and Electronic Commerce (AIMSEC '11)*, pp. 3055–3058, August 2011.

[11] Y. Chen, C. J. Tang, R. Li, M. F. Zhu, C. Li, and J. Zuo, "Reduced-GEP: improving gene expression programming by gene reduction," in *Proceedings of the 2nd International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC '10)*, pp. 176–179, IEEE, Nanjing, China, August 2010.

[12] J. Zhang, Z. Wu, Z. Wang, J. Guo, and Z. Huang, "Unconstrained gene expression programming," in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '09)*, pp. 2043–2048, May 2009.

[13] X. Du, L. Ding, and L. Jia, "Asynchronous distributed parallel gene expression programming based on estimation of distribution algorithm," in *Proceedings of the 4th International Conference on Natural Computation (ICNC '08)*, pp. 433–437, October 2008.

[14] Message Passing Interface, http://www.mcs.anl.gov/research/projects/mpi/.

[15] X. Du, Y. Ni, Z. Yao, R. Xiao, and D. Xie, "High performance parallel evolutionary algorithm model based on MapReduce framework," *International Journal of Computer Applications in Technology*, vol. 46, no. 3, pp. 290–295, 2013.

[16] N. P. A. Browne and M. V. dos Santos, "Adaptive representations for improving evolvability, parameter control, and parallelization of gene expression programming," *Applied Computational Intelligence and Soft Computing*, vol. 2010, Article ID 409045, 19 pages, 2010.

[17] Apache Hadoop, http://hadoop.apache.org.

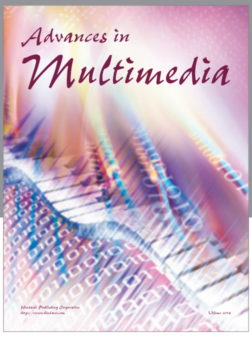
[18] J. Venner, *Pro Hadoop*, Springer, New York, NY, USA, 2009.

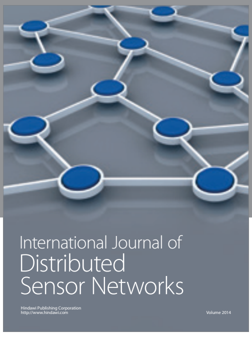
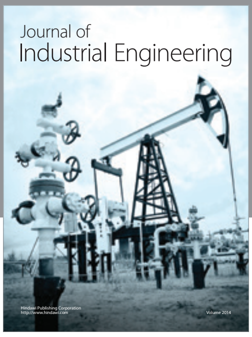
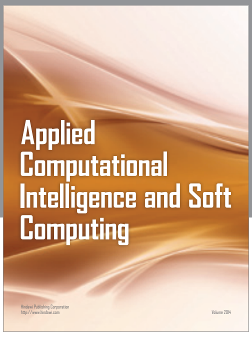
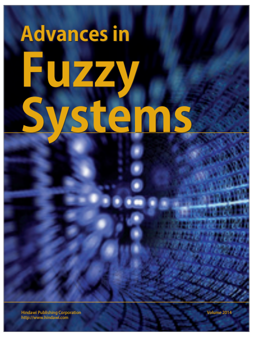
[19] N. K. Alham, *Parallelizing support vector machines for scalable image annotation [Ph.D. thesis]*, Brunel University, Uxbridge, UK, 2011.

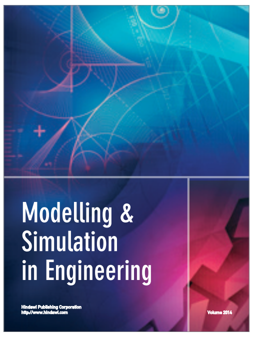
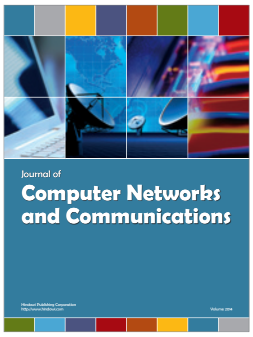
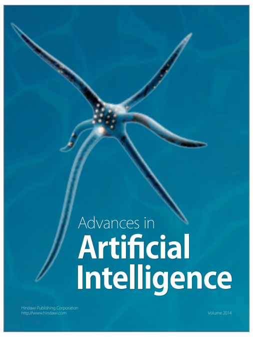
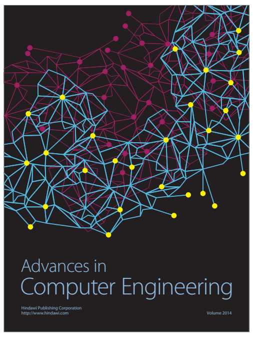
[20] N. K. Alham, M. Li, Y. Liu, and M. Qi, "A MapReduce-based distributed SVM ensemble for scalable image classification and annotation," *Computers and Mathematics with Applications*, vol. 66, no. 10, pp. 1920–1934, 2013.

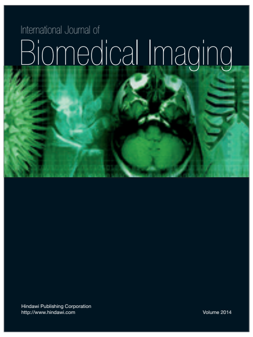
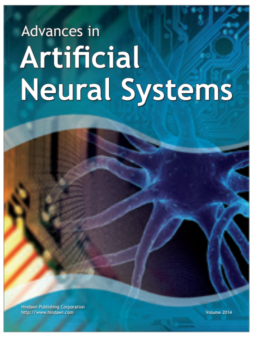
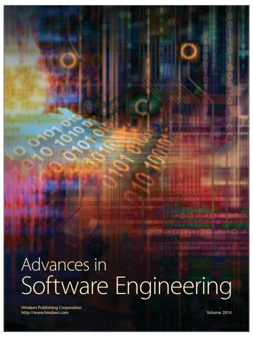
[21] The Iris Dataset, https://archive.ics.uci.edu/ml/datasets/Iris.

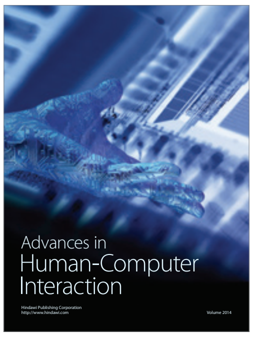
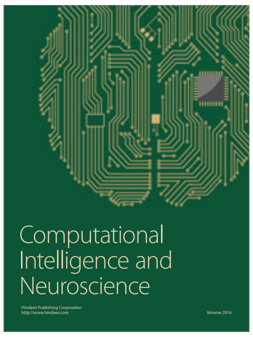
[22] The Wine Dataset, https://archive.ics.uci.edu/ml/datasets/wine.


Advances in
Multimedia

The Scientific World Journal

International Journal of
Distributed Sensor Networks

Journal of
Industrial Engineering

Applied Computational Intelligence and Soft Computing

Advances in
Fuzzy Systems


Modelling & Simulation in Engineering

Journal of
Computer Networks and Communications

Advances in
Artificial Intelligence

Advances in
Computer Engineering

International Journal of
Computer Games Technology

International Journal of
Biomedical Imaging

Advances in
Artificial Neural Systems

Advances in
Software Engineering

Journal of
Robotics

Advances in
Human-Computer Interaction

Computational Intelligence and Neuroscience

International Journal of
Reconfigurable Computing

Journal of
Electrical and Computer Engineering