Received: Added at production | Revised: Added at production | Accepted: Added at production
DOI: xxx/xxxx

ARTICLE TYPE

Parallel Particle Swarm Optimization Classification Algorithm Variant Implemented with Apache Spark

Jamil Al-Sawwa | Simone A. Ludwig

[1]Department of Computer Science, North Dakota State University, Fargo, ND, USA

Summary

With the rapid development of technologies such as the internet, the amount of data that is collected or generated in many areas such as in the agricultural, biomedical, and finance sectors pose challenges to the scientific community because of the volume and complexity of the data. Furthermore, the need of analysis tools that extract useful information for decision support has been receiving more attention in order for researchers to find a scalable solution to traditional algorithms. In this paper, we proposed a scalable design and implementation of a particle swarm optimization classification (SCPSO) approach that is based on the Apache Spark framework. The main idea of the SCPSO algorithm is to find the optimal centroid for each target label using particle swarm optimization and then assign unlabeled data points to the closest centroid. Two variants of SCPSO, SCPSO-F1 and SCPSO-F2, were proposed based on different fitness functions, which were tested on real data sets in order to evaluate their scalability and performance. The experimental results revealed that SCPSO-F1 and SCPSO-F2 scale very well with increasing data set sizes and the speedup of SCPSO-F2 is almost identical to the linear speedup while the speedup of SCPSO-F1 is very close to the linear speedup. Thus, SCPSO-F1 and SCPSO-F2 can be efficiently parallelized using the Apache Spark framework.

KEYWORDS:
Classification, Particle Swarm Optimization, Big Data Analytic

1 | INTRODUCTION

Data mining is a multidisciplinary research area which combines machine learning, statistics, and database research and aims to analyze historical data in order to extract valuable information. Classification is one of the data mining tasks that describes the process of building and training a model using a sophisticated algorithm and a labeled data set that comprises of the input with the corresponding output. The model is applied to a testing data set to predict an accurate outcome for each data instance. In recent decades, classification has been successfully applied in many areas such as agriculture and biomedicine [1,2].

The Swarm intelligence (SI) method is a model that imitates the social behavior of a group of individuals and how they interact with each other and with their environment to achieve a particular goal, e.g. finding the best food source. In the group of individuals, there is no leader to guide the individuals; each individual somehow behaves to some extent randomly within the search space and share information with its neighbors until the best source of food in a certain area is discovered [3,4]. Since the 1990s, many of the SI methods have been proposed which are inspired by various biological systems. For instance, ant colony optimization (ACO) [5] is inspired by the foraging behavior of an ant colony, which imitates the behavior of ants seeking the optimal path between their nest and a food source. Similarly, particle swarm optimization (PSO) [6] simulates the motion

of bird flocking when they are searching for food in a certain area. These SI methods were mainly proposed to optimize single-objective and multi-objective functions and solve NP-hard problems. However, during the last decade, SI methods have been successfully applied to solve data mining problems such as classification or clustering or to optimize the initial parameters of a classification algorithm such as support vector machine.

With the fast development of technologies such as the internet and smartphone, the amount of data that is collected or generated has been growing exponentially. The complexity, variety, and volume of this data is now contributing to the big challenges that face the scientific community. In recent decades, many big data frameworks have been proposed to cope with big data and its characteristics. Apache Spark is one of these frameworks, which was originally implemented by a team of researchers from the University of California-Berkeley in 2009 [7], to overcome the limitations of Hadoop MapReduce [8], in particular for processing iterative and interactive jobs. Apache Spark is an in-memory computing framework for processing big data using a cluster of nodes. The Resilient Distributed Dataset (RDD) is a basic component in Apache Spark, which is an immutable collection of objects distributed across a cluster of nodes in a fault-tolerant manner. Two operations can be performed on RDD, a transformation and an action. During the transformation, a new RDD is created from an existing RDD using functions such as filter, reduce, and map. The action operation is triggered by an RDD to compute and return the result [9,10]. RDDs' operations are performed in a 'lazy' fashion, which means that the operation on the RDD is only performed when one of the operations is called. This effectively avoids repeated evaluations. Furthermore, fault-tolerance of Spark applications is achieved by keeping track of the "lineage" of each RDD, which represent the sequence of operations that produced the result. This allows the reconstruction of the operations and data flow in the case of data loss.

Compared to the Hadoop MapReduce framework, Apache Spark is more efficient and approximately 10 to 100 times faster for certain data processing task on clusters of nodes. This is because a MapReduce jobs need I/O disk operations to shuffle and sort the data during the Map and Reduce phases. Furthermore, Apache Spark provides a rich APIs in several languages (Java, Scala, Python and R) for developers to choose from in order to perform complex operations on distributed RDDs [11,12].

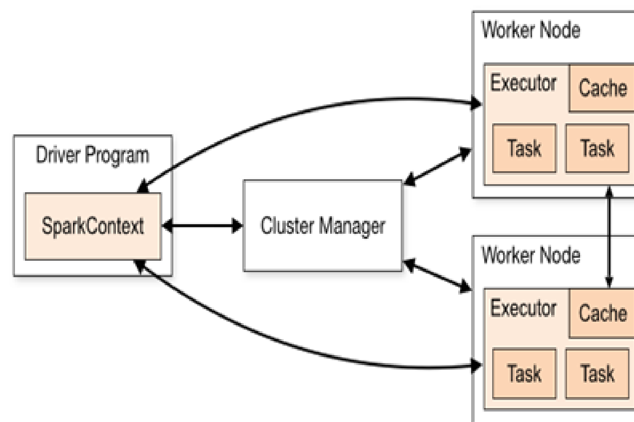


FIGURE 1 Spark Architecture [9]

As shown in Fig. 1, the Apache Spark architecture consists of two layers and a cluster manager. The spark application starts by creating one driver program in the master node and launches a spark context object within it. The driver program creates a logical plan for the RDD operations as a directed acyclic graph (DAG). The aim of the DAG is the recovery of the RDDs in case of a worker node failure or any RDD loss during the running of the application. Next, the DAG is converted using pipelining transformations to a physical plan, which is a set of stages, where each stage comprises of a set of tasks. After the physical plan is created, the driver program asks the cluster manager through the spark context to allocate the resources (Memory, CPU) and run executors on the worker nodes. At this stage, the cluster manager registers the executors to the driver program. During the running of the spark application, the driver program monitors the executors and sends the tasks to the executors to run in multi-thread mode. The spark application keeps running until the spark context's stop method is invoked or the main function of the application is finished [9,10,11,12].

Although Apache Spark does not support a shared global memory between a driver program and the tasks that are running on the worker nodes. It provides two types of shared variables, Broadcast and Accumulator. The broadcast variable is a read-only variable that is created by the driver program and is cached on the memory of executors, which is used by the executors during the task execution. The Accumulator variable is a write-only variable, which is used to aggregate values from the executors in the driver program [11,12].

In [13], a PSO classification algorithm was proposed by Falco et al. to solve classification problem. The main idea of the PSO classification algorithm is to find the optimal centroid for each target label in a data set using the basic PSO algorithm and then assign each data instance in a testing data set to the closest centroid. Because of inefficient performance of PSO classification algorithms to handle big data sets, in this paper, we proposed a parallel version for the PSO classification algorithm using the Apache Spark framework, which is further referred to as SCPSO. In addition, we investigated the scalability and the performance of SCPSO.

To the best of our knowledge, this is the first work that implements the PSO classification algorithm using the Apache Spark framework. The aim is to shows how the PSO classification takes advantage of Apache Spark to work on very large data sets to achieve high accuracy and scalability.

The remaining of this paper is organized as follows. In Section 2, we present the related work of parallel data mining algorithms that are based on big data frameworks. Section 3 describes the basic particle swarm optimization algorithm. In Section 4, we introduce our proposed approach, SCPSO. Section 5 presents the experimental evaluation as well as the results. Finally, Section 6 presents our conclusions.

2 | RELATED WORK

A scalable design and implementation of traditional data mining algorithms has recently received attention by researchers to overcome the inefficiency of traditional algorithms for managing and analyzing big data. During the last decade, many of the big data frameworks have been proposed, such as MapReduce and Spark, to capture the characteristics of big data which are Volume, Velocity, Veracity and Variety [14]. In this section, we will present the related work of parallel data mining algorithms that are based on big data frameworks.

In [15], the authors proposed a parallelized version of the K-means clustering algorithm using the MapReduce framework. In their proposed approach, the K-means algorithm is formulated as a MapReduce job to find cluster centroid points. In the MapReduce job, the Map function assigns each data point to the closest centroid, while the Reduce function updates the centroids. The MapReduce job is repeated until the stop condition is met. The experimental results revealed that the algorithm can process the large data efficiently on a cluster of nodes.

The parallelization and scalability of a common and effective fuzzy clustering algorithm named Fuzzy C-Means (FCM) algorithm was proposed in [16]. The algorithm was parallelized using the MapReduce paradigm. A validity analysis was conducted in order to show that the implementation works correctly achieving competitive purity results compared to state-of-the art clustering algorithms. Furthermore, a scalability analysis was conducted to demonstrate the performance of the parallel FCM implementation for increasing number of computing nodes used.

In [17], the authors proposed a parallel K-means clustering algorithm based on the Apache Spark framework to overcome the limitations of the K-means algorithm that is provided by the Spark MLIB library. The main work of the algorithm is to perform the distance computation between data points and the centroids on the worker nodes, and the centroid update on the master node. The algorithm was tested with real data sets and the results showed the effectiveness and efficiency of the parallel K-mean algorithm.

Yang and Li in [18] proposed a parallel ant-colony clustering algorithm using the MapReduce framework. In their work, heterogeneous big data is automatically decomposed into clusters using the ant colony algorithm. The algorithm was tested with big data sets and the experimental results demonstrate that the algorithm achieved good accuracy with good efficiency.

In [19], the authors proposed a parallelized version of the particle swarm optimization clustering algorithm using the MapReduce framework (MR-CPSO). In MR-CPSO, the data points are divided into clusters by taking the minimum distances between data points and the cluster centroids. The updating of the particle centroids and the fitness function evaluation in MR-CPSO are carried out through three modules. In the first module, the particle centroids are updated using Map and Reduce functions. The second module uses the result of the first module, which are the particle centroids, to evaluate the fitness value of each particle using Map and Reduce functions. In the third module, the results from the first and second modules are merged to form a single new swarm for the next iteration. In addition to that, the Best Global and Best Local positions are updated. The scalability of the algorithm was evaluated with synthetic data sets and the results showed the effectiveness and efficiency of the algorithm to process big data sets.

In [20], the author used a PSO clustering algorithm running on the MapReduce platform to cluster streaming twitter data. Twitter data was pre-processed through three phases; tokenizing, stemming, and filtering by removing stop words, before applying the PSO clustering algorithm. The experimental results showed that the parallel PSO algorithm outperformed K-means in terms of the F-measure, and the parallel PSO scaled very well when increasing the dimensionality and size of the data. Another work is reported in [21] whereby the authors proposed an intrusion detection model using the MapReduce-based PSO clustering algorithm to analyze the network traffic. The experiments were conducted with a real traffic network data set and the experimental results revealed that the model achieved good detection rate with a very low rate of false alarms. In addition, the speedup of the model is very close to the linear speedup especially for large data sets.

A new MapReduce-based artificial bee colony clustering algorithm is proposed in [22]. The artificial bee colony (ABC) method was introduced to imitate the foraging behavior of honey bees. In ABC, the colony consists of the employee and onlooker bees that cooperate to find the best food source. ABC is used to solve the data clustering problem by finding the optimal centroids when minimizing the sum of distances between data points

and the cluster centroids. The authors proposed a model for ABC clustering using MapReduce in order to handle large data sets. In this model, the two main operations, the updating of centroids and the fitness evaluation, are adapted as a MapReduce job. In the MapReduce job, the Map function computes the distance between the data point and the centroids, then generate a new key-value pair whereby the value is the minimum distance. After that, the Map function emits the key-value pairs to the Reduce function. The Reduce function computes the average distance after aggregating the values with the same key. The experiments were performed on real and synthetic data sets in order to measure the performance and scalability of ABC clustering. The results revealed the effectiveness and efficiency of the MapReduce-based ABC clustering algorithm.

The grey wolf optimizer (GWO) is one of the most recent swarm intelligence methods that simulates the hunting behavior of grey wolves. In [23], the authors proposed a new algorithm for data clustering using enhanced GWO and MapReduce called MR-EGWO. The MR-EGWO was tested with real and synthetic data sets and compared with four clustering algorithms that are based on MapReduce; parallel K-Means, parallel K-PSO, MapReduce based artificial bee colony optimization (MR-ABC), and the dynamic frequency based parallel k-bat algorithm (DFBPKBA). The results showed that MR-EGWO outperformed the four algorithms in terms of the F-measure, and it also achieved significant speedup performance for large data sets.

3 | BASIC PARTICLE SWARM OPTIMIZATION ALGORITHM

The PSO algorithm was introduced by Kennedy and Eberhart in 1995 [6], which is a stochastic search method that was inspired by the social behavior of a flock of birds when they are looking for the best source of food in a certain area. In PSO, each particle represents a potential solution within a solution space, which moves through a search space seeking the best solution. The direction of a particle's movement is affected by three factors:

- Inertia Weight (w), which controls how much the particle follows its current position.
- Cognitive Learning (c_1), which controls how much the particle follows the best position that it has found so far.
- Social Learning (c_2), which controls how much the particle follows the best position that was found by the entire swarm.

Algorithm 1 shows the pseudo code of the PSO algorithm [13,24]. The PSO algorithm starts with a predefined number of particles, where each particle's position vector $p = \{p_1, p_2, p_3, \ldots, p_n\}$ is initialized randomly in an N-dimensional search space with a random velocity vector $v = \{v_1, v_2, v_3, \ldots, v_n\}$. At each iteration of PSO, a fitness function computes the fitness for each particle by evaluating the particle's position to determine how close this particle is to achieve the goal (best solution). Based on the fitness, the personal best position ($PBest$) and the global best position ($GBest$) are updated. $PBest$ is the best position that has been discovered so far by a particle, and $GBest$ is the best position that has been discovered so far by any particle of the entire swarm. Then, the current velocity vector of each particle is updated using $PBest$ and $GBest$ as follows:

$$\begin{aligned} v_j^{(t+1)} &= w v_j^t + r_1 c_1 \left(PBest - p_j^t\right) \\ &+ r_2 c_2 \left(GBest - p_j^t\right) \end{aligned} \tag{1}$$

here, p_j^t and v_j^t are the current position vector and the current velocity vector of particle j at iteration t, respectively; r_1 and r_2 are the random vectors, c_1 (cognitive learning) and c_2 (social learning) are constant coefficients specified by the user. In our work, the inertia weight value w is linearly decrementing with increasing numbers of iterations, which is calculated using Eq. 2 at each iteration t [25]. v_j^{t+1} is a new velocity vector of particle j, where the new value at each dimension of the velocity vector is clamped within the range $[v_{min}, v_{max}]$.

$$w(t) = w_{max} - \left((w_{max} - w_{min})\frac{t}{T_{max}}\right) \tag{2}$$

Then, the new position of the particle j ($p_j^{(t+1)}$) is computed using the current particle's position (p_j^t) and the new velocity vector ($v_j^{(t+1)}$) as follows:

$$p_j^{(t+1)} = p_j^t + v_j^{(t+1)} \tag{3}$$

All previous operations are repeated until the stopping criterion is satisfied [24].

Algorithm 1 PSO Algorithm

```
for each particle do
    Randomly initialize particle's position and velocity
end for
repeat
    for each particle do
        Compute fitness (FV)
        if Current fitness is better than the best fitness in particle's memory then
            Take current particle position as personal best position (PBest)
        end if
    end for
    Take position of particle whose best fitness as global best position (GBest)
    for each particle do
        Update particle velocity using Eq. (1)
        Update particle position using Eq. (3)
    end for
    Update the inertia weight using on Eq. (2)
until stopping criterion is satisfied
```

4 | PROPOSED APPROACH

The SCPSO algorithm is based on PSO to find the optimal centroid for all target classes in a data set. In SCPSO, each particle's position and velocity are encoded as vectors as follows:

$$\overrightarrow{p_j} = \{p_j^{c1}, p_j^{c2}, \ldots, p_j^{c_i}\} \tag{4}$$

$$\overrightarrow{v_j} = \{v_j^{c1}, v_j^{c2}, \ldots, v_j^{c_i}\} \tag{5}$$

where $p_j^{c_i}$ and $v_j^{c_i}$ are the position and the velocity vector for particle j 's class label c_i, respectively, which are represented in an N-dimensional space as follows:

$$p_j^{c_i} = \{p_1, p_2, \ldots, p_n\} \tag{6}$$

$$v_j^{c_i} = \{v_1, v_2, \ldots, v_n\} \tag{7}$$

where p_n and v_n are the real values of the position and velocity, respectively, for dimension n.

In addition, each particle has the following attributes:

- Particle Identification Number (ParticleID)
- Current Fitness (FV)
- Best Centroids Vector that has been discovered so far during the journey of particle (PBest)
- Best fitness that has been discovered so far during the journey of particle

Furthermore, SCPSO keeps track of the best centroids vector (GBest) and the best group fitness which are achieved by any particle.

To evaluate the fitness of each particle, two fitness functions are used. The first fitness function (F1) computes the fitness of particle j using Eq. 8 by taking the average of the sum of all Euclidean distances (Eq. 9) between the current centroid vector of class label c_i ($\overrightarrow{p_j}^{c_i}$) and the data instances that belong to class label c_i according to the training data set [13].

$$F1(j) = \frac{1}{D}\sum_{i=1}^{D} d\left(\overrightarrow{x}_i, \overrightarrow{p_j}^{c_i}\right) \tag{8}$$

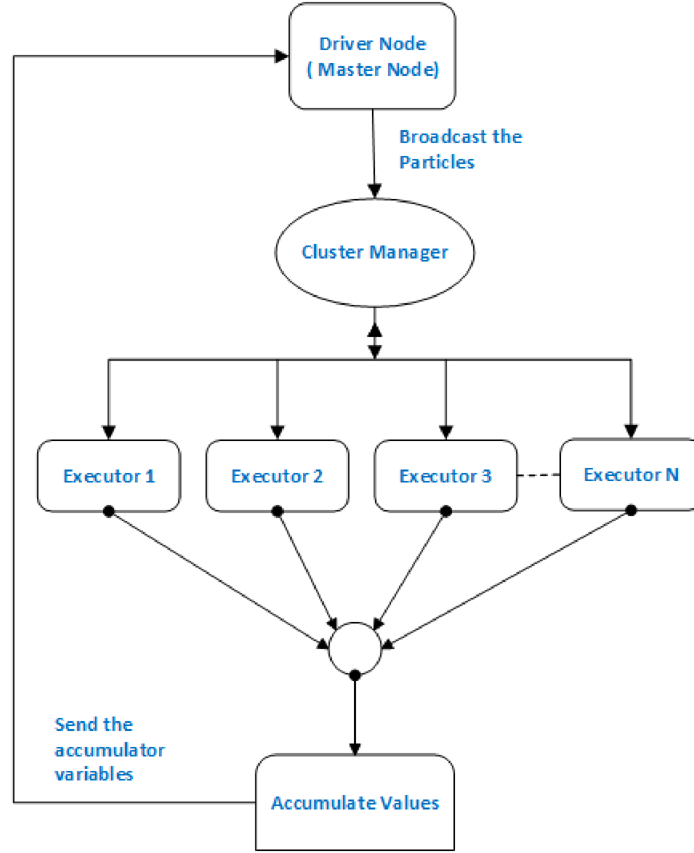


FIGURE 2 SCPSO Architecture

where D is the number of data instances in a training data set, and $\vec{x}_i$ is the data instance in the training data set. It should be noted here that each data instance for each dimension (feature) is normalized within [0.0,1.0] using min-max normalization, and that the sum of those distances is divided by N (total number of features in data set), and thus the computed distance is also within the range of [0.0,1.0] [13].

$$d\left(\vec{a}, \vec{b}\right) = \sqrt{\sum_{k=1}^{n} (a_k - b_k)^2} \quad (9)$$

For the second fitness function F2, the fitness of particle j is computed as follows, which is a linear combination of two fitness values [13]:

$$F2(j) = \frac{1}{2}(F1(j) + F\psi(j)) \quad (10)$$

The first fitness is computed using the first fitness function F1 (Eq. 8) and the second fitness is computed using the $F\psi$ fitness function. $F\psi$ computes the fitness in two stages. In the first stage, all data instances in a training data set are assigned to the closest centroid. The second stage calculates the percentage of incorrectly classified instances of a training data set [13]:

$$F\psi\,(j) = \frac{1}{D}\sum_{i=1}^{D} \delta(\vec{x}_i) \quad (11)$$

$$\delta(\vec{x}_i) = \begin{cases} 1 & \text{if} \quad c_{predicted}(\vec{x}_i) \neq c_{actual}(\vec{x}_i) \\ 0 & \text{otherwise} \end{cases} \quad (12)$$

where D is the number of data instances in a training data set, c_{actual} is the actual outcome of $\vec{x}_i$, and $c_{predicted}$ is the predicted outcome of $\vec{x}_i$.

The particle position updating and fitness evaluation at each iteration are needed to be adapted in order to apply the classification task on a large data set. In SCPSO, these two operations are performed in two stages, as shown in Figure 2. In the first stage, the driver program sends the tasks to the executors. Besides, all particles are sent to the executors via a broadcast variable through a cluster manager. Then, each executor reads a portion of data records that is encapsulated in an RDD. The pseudo code of each executor for fitness function F1 and F2 evaluation are

shown in Algorithm 2 and 3, respectively. It should be noted here that the executors read a portion of data instances only once and cache these data instances in their memory for the next iterations.

In Algorithm 2, the executor extracts the data instance vector x from RDD and the position vector p that belongs to the same class label of x from the particle. Then, the distance between p and x is calculated using Eq. 9. After that, ParticleID and the calculated distance are added to the accumulator variable $\mathsf{Accumulator}_{F1}$. This process is repeated over all data instances in the RDD.

Algorithm 2 Fitness Function 1 Evaluation

```
for each data instance x in RDD do
    for each particle j in Swarm do
        - Extract centroid vector p from particle j which belongs to the same class label of x
        - Compute the distance between x and p using Eq. (9)
        - Add ParticleID and the distance to accumulator Accumulator_F1
    end for
end for
```

Algorithm 3 Fitness Function 2 Evaluation

```
for each data instance x in RDD do
    for each particle j in Swarm do
        // FV1
        - Extract centroid vector p from particle j which belongs to same class label of x
        - Compute the distance between x and p using Eq. (9)
        - Add ParticleID and distance to accumulator Accumulator_F1

        // FVψ
        - Assign x to closest centroid
        if Assigned_Class(x) != Actual_Class(x) then
            - Add 1.0 and ParticleID to accumulator Accumulator_F2
        end if
    end for
end for
```

In Algorithm 3, the executor extracts data instance vector x from the RDD and computes the fitness in two steps. The first step is the same as the previous one and the result is added to the accumulator variable $\mathsf{Accumulator}_{F1}$. In the second step, the data instance x is assigned to the class label whose position is the closest to that data instance and then the predicted class label is compared with the actual class label of x. If the predicted class label is not the same as the actual class label, then the value is 1.0, and ParticleID are added to the accumulator variable $\mathsf{Accumulator}_{F2}$. This process is repeated over all data instances in the RDD.

After the executors finish their work, the second stage starts by sending the accumulator variables $\mathsf{Accumulator}_{F1}$ or/and $\mathsf{Accumulator}_{F2}$ to the driver program to update the fitness FV and the position vector p of the particles. The portion of pseudo code of the driver program is shown in Algorithm 4.

The previous operations are repeated until the maximum number of iterations is reached. Then, each data instance in a testing data set is assigned to the class label whose centroid is the closest using Eq. 9.

5 | EXPERIMENTS AND RESULTS

In this section, we will start by presenting the data sets that are used in our experiments, then describe the execution environment of our experiments. Finally, describe the experiments and the results. We will focus on the scalability measurements; speedup and scaleup for the SCPSO evaluation.

Algorithm 4 Driver Program

```
Find number of instances in training data set D        ▷ Is executed only once during the application run
for each particle j in Swarm do
    if Fitness_Function = F1 then
        - Extract a value V of particle j from Accumulator_F1
        - Update FV of the particle j using Eq. (8)
        - Update PBest if necessary
    else if Fitness_Function = F2 then
        - Extract a value V1 of particle j from Accumulator_F1
        - Extract a value V2 of particle j from Accumulator_F2
        - Update FV of particle j using Eq. (10)
        - Update PBest if necessary
    end if
end for
- Take position vector of particle whose best FV value as GBest
for each particle j in Swarm do
    - Update particle velocity using Eq. (1)
    - Update particle position using Eq. (3)
end for
```

TABLE 1 Properties of Data Sets

Data set	#Instances	#Features	#Training Instances	#Testing Instances	File Size (MB)	#Class labels
HIGGS	10,809,619	28	8,620,812	2,188,807	5,208.878	2
CICIDS2017 [27]	1,223,895	70	973,650	250,245	923.898	2
Skin	245,057	3	196,000	49,057	14.092	2
NSL-KDD [28]	140,491	37	112,394	28,097	31.411	2
Electricity	45,312	8	36,250	9,062	3.348	2

5.1 | Data Sets

In our experiments, we used real-world data sets to evaluate the scalability, performance, and robustness of our proposed SCPSO algorithm, which were taken from the UCI Machine Learning Repository[†], Canadian Institute for Cybersecurity data sets[‡], and MOA Machine Learning for Streams[§]. Table 1 shows the data sets and their properties. All data sets in the table were normalized using min-max normalization (Apache Spark MinMaxScaler)[26] to rescale the features to the range [0,1]. In addition, all data instances which have missing values were removed from the data sets. Furthermore, we split the data set into two data sets whereby 80% of the instances were used for training and the remaining for testing.

5.2 | Environment

We ran our experiments on the SDSC Dell Cluster with Intel Haswell Processors (COMET) operated by the San Diego Supercomputer Center at UC San Diego[¶]. The SDSC-Comet cluster consists of 1,948 nodes where each node has 24 Intel Xeon cores (2.5 GHz speed) and 128GB of DRAM. For the Apache Spark environment, we used Spark version 2.1, standalone cluster manager, and Java Runtime 1.8 to implement the SCPSO algorithm.

The parameter values of the SCPSO algorithm except for the maximum number of iterations are as in[13]:

- maximum number of iterations = 300
- acceleration coefficient constants c_1 and c_2 = 2.0

[†] http://archive.ics.uci.edu/ml
[‡] http://www.unb.ca/cic/datasets/index.html
[§] https://moa.cms.waikato.ac.nz/datasets/
[¶] https://portal.xsede.org/sdsc-comet

TABLE 2 Accuracy

Data set	SCPSO-F1	SCPSO-F2
HIGGS	52.99	**61.16**
CICIDS2017	80.00	**92.69**
Skin	91.01	**94.35**
NSL-KDD	86.86	**90.02**
Electricty	53.37	**75.89**

- swarm size = 50
- velocity range [v_{min} = -0.05, v_{max} = 0.05]
- weight inertia range [w_{min}= 0.4, w_{max}= 0.9]

5.3 | Evaluation Measures

To evaluate the scalability, performance, and robustness of SCPSO, we used the speedup [29] and scaleup [29] measures as well as classification accuracy.

The speedup measures the parallelization ability of the algorithm by taking the ratio of the running time on a single node to the running time on parallel nodes n. The speedup is calculated as follows, where the data set size is fixed while the number of nodes is increased by a certain ratio [19,29]:

$$Speedup = \frac{T_1}{T_n} \tag{13}$$

where T_1 is the running time using a single node, and T_n is the running time using n nodes.

On the other hand, scaleup measures how the cluster of nodes are utilized efficiently by the parallel algorithm. The scaleup is calculated as follows, where the data set size and the number of nodes are increased by the same ratio [19,29]:

$$Scaleup = \frac{T_{sn}}{T_{Rsn}} \tag{14}$$

where T_{sn} is the running time for the data set with size s using n nodes, and T_{Rsn} is the running time for the data set with size Rs using Rn nodes.

For the robustness evaluation of a model, we used the classification accuracy measure, which is calculated as follows:

$$Accuracy = (\frac{\#InstancesCorrectlyClassified}{\#Instances}) \times 100 \tag{15}$$

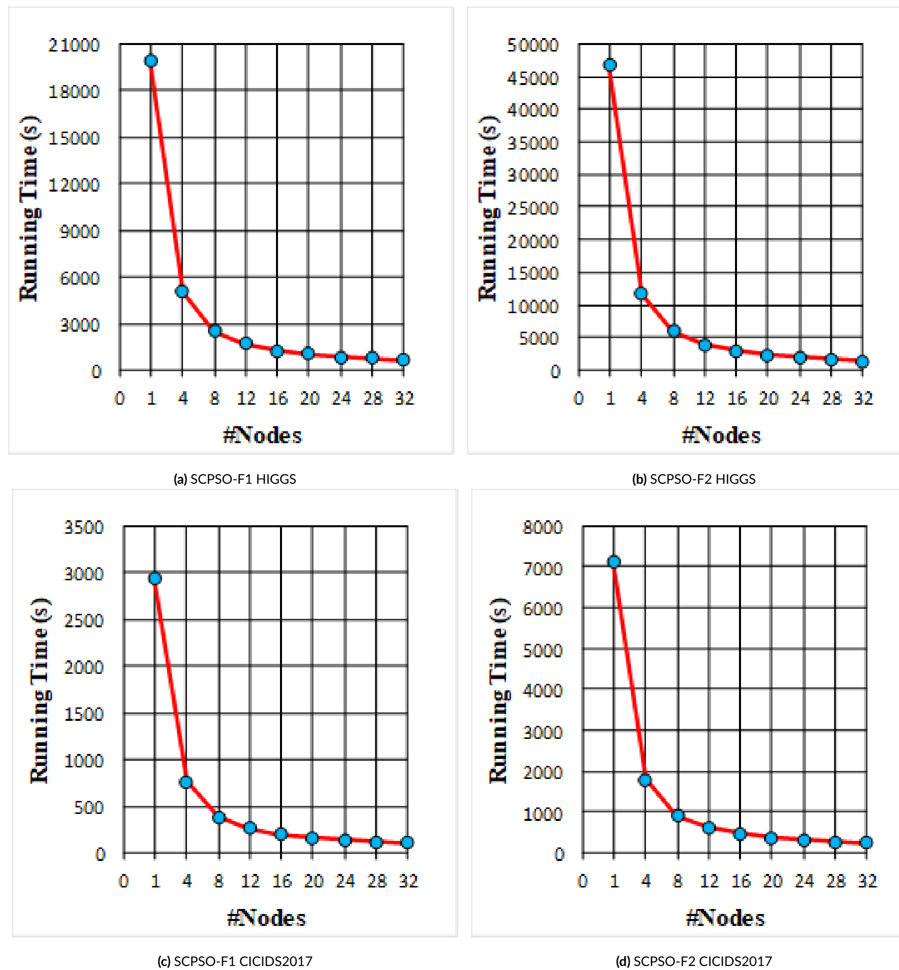
5.4 | Results

To evaluate the robustness of the SCPSO algorithm, we ran SCPSO using two fitness functions F1 and F2 on all data sets given in Table 1. Table 2 shows the accuracy that was achieved by SCPSO-F1 and SCPSO-F2. From the results, we can see that the performance of SCPSO-F2 outperformed the performance of SCPSO-F1 for all data sets in terms of accuracy, where SCPSO-F2 obtained 61.16%, 92.69%, 94.35%, 90.02% and 75.89% compared to the accuracy values obtained by SCPSO-F1, which were 52.99%,80.00%, 91.01%, 86.86% and 53.37%, for HIGGS, CICIDS2017, Skin, NSL-KDD, and Electricity, respectively.

For the scalability evaluation of the SCPSO algorithm, we used the data sets which have more than 1,000,000 instances. Based on the data sets in Table 1, HIGGS and CICIDS2017 contain 10,809,619 and 1,223,895, respectively. We performed two experiments using the HIGGS and CICIDS2017 data sets to evaluate the speedup and scaleup of the SCPSO algorithm.

In the first experiment, we ran the SCPSO algorithm using two fitness functions F1 and F2 on the HIGGS and CICIDS2017 data sets on the COMET cluster. In each run, the number of nodes is increased by a factor of 4. In addition, we reported the running time in seconds and speedup of the SCPSO algorithm for the HIGGS and CICIDS2017 data sets as shown in Figures 3 and 4.

From Figure 3, we can easily see that the running time measured in seconds (s) of SCPSO-F1 is better than the running time of SCPSO-F2 on both, the HIGGS and CICIDS2017 data set, over all nodes. The reason for this is because fitness function F2 computes the fitness in two steps and thus takes longer. In Figures 3a and 3c the running time of SCPSO-F1 for the HIGGS and CICIDS2017 data sets using 32 nodes were 657s and 113s, respectively, compared to the running time using a single node which were 19,975s and 2,946s for HIGGS and CICIDS2017, respectively. For Figures 3b and 3d, the running time of SCPSO-F2 for the HIGGS and CICIDS2017 data sets using a single node were 46,703s and 7,136s,



(a) SCPSO-F1 HIGGS

(b) SCPSO-F2 HIGGS

(c) SCPSO-F1 CICIDS2017

(d) SCPSO-F2 CICIDS2017

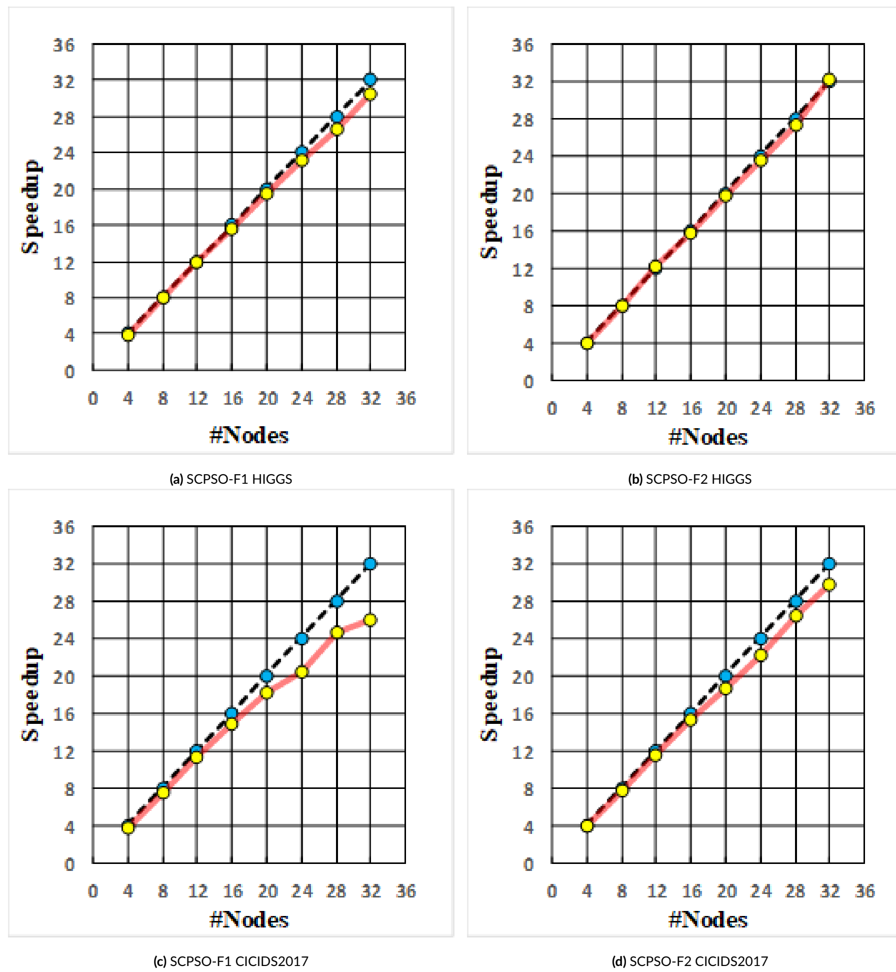
FIGURE 3 SCPSO-F1 and SCPSO-F2 Running Times

respectively, which decreased to 1,453s for HIGGS and 240s for CICIDS2017 using 32 nodes. A significant conclusion that we can draw from Figures 3a-3d is that the running time of SCPSO-F1 and SCPSO-F2 decreases approximately linearly when the number of nodes are increased.

On other hand as shown in Figure 4, we can note that the speedup of SCPSO-F2 is better than the speedup of SCPSO-F1 on the HIGGS and CICIDS2017 data sets. For example, using 32 nodes, the speedup of SCPSO-F2 for HIGGS and CICIDS2017 were 32.14 and 29.73, respectively, compared to the speedup of SCPSO-F1, which were 30.4 and 26.07 for HIGGS and CICIDS2017, respectively. This is because F2 is more complex, thus, the utilization using the Spark framework is higher compared to the less complex F1 function. This means that the parallelization is more effective on F2. In Figure 4a, the speedup results of SCPSO-F1 achieved with 4 to 20 nodes using the HIGGS data set were approximately the linear speedup, then the speedup drifts a little away from the linear speedup. In Figure 4c, SCPSO-F1 achieved linear speedup results with 4 to 12 nodes using the CICIDS2017 data set, then the speedup results drift away from the linear speedup. In Figure 4b and 4d, SCPSO-F2 achieved approximately linear speedup results using the HIGGS data set. For the CICIDS2017 data set, the speedup results of SCPSO-F2 using 4 to 16 nodes were approximately similar to the linear speedup, then the speedup drifts a little away starting from 20 nodes.

Finally, we can conclude from Figures 4a-4d that both versions of SCPSO achieved a significant speedup where the speedup of SCPSO-F2 is almost identical to the linear speedup, and the speedup of SCPSO-F1 is quite close to the linear speedup.

For the second experiment, we measured the scaleup of SCPSO-F1 and SCPSO-F2 to see how the SCPSO-F1 and SCPSO-F2 variants utilize the cluster of nodes efficiently, as shown in Figure 5. We can note from Figures 5a-5d that the scaleup of SCPSO-F1 has almost a constant ratio ranging between 0.923 and 1.0 for HIGGS, and between 0.981 and 1.0 for CICIDS2017. The scaleup of SCPSO-F2 is a little better than the scaleup of SCPSO-F1. The scaleup of SCPSO-F2 has almost a constant ratio within a range of 0.976 to 1.011 for HIGGS, and a range of 0.998 to 1.018 for CICIDS2017. Overall, the SCPSO algorithm becomes more scalable as the data set size increases achieving a significant speedup.



(a) SCPSO-F1 HIGGS

(b) SCPSO-F2 HIGGS

(c) SCPSO-F1 CICIDS2017

(d) SCPSO-F2 CICIDS2017

FIGURE 4 SCPSO-F1 and SCPSO-F2 Speedup; black dashed line represents linear speedup and orange straight line represents speedup of SCPSO.

6 | CONCLUSION

In this paper, we designed and implemented a parallel particle swarm optimization classification algorithm (SCPSO) using the Apache Spark framework to overcome the inefficiency of PSO classification when handling big data sets. The SCPSO algorithm uses the basic PSO algorithm to find the optimal centroid for each target label in a data set, and then assigns each unlabeled data instance in a testing data set to the closest centroid. Two variants of SCPSO, SCPSO-F1 and SCPSO-F2, have been proposed based on a different fitness function. SCPSO-F1 and SCPSO-F2 were tested with real data sets to evaluate their scalability and robustness.

The scalability analysis for SCPSO-F2 revealed that the speedup of SCPSO-F2 is almost identical to the linear speedup, and SCPSO-F2 scales very well with increasing data set sizes. For SCPSO-F1, the scalability analysis showed that the running time of SCPSO-F1 is better than the running time of SCPSO-F2, and the speedup of SCPSO-F1 is very close to the linear speedup. In terms of accuracy of the predictive model, the performance of SCPSO-F2 outperformed the performance of SCPSO-F1 on all tested data sets.

Overall, the experimental results showed that SCPSO-F1 and SCPSO-F2 can be efficiently parallelized with the Apache Spark framework running on a cluster of nodes and the performance of SCPSO-F1 and SCPSO-F2 almost follow the ideal speedup with increasing data set sizes.

Our future work aims to apply the SCPSO algorithm on other real-world applications with very large data sets (terabyte size) using hundreds of nodes and conduct comprehensive experiments to investigate the utilization of the resources used.

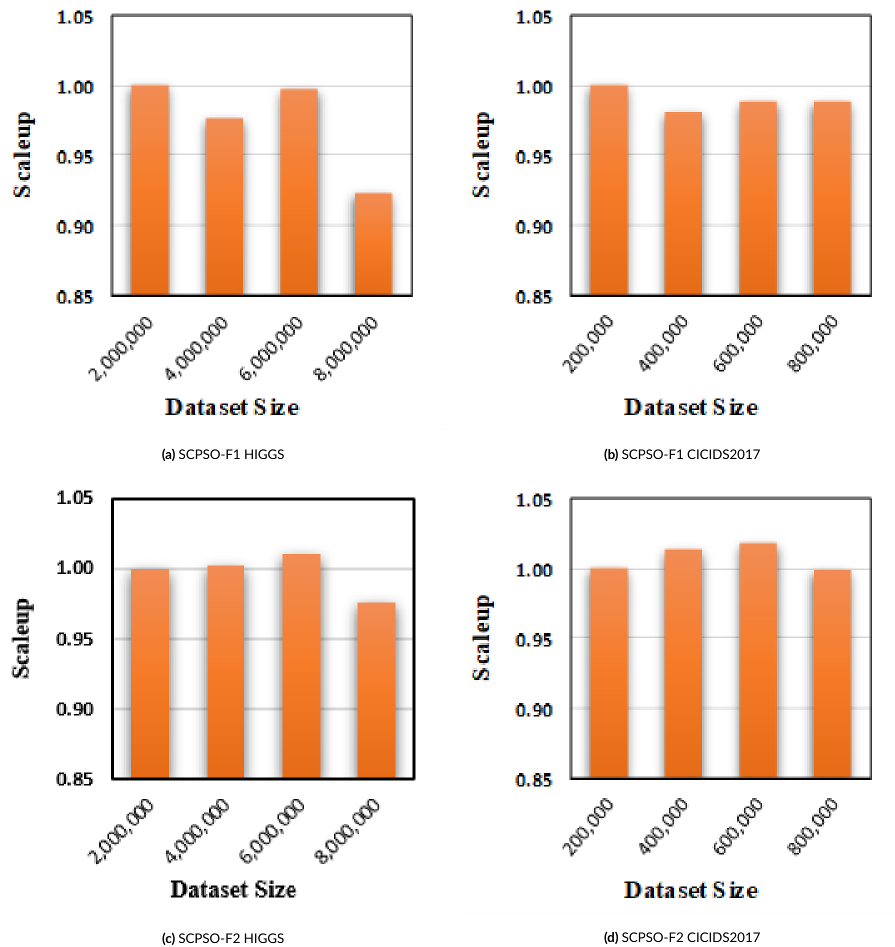


FIGURE 5 SCPSO-F1 and SCPSO-F2 Scaleup

ACKNOWLEDGMENT

This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number ACI-1548562.

References

1. Sayad S. Real Time Data Mining. Cambridge, Ontario: Self-Help Publishers; 2011.
2. Zaki MJ, Meira W. Data Mining and Analysis: Fundamental Concepts and Algorithms. New York: Cambridge University Press; 2017.
3. Coello CAC, Dehuri S, Ghosh S. Swarm Intelligence for Multi-Objective Problems in Data Mining. Berlin, Heidelberg: Springer Berlin Heidelberg; 2009.
4. Parsopoulos KE, Vrahatis MN. Particle Swarm Optimization and Intelligence: Advances and Applications. Hershey, PA: Information Science Reference; 2010.
5. Dorigo M, Stutzle T. Ant Colony Optimization. Cambridge, MA: MIT Press; 2004.
6. Eberhart R, Kennedy J. Particle swarm optimization. Proceedings of the IEEE international conference on neural networks. 1995.

7. Zaharia M, Chowdhury M, Das T, Dave A, Ma J, McCauley M, Franklin MJ, Shenker S, Stoica I. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation. 2012.

8. MapReduce Tutorial. https://hadoop.apache.org/docs/r1.2.1/index.html. Accessed June 14, 2018.

9. Spark Overview. Overview - Spark 2.1.0 Documentation. https://spark.apache.org/docs/2.1.0/. Accessed June 14, 2018

10. Karau H, Konwinski A, Wendell P, Zaharia M. Learning Spark: Lightning-Fast Big Data Analytics. Beijing: OReilly; 2015.

11. Salloum S, Dautov R, Chen X, Peng PX, Huang JZ. Big data analytics on Apache Spark. International Journal of Data Science and Analytics. 2016.

12. Gulati S, Kumar S. Apache Spark 2.x for Java Developers: Explore Data at Scale Using the Java APIs of Apache Spark 2.x. Birmingham: Packt Publishing; 2017.

13. Falco ID, Cioppa AD, Tarantino E. Facing classification problems with Particle Swarm Optimization. Applied Soft Computing. 2007;7(3):652-658. doi:10.1016/j.asoc.2005.09.004

14. Inoubli W, Aridhi S, Mezni H, Maddouri M, Nguifo EM. An experimental survey on big data frameworks. Future Generation Computer Systems. 2018;86:546-564. doi:10.1016/j.future.2018.04.032

15. Zhao W, Ma H, He Q. Parallel k-means clustering based on mapreduce. In IEEE International Conference on Cloud Computing. 2009.

16. Ludwig SA. MapReduce-based fuzzy c-means clustering algorithm: implementation and scalability. International Journal of Machine Learning and Cybernetics. 2015;6(6):923-934. doi:10.1007/s13042-015-0367-0

17. Wang B, Yin J, Hua Q, Wu Z, Cao J. Parallelizing K-Means-Based Clustering on Spark. 2016 International Conference on Advanced Cloud and Big Data (CBD). 2016. doi:10.1109/cbd.2016.016

18. Yang J, Li X. MapReduce Based Method for Big Data Semantic Clustering. 2013 IEEE International Conference on Systems, Man, and Cybernetics. 2013. doi:10.1109/smc.2013.480

19. Aljarah I, Ludwig SA. Parallel particle swarm optimization clustering algorithm based on MapReduce methodology. 2012 Fourth World Congress on Nature and Biologically Inspired Computing (NaBIC). 2012. doi:10.1109/nabic.2012.6402247

20. Chunne AP, Chandrasekhar U, Malhotra C. Real time clustering of tweets using adaptive PSO technique and MapReduce. 2015 Global Conference on Communication Technologies (GCCT). 2015. doi:10.1109/gcct.2015.7342704

21. Aljarah I, Ludwig SA. MapReduce intrusion detection system based on a particle swarm optimization clustering algorithm. 2013 IEEE Congress on Evolutionary Computation. 2013. doi:10.1109/cec.2013.6557670

22. Banharnsakun A. A MapReduce-based artificial bee colony for large-scale data clustering. Pattern Recognition Letters. 2017;93:78-84. doi:10.1016/j.patrec.2016.07.027

23. Tripathi AK, Sharma K, Bala M. A Novel Clustering Method Using Enhanced Grey Wolf Optimizer and MapReduce. Big Data Research. 2018;14:93-100. doi:10.1016/j.bdr.2018.05.002

24. Al-Sawwa J, Ludwig SA. Centroid-Based Particle Swarm Optimization Variant for Data Classification. 2018 IEEE Symposium Series on Computational Intelligence (SSCI). 2018. doi:10.1109/ssci.2018.8628926.

25. Shi Y, Eberhart R. A modified particle swarm optimizer. 1998 IEEE International Conference on Evolutionary Computation Proceedings IEEE World Congress on Computational Intelligence (Cat No98TH8360). doi:10.1109/icec.1998.699146

26. Extracting, transforming and selecting features. Extracting, transforming and selecting features - Spark 2.1.0 Documentation. https://spark.apache.org/docs/2.1.0/ml-features.html. Accessed June 14, 2018.

27. Sharafaldin I, Lashkari AH, Ghorbani AA. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. Proceedings of the 4th International Conference on Information Systems Security and Privacy. 2018. doi:10.5220/0006639801080116

28. Tavallaee M, Bagheri E, Lu W, Ghorbani AA. A detailed analysis of the KDD CUP 99 data set. 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications. 2009. doi:10.1109/cisda.2009.5356528

29. Grama A, Gupta A, Karypis G, Kumar V. Introduction to Parallel Computing, Second Edition. Addison-Wesley; 2003.