MAPREDUCE-ENABLED SCALABLE NATURE-INSPIRED APPROACHES FOR CLUSTERING

A Dissertation
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Ibrahim Mithgal Aljarah

In Partial Fulfillment of the Requirements
for the Degree of
DOCTOR OF PHILOSOPHY

Major Department:
Computer Science

September 2013

Fargo, North Dakota

North Dakota State University
Graduate School

Title

MAPREDUCE-ENABLED SCALABLE NATURE-INSPIRED APPROACHES FOR CLUSTERING

By

Ibrahim Mithgal Aljarah

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

DOCTOR OF PHILOSOPHY

SUPERVISORY COMMITTEE:

Dr. Simone Ludwig
Chair

Dr. Jun Kong

Dr. Changhui Yan

Prof. Dogan Comez

Approved:

2/21/2014	Prof. Brian M. Slator
Date	Department Chair

ABSTRACT

The increasing volume of data to be analyzed imposes new challenges to the data mining methodologies. Traditional data mining such as clustering methods do not scale well with larger data sizes and are computationally expensive in terms of memory and time.

Clustering large data sets has received attention in the last few years in several application areas such as document categorization, which is in urgent need of scalable approaches. Swarm intelligence algorithms have self-organizing features, which are used to share knowledge among swarm members to locate the best solution. These algorithms have been successfully applied to clustering, however, they suffer from the scalability issue when large data is involved. In order to satisfy these needs, new parallel scalable clustering methods need to be developed.

The MapReduce framework has become a popular model for parallelizing data-intensive applications due to its features such as fault-tolerance, scalability, and usability. However, the challenge is to formulate the tasks with map and reduce functions.

This dissertation firstly presents a scalable particle swarm optimization (MR-CPSO) clustering algorithm that is based on the MapReduce framework. Experimental results reveal that the proposed algorithm scales very well with increasing data set sizes while maintaining good clustering quality. Moreover, a parallel intrusion detection system using the MR-CPSO is introduced. This system has been tested on

a real large-scale intrusion data set to confirm its scalability and detection quality.

In addition, the MapReduce framework is utilized to implement a parallel glowworm swarm optimization (MR-GSO) algorithm to optimize difficult multimodal functions. The experiments demonstrate that MR-GSO can achieve high function peak capture rates.

Moreover, this dissertation presents a new clustering algorithm based on GSO (CGSO). CGSO takes into account the multimodal search capability to locate optimal centroids in order to enhance the clustering quality without the need to provide the number of clusters in advance. The experimental results demonstrate that CGSO outperforms other well-known clustering algorithms.

In addition, a MapReduce GSO clustering (MRCGSO) algorithm version is introduced to evaluate the algorithm's scalability with large scale data sets. MRCGSO achieves a good speedup and utilization when more computing nodes are used.

ACKNOWLEDGMENTS

First and foremost, I would like to express my gratitude and obligation to Allah (God) my loving creator who loves us to explore his creation and for providing me the blessings to write this disseration. You given me the power to believe in my passion and follow my dreams. I could never have done this work without the faith I have in you.

I would like to express my most appreciation to my adviser Professor Simone A. Ludwig, who always believed in me and never hesitate to provide endless support and motivation at all times. Professor Ludwig always inspired me and motivated me through my graduate study years, without her advices and efforts, I would have not been able to complete this dissertation. For everything you have done for me, Professor Ludwig, I thank you. I would like to thank the supervisory committee members, Prof. Jun Kong, Prof. Changhui Yan, and Prof. Dogan Comez for their interest in my dissertation, for their support, and for valuable and helpful suggestions they made.

I have to thank my beloved parents for their supplications, love, unconditional support throughout my life. Thank you for prodigious sacrifices that you made to ensure that I had an preferential education. For this and much more, I am forever in their debt. Special thanks to my brothers and my sister for their endless advises and support. To my beloved wife Nailah: What can I say? You are one of the main reasons for my success I am so thankful that I have you in my life pushing me when I am ready to give up. Thanks for not just support, but knowing that I could do this! I Love You Always and Forever!. To my beloved daughter Nada, when I look in her eyes, I derive hope to work for a bright future, I would like to express my love for being such a good girl always cheering me up. Special thanks to all my friends for their encouragement and motivation that helped me reach here.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

CHAPTER 1. INTRODUCTION

Managing scientific data has been identified as one of the most important emerging needs of the scientific community in recent years. This is because of the sheer volume and increasing complexity of data being created or collected. In particular, in the growing field of computational science where increases in computer performance allow ever more realistic simulations and the potential to automatically explore large parameter spaces. As noted by Bell et al. [1]: "As simulations and experiments yield ever more data, a fourth paradigm is emerging, consisting of the techniques and technologies needed to perform data intensive science".

The question to address is how to effectively generate, manage and analyze the data and the resulting information. The solution requires a comprehensive, end-to-end approach that encompasses all the stages from the initial data acquisition to its final analysis. This chapter briefly describes the background to the research topics investigated in this dissertation, the motivation of the work, contributions and the structure of the dissertation.

1.1. Background

Data mining [2, 3] is a process of discovering interesting patterns in large data sets by transforming it into useful information. There are many data mining techniques that have been introduced in the literature such as association rule mining, data classification, and data clustering. Data clustering [3] is a widely studied data mining and most important unsupervised learning technique used when analyzing data. Data clustering is a technique to understand and convert data streams into beneficial information, and is increasingly being used in different applications, for instance, pattern recognition [4], document categorization [5], social networking [6], and bioinformatics applications [7].

Advances in parallel computing environments lead to the development of parallel

data mining algorithms making them able to cope with the exponentially increasing sizes of data sets. Furthermore, clustering very large data sets that contain large numbers of instances with high dimensions is considered a very important issue nowadays. Most sequential clustering algorithms suffer from the problem that they do not scale with larger data set sizes, and most of them are computationally expensive in memory space and time complexities. For these reasons, the parallelization of data clustering algorithms is paramount in order to deal with large scale data. To develop a good parallel clustering algorithm that takes big data into consideration, the algorithm should be efficient, scalable and obtain high quality clusters.

In general, developing traditional parallel algorithms using the Message Passing Interface (MPI) methodology [8] faces a wide range of difficulties such as handling the network communication in an efficient manner and balancing the distribution of the processing load between different processors. Also, parallel algorithms suffer from node failure, thus, reducing the algorithm's scalability. As a result, the development of an efficient parallel algorithm that should be scalable and obtain high quality result is important. The MapReduce programming model [9] has recently become a very promising model for parallel processing. MapReduce is easier to understand, while MPI is somehow more complicated since the communication between processors need to be managed. MapReduce communicates between the nodes by disk operations (the shared data is stored in a distributed file system such as Hadoop Distributed File System), which is faster than local file systems, while MPI communicates via the message passing model. MapReduce provides fault-tolerance of node failures, while the MPI processes are terminated when a node fails, and need to be restarted manually.

1.2. Data Clustering

The core objective behind the clustering problem is to produce different groups

from data instances without any information about the instance labels. The clustering algorithm collects the similar data instances having common attributes and splits them into different partitions/clusters based on a similarity metric.

Clustering [3] is a widely studied data mining and most important unsupervised learning technique used when analyzing data. Clustering algorithms can be used in many applications, for instance, pattern recognition [4], document categorization [5], and bioinformatics applications [7]. The core objective behind the clustering problem is to produce different groups from data instances without any information about the instance labels. The clustering algorithm collects the similar data instances having common attributes and splits them into different partitions/clusters based on a similarity metric.

Generally, clustering algorithms can be classified into three basic classes [2]: partitional clustering, density clustering, and hierarchical clustering. The partitional clustering (e.g., K-means) [10] constructs several disjoint clusters and then evaluates them by some measure such as minimizing the squared errors among the cluster representatives (centroids) and data instances. The density based clustering approaches (e.g., DBSCAN) [11] apply a density criterion to locate the dense regions that have more connectivity between the cluster members and then separates them by low density regions. Hierarchical clustering [12] on the other hand, splits a big cluster into smaller ones (divisive) or merges smaller clusters into their nearest cluster (agglomerative) based on a similarity measure. In this dissertation, we are concerned with partitional clustering.

K-means clustering [10] is considered a common partitional clustering algorithm which is basically a minimization of the squared error objective function. K-means clustering suffers from some drawbacks such as the sensitivity of the initial centroids

and the local optima convergence problem.

1.3. Nature-inspired Algorithms

Nature-inspired algorithms mimick the different natural phenomena such as the concept of evolution and the behavior of the nature systems like Particle Swarm Optimization (PSO) [13], Glowworm Swarm Optimization (GSO) [14], Genetic algorithms (GA) [15], and Ant Colony Optimization (ACO) [16]. Swarm intelligence [17] is one of nature-inspired algorithms that simulates the natural swarms such as ant colonies, flocks of birds, bacterial growth, and schools of fishes. The behavior of the swarm is based on the sense of the member's interactions in the swarm by exchanging the local information with each other to help reaching the food sources. There is no central member in the swarm, but rather all swarm members participate equally to achieve the goal.

In recent years, some researchers discussed clustering based on the idea of swarm intelligence [17] such as, ant colony optimization [18] and particle swarm optimization [19]. The use of swarm intelligence clustering algorithms is very efficient since these algorithms avoid the k-means drawbacks of the initial number of centroids as well as premature convergence.

1.3.1. Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a swarm intelligence method first introduced by Kennedy and Eberhart in 1995 [13]. The behavior of PSO is inspired by bird flocks searching for optimal food sources, where the direction in which a bird moves is influenced by its current movement, the best food source it ever experienced, and the best food source any bird in the flock ever experienced. In PSO, the problem solutions, called particles, move through the search space by following the best particles. The movement of a particle is affected by its inertia, its personal best position, and the global best position. A swarm consists of multiple particles, each particle has a

fitness value which is assigned by the objective function to be optimized based on its position. Furthermore, a particle contains other information besides the fitness value and position such as the velocity which direct the moving of the particle. In addition, PSO maintains the best personal position with the best fitness value the particle has ever seen. Also, PSO holds the best global position with the best fitness value any particle has ever experienced. Many variants of PSO were introduced in literature. In our work, the Global Best PSO [13, 20] variant is used.

The following equation is used to move the particles inside the problem search space:

$$X_i(t+1) = X_i(t) + V_i(t+1) \tag{1.1}$$

where X_i is the position of particle i, t is the iteration number, and V_i is the velocity of particle i.

PSO uses the following equation to update the particle velocities, also used in our proposed algorithm:

$$V_i(t+1) = W \cdot V_i(t) + (r_1 \cdot cons_1) \cdot [XP_i - X_i(t)] + (r_2 \cdot cons_2) \cdot [XG - X_i(t)] \tag{1.2}$$

where W is inertia weight, r_1 and r_2 are randomly generated numbers, $cons_1$, $cons_2$ are constant coefficients, XP_i is the current best position of particle i and XG is the current best global position for the whole swarm.

1.3.2. Glowworm Swarm Optimization

Glowworm Swarm Optimization (GSO) [14] is an optimization algorithm, which belongs to the swarm intelligence field [17] that is inspired by simulated experiments of the behavior of insects that are called glowworms or lighting worms. These glowworms are able to control their light emission and use it to glow for different objectives such as e.g., attracting the other worms during the breeding season. Most swarm

intelligence algorithms are concerned with locating the global solution based on the objective function for the given optimization problem. In addition, locating one global solution is considered easier than locating multiple solutions. The GSO algorithm is especially useful for a simultaneous search of multiple solutions, having different or equal objective function values. To achieve this, a swarm must have the ability to divide itself into separated groups.

GSO was first introduced by Krishnan and Ghose in 2005 [14]. The swarm in the GSO algorithm is composed of N individuals called glowworms. A glowworm i has a position $X_i(t)$ at time t in the function search space, a light emission which is called the luciferin level $L_i(t)$, and a local decision range $rd_i(t)$. The luciferin level is associated with the objective value of the individual's position based on the objective function J.

A glowworm that emits more light (high luciferin level) means that it is closer to an actual position and has a high objective function value. A glowworm is attracted by other glowworms whose luciferin level is higher than its own within the local decision range. If the glowworm finds some neighbors with a higher luciferin level and within its local range, the glowworm moves towards them. At the end of the process, most glowworms will be gathered at the multiple peak locations in the search space.

The GSO algorithm consists of four main stages: glowworm initialization, luciferin level update, glowworm movement, and glowworm local decision range update.

In the first stage, N glowworms are randomly deployed in the specific objective function search space. In addition, in this stage the constants that are used for the optimization are initialized, and all glowworm luciferin levels are initialized with the same value (L_0). Furthermore, local decision range rd and radial sensor range r_s are initialized with the same initial value (r_0).

The luciferin level update is considered the most important step in GSO because

in this stage the objective function is evaluated at the current glowworm position (X_i). The luciferin level for all swarm members are modified according to the objective function values. The process for the luciferin level update is done with the following equation:

$$L_i(t) = (1 - \rho)L_i(t - 1) + \gamma J(X_i(t)) \tag{1.3}$$

where $L_i(t)$ and $L_i(t - 1)$ are the new luciferin level and the previous luciferin level for glowworm i, respectively, ρ is the luciferin decay constant ($\rho \in (0, 1)$), γ is the luciferin enhancement fraction, and $J(X_i(t))$ represents the objective function value for glowworm i at current glowworm position (X_i) at iteration t.

After that, and throughout the movement stage, each glowworm tries to extract the neighbor group $N_i(t)$ based on the luciferin levels and the local decision range (r_d) using the following rule:

$$j \in N_i(t) \; \textit{iff} \; d_{ij} < rd_i(t) \; \textit{and} \; L_j(t) > L_i(t) \tag{1.4}$$

where j is one of the glowworms near to glowworm i, $N_i(t)$ is the neighbor group, d_{ij} is the Euclidean distance between glowworm i and glowworm j, $rd_i(t)$ is the local decision range for glowworm i, and $L_j(t)$ and $L_i(t)$ are the luciferin levels for glowworm j and i, respectively.

After that, the actual selected neighbor is identified by two operations: the probability calculation operation to figure out the movement direction toward the neighbor with the higher luciferin value. This is done by applying the following equation:

$$Prob_{ij} = \frac{L_j(t) - L_i(t)}{\sum_{k \in N_i(t)} L_k(t) - L_i(t)} \tag{1.5}$$

where j is one of the neighbor group $N_i(t)$ of glowworm i.

After the probability calculation, in the second operation, glowworm i selects a glowworm from the neighbor group using the roulette wheel method whereby glowworm with the higher probability has more chance to be selected from the neighbor group.

Then, at the end of the glowworm movement stage, the position of the glowworm is modified based on the selected neighbor position using the following equation:

$$X_i(t) = X_i(t-1) + s \ \frac{X_j(t) - X_i(t)}{\delta_{ij}} \tag{1.6}$$

where $X_i(t)$ and $X_i(t-1)$ are the new position and previous position for the glowworm i, respectively, s is a step size constant, and δ_{ij} is the Euclidean Distance between glowworm i and glowworm j.

The last stage of GSO is the local decision range update, where the local decision range rd_i is updated in order to add flexibility to the glowworm to formulate the neighbor group in the next iteration. The following equation is used to update rd_i in the next iteration:

$$\begin{aligned} rd_i(t) = min\{rs, max[0, rd_i(t-1) \\ +\beta(nt - |N_i(t-1)|)]\} \end{aligned} \tag{1.7}$$

where $rd_i(t)$ and $rd_i(t-1)$ are the new local decision range, and the previous local decision range for glowworm i respectively, rs is the constant radial sensor range, β is a model constant, nt is a constant parameter used to control the neighbor count, and $|N_i(t)|$ is the actual number of neighbors.

1.4. MapReduce

The MapReduce distributed programming model [9], introduced by Google, has become very popular as an alternative model for data parallel programming over the

past few years compared to the MPI methodology [8]. The strength that makes the MapReduce to be good model for parallelizing the tasks is that the process can be performed automatically without having to know too many parallel programming details. Furthermore, the parallelization with MapReduce is remarkable because it presents a programming model that provides fault-tolerance, load balancing, and data locality. In addition, MapReduce moves the processing to the data and processes data sequentially to avoid random access that requires expensive seeks and disk times.

Apart from Google's implementation of MapReduce, there are several popular open source implementations available such as Apache Hadoop MapReduce [21], and Disco [22]. MapReduce is a highly scalable model and can be used across many computer nodes. In addition, MapReduce is suggested when the target problem is considered data-intensive and computing resources have restrictions on multiprocessing and large shared-memory hardware.

MapReduce usually divides the input data set into independent splits, which depend on the size of the data set and the number of computer nodes used. In MapReduce, the problem is formulated as a functional procedure using two core functions: the *Map* function and *Reduce* function. The basic idea behind the MapReduce model is the mapping of data into a list of <key,value> pairs, and then applying the reducing operation over all pairs with the same key. The *Map* function iterates over a large number of input units and processes them to extract intermediate output from each input unit, and all output values that have the same key are sent to the same *Reduce* function. On the other hand, the *Reduce* function collects the intermediate results with the same key that is retrieved by the *Map* function, and then merges and aggregates all intermediate results to generate the final results. Figure 1.1 shows the MapReduce's core functions.

Map Operation:

Map (k, v) $\rightarrow$ [(k', v')]

Reduce Operation:

Reduce (k', [v']) $\rightarrow$ [(k', v')]

Figure 1.1: The *Map* and *Reduce* operations.

Apache Hadoop [21] is the commonly used MapReduce implementation, and it is an open source software framework that supports data-intensive applications licensed under Apache. It enables applications to work with thousands of computational independent computers and petabytes of data. The main strengths of Hadoop are its scalability; it works with one machine, and can quickly grow to thousands of computer nodes developed to run on commodity hardware.

Apache Hadoop consists of two main components: Hadoop Distributed File System (HDFS), which is used for data storage, and MapReduce, which is used for data processing. HDFS supports the management and processing of large scale data sets. HDFS provides a high-throughput access to the data while maintaining fault tolerance to avoid the failure node issues by replicating multiple copies of data blocks. MapReduce works together with HDFS to provide the ability to move the computation to the data to maintain the data locality feature. Figure 1.2 shows the Hadoop architecture diagram and control flow between the two components. Interested readers may refer to [21] for more details. The Hadoop framework is used in our proposed algorithm implementations.

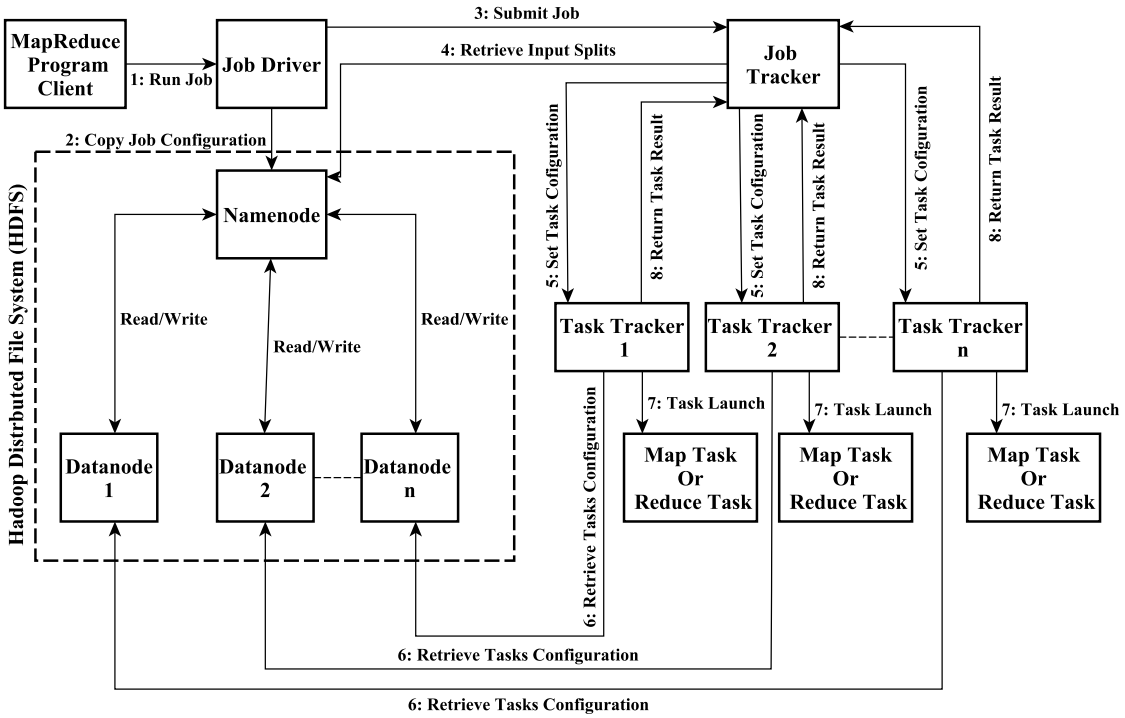


Figure 1.2: Hadoop architecture diagram.

1.5. Motivation and Problem Statement

The vast volumes of data require automated analysis methods to detect interesting patterns and extract knowledge. Traditional clustering techniques do not scale and are unable to construct models from huge collections of data. Another issue associated with the traditional clustering algorithms is the lack in handling multi-dimensionality with the rapid growth in data sizes. Furthermore, the enlargement in data sizes increases the computational and space complexities, which reduces the algorithm's performance in terms of runtime and memory requirements for data-intensive applications.

Many algorithms have been proposed to do data clustering, such as swarm intelligence algorithms. The use of swarm intelligence to solve the clustering problems has proved its efficiency [18, 19], since these algorithms avoid the partitioning-based clustering algorithm drawbacks as well as premature convergence.

The clustering based nature-inspired optimization algorithms find better solutions for clustering analysis problems by mapping the clustering problem as an optimization problem. These algorithms locate the optimal solution based on different similarity metrics to maintain high degree of cluster compactness. However, these algorithms face the same challenges as the traditional clustering techniques when being applied to large scale data.

To overcome these challenges, additional research is required to develop clustering algorithms that can be applied in real-world applications and deal with huge data sets. To build a scalable and efficient clustering algorithm, one methodology is using parallel computing models, such as MapReduce, which allow adding more computer nodes into the development environment to scale horizontally.

MapReduce has been recently applied to many data-intensive applications such as Bioinformatics [23] and Geosciences [24] applications, where codes are written using open source MapReduce tools. In addition, MapReduce has also been adopted by many companies in industry (e.g., Facebook [25], and Yahoo [26]).

The main motivation of this research is to investigate the role of the MapReduce framework for building scalable nature-inspired algorithms to solve clustering problems. Basically, we propose scalable algorithms in order to help the information technology community to use them for the big data analysis. Moreover, the proposed algorithms are applied to real-world applications such as intrusion detection.

1.6. Contributions

This dissertation makes several contributions towards scalable nature-inspired algorithms and outlines a design for newly developed clustering MapReduce based algorithms in a Hadoop environment. The contributions are:

1. A parallel particle swarm optimization clustering algorithm based on the MapReduce framework is presented, which makes use of the MapReduce framework that has been proven successful as a parallelization methodology for data-intensive applications. The proposed algorithm has been tested on large scale synthetic data sets with different sizes to show its speedup and scalability. In addition, the proposed algorithm has been tested on real data sets with different settings to demonstrate its effectiveness and quality.

2. A parallel intrusion detection system based on the MapReduce framework is presented. The proposed system incorporates clustering analysis to build the detection model by formulating the intrusion detection problem as an optimization problem. Furthermore, the proposed system has been tested on a real large-scale intrusion data set with different training subset sizes to show its speedup and scalability, and to present its detection quality.

3. The MapReduce methodology is utilized to create a parallel glowworm swarm optimization algorithm. The purpose of applying MapReduce to GSO goes further than merely being a hardware utilization. Rather, a distributed model is developed, which achieves better solutions since it is scalable with a overall reduced computation time. The proposed algorithm has been tested on large scale multimodal benchmark functions with different dimensions to show the speedup and scalability while maintaining the optimization quality.

4. Making use of GSO optimization to solve the clustering problem, which takes

into account the advantages of GSO's ability to search for optimal centroids simultaneously. The proposed algorithm is designed to discover the clusters without the need to provide the number of clusters in advance. Furthermore, different fitness functions are introduced to add flexibility and robustness to the proposed algorithm. Furthermore, the proposed clustering algorithm is parallelized using the MapReduce methodology to extend the algorithm to work with large data sets. The proposed algorithm and the parallel-based version are tested on real and artificial data sets with different shapes to demonstrate the clustering quality as well as the algorithm speedup.

1.7. Dissertation Overview

This dissertation is a paper-based version, where each chapter has been derived from papers published during the PhD work. This is an overview of the remaining chapters of this dissertation:

1. In **Chapter 2**, a parallel PSO algorithm that is based on MapReduce for data clustering is described. This chapter is derived from the publication: *Ibrahim Aljarah and Simone A. Ludwig, "Parallel Particle Swarm Optimization Clustering Algorithm based on MapReduce Methodology", In Proceedings of the Fourth World Congress on Nature and Biologically Inspired Computing (IEEE NaBIC12), Mexico City, Mexico, November 2012.*

2. In **Chapter 3**, an intrusion detection system based on a parallel PSO clustering algorithm using the MapReduce methodology is introduced. This chapter is derived from the publication, which is accepted as a short paper in *Proceeding of Genetic and Evolutionary Computation Conference (ACM GECCO13), Amsterdam, July 2013* which is titled as *"Towards a Scalable Intrusion Detection System based on Parallel PSO Clustering Using MapReduce"*. Furthermore,

the long version of the paper is accepted in *IEEE Congress on Evolutionary Computation (CEC13) Conference*.

3. In **Chapter 4**, we outline how GSO can be modeled based on the MapReduce parallel programming model. We describe the implementation with MapReduce and present how GSO can be naturally expressed by this model. This chapter is derived from the publication: *Ibrahim Aljarah and Simone A. Ludwig, "A MapReduce based Glowworm Swarm Optimization Approach for Multimodal Functions", In Proceedings of the IEEE Symposium Series on Computational Intelligence (SSCI13), Singapore, April 2013.*

4. In **Chapter 5**, we proposed a new clustering algorithm based GSO optimization, where GSO is adjusted to solve the data clustering problem to locate multiple optimal centroids based on the multimodal search capability of GSO. Furthermore, we introduced special fitness functions to evaluate the goodness of the GSO individuals achieving high quality clusters. Moreover, a parallel version of the proposed GSO clustering algorithm using MapReduce is introduced to enable the algorithm to work on large scale data sets. Part of this chapter is derived from the publication: *Ibrahim Aljarah and Simone A. Ludwig, "A New Clustering Approach based on Glowworm Swarm Optimization", In Proceedings of 2013 IEEE Congress on Evolutionary Computation Conference (IEEE CEC13), Cancun, Mexico, June 2013.*

5. In **Chapter 6**, we conclude the dissertation by summarizing the contributions. It also provides directions to future work that could be addressed.

CHAPTER 2. PARALLEL PARTICLE SWARM OPTIMIZATION CLUSTERING ALGORITHM BASED ON MAPREDUCE METHODOLOGY

Large scale data sets are difficult to manage. Difficulties include capture, storage, search, analysis, and visualization of large data. In particular, clustering of large scale data has received considerable attention in the last few years and many application areas such as bioinformatics and social networking are in urgent need of scalable approaches. The new techniques need to make use of parallel computing concepts in order to be able to scale with increasing data set sizes. In this chapter, we propose a parallel particle swarm optimization clustering (MR-CPSO) algorithm that is based on MapReduce. The experimental results reveal that MR-CPSO scales very well with increasing data set sizes and achieves a very close to the linear speedup while maintaining the clustering quality. The results also demonstrate that the proposed MR-CPSO algorithm can efficiently process large data sets on commodity hardware.

The rest of this chapter is organized as follows: Section 2.1 presents the related work in the area of parallel data clustering algorithms. In Section 2.2, the proposed MR-CPSO algorithm is introduced. Section 2.3 presents the experimental evaluation, and Section 2.4 presents chapter conclusions.

2.1. Related Work

MapReduce has recently received a significant amount of attention in many computing fields but especially in the data mining area. Clustering has numerous applications and is becoming more challenging as the amount of data rises. Due to space constraints, we focus only on closely related work of parallel data clustering algorithms that employ the MapReduce methodology.

Zaho et al. in [27] proposed a parallel algorithm for k-means clustering based on MapReduce. Their algorithm randomly selects initial k objects as centroids. Then,

centroids are calculated by the weighted average of the points within a cluster through the *Map* function; afterwards the *Reduce* function updates the centroids based on the distances between the data points and the previous centroids in order to obtain new centroids. Then, an iterative refinement technique is applied by MapReduce job iterations.

Li et al. in [28] proposed another MapReduce K-means clustering algorithm that uses the ensemble learning method bagging to solve the outlier problem. Their algorithm shows that the algorithm is efficient on large data sets with outliers.

Surl et al. [29] applied the MapReduce framework on co-clustering problems introducing a practical approach that scales well and achieves efficient performance with large data sets. The authors suggested that applying MapReduce on co-clustering mining tasks is very important, and discussed the advantages in many application areas such as collaborative filtering, text mining, etc. Their experiments were done using 3 real data sets and the results showed that the co-clustering with MapReduce can scale well with large data sets.

A fast clustering algorithm with constant factor approximation guarantee was proposed in [30], where they use sampling to decrease the data size and run a time consuming clustering algorithm such as local search on the resulting data set. A comparison of this algorithm with several sequential and parallel algorithms for the k-median problem was done using randomly generated data sets and a single machine where each machine used by the algorithms was simulated. The results showed that the proposed algorithm obtains better or similar solutions compared to the other algorithms. Moreover, the algorithm is faster than other parallel algorithms on very large data sets. However, for the k-median problem they have a small loss in performance.

In [31], the authors explored how to minimize the I/O cost for clustering with

the MapReduce model and tried to minimize the network cost among the processing nodes. The proposed technique BOW (Best Of both Worlds), is a subspace clustering method to handle very large data sets in efficient time and derived its cost functions that allow the automatic, dynamic differentiation between disk delay and network delay. Experiments on real and synthetic data of millions of points with good speedup results were reported.

In this chapter, the clustering task is expressed as an optimization problem to obtain the best solution based on the minimum distances between the data points and the cluster centroids. For this task, we used PSO algorithm [13] as it performs a globalized search to find the best solution for the clustering task problem (this solves the K-means [32, 2] sensitivity of the selection of the initial cluster centroids and avoids the local optima convergence problem). PSO is one of the common optimization techniques that iteratively proceeds to find the best solution based on a specific measure.

PSO has been used to solve a clustering task in [33], where the problem discussed was document clustering. The authors compared their results with K-Means, whereby the PSO algorithm proved to generate more compact clustering results. However, in this chapter we are validating this approach with more generalized and much larger data sets. In addition, the MapReduce framework has been chosen as the parallelization technique in order to tackle the computational and space complexities that large data sets incur causing an efficiency degradation of the clustering.

To the best of our knowledge, this is the first work that implements PSO clustering with MapReduce. Our goal is to show that PSO clustering benefits from the MapReduce framework and works on large data sets achieving high clustering quality, scalability, and a very good speedup.

2.2. Proposed MapReduce PSO Clustering Algorithm (MR-CPSO)

In the MR-CPSO algorithm, we formulated the clustering task as an optimization problem to obtain the best solution based on the minimum distances between the data points and the cluster centroids. The MR-CPSO is a partitioning clustering algorithm similar to the k-means clustering approach, in which a cluster is represented by its centroid. In k-means clustering, the centroid is calculated by the weighted average of the points within a cluster. In MR-CPSO, the centroid for each cluster is updated based on the swarm particles' velocities.

In MR-CPSO, each particle P_i contains information which is used in the clustering process such as:

- Centroids Vector (CV): Current cluster centroids vector.
- Velocities Vector (VV): Current velocities vector.
- Fitness Value (FV): Current fitness value for the particle at iteration t.
- Best Personal Centroids (BPC): Best personal centroids seen so far for P_i.
- Best Personal Fitness Value (BPC_{FV}): Best personal fitness value seen so far for P_i.
- Best Global Centroids (BGC): Best global centroids seen so far for whole swarm.
- Best Global Fitness Value (BGC_{FV}): Best global fitness value seen so far for whole swarm.

This information is updated in each iteration based on the previous swarm state.

In MR-CPSO, two main operations need to be adapted and implemented to apply the clustering task on large scale data: the fitness evaluation, and particle centroids updating.

Particle centroids updating is based on PSO movement Equations 1.1 and 1.2 that calculate the the new centroids in each iteration for the individual particles. The particle centroids update takes a long time, especially when the particle swarm size is large.

Besides the update of the particle centroids, the fitness evaluations are based on a fitness function that measures the distance between all data points and particle centroids by taking the average distance between the particle centroids. The fitness evaluation is based on the following equation:

$$Fitness = \frac{\sum_{j=1}^{k} \frac{\sum_{i=1}^{n_j} Distance(R_i, C_j)}{n_j}}{k} \tag{2.1}$$

where n_j denotes the number of records that belong to cluster j; R_i is the i^{th} record; k is the number of available clusters; $Distance(R_i, C_j)$ is the distance between record R_i and the cluster centroid C_j. In this chapter, we use the Manhattan distance applying the following equation:

$$Distance(R_i, C_j) = \sum_{v=1}^{D} |R_{iv} - C_{jv}| \tag{2.2}$$

where D is the dimension of record R_i; R_{iv} is the value of dimension v in record R_i; C_{jv} is the value of dimension v in centroid C_j.

The fitness evaluation takes a long time to execute when working with large data sets. For example, if the data set contains 5 million data points with 10 dimensions, and the number of clusters is 5, the swarm size is 30, then the algorithm needs to calculate $5 \times 10^6 \times 5 \times 10 \times 30 = 75 \times 10^8$ distance values for one iteration. This takes 400 minutes running on a 3.2 GHz processor.

The MR-CPSO algorithm consists of three main sub-modules:

- The first module is a MapReduce job to update the particle swarm centroids.

- The second module is a MapReduce job for the fitness evaluation of the swarm with new particle centroids that are generated in the first module.

- The third module (merging) is used to merge the fitness values calculated from the second module with the updated swarm which is generated in the first module. Also, in this module, the best personal centroids and best global centroids are updated. Afterwards, the new particles are ready for the next iteration. Figure 2.1 shows the MR-CPSO architecture diagram.

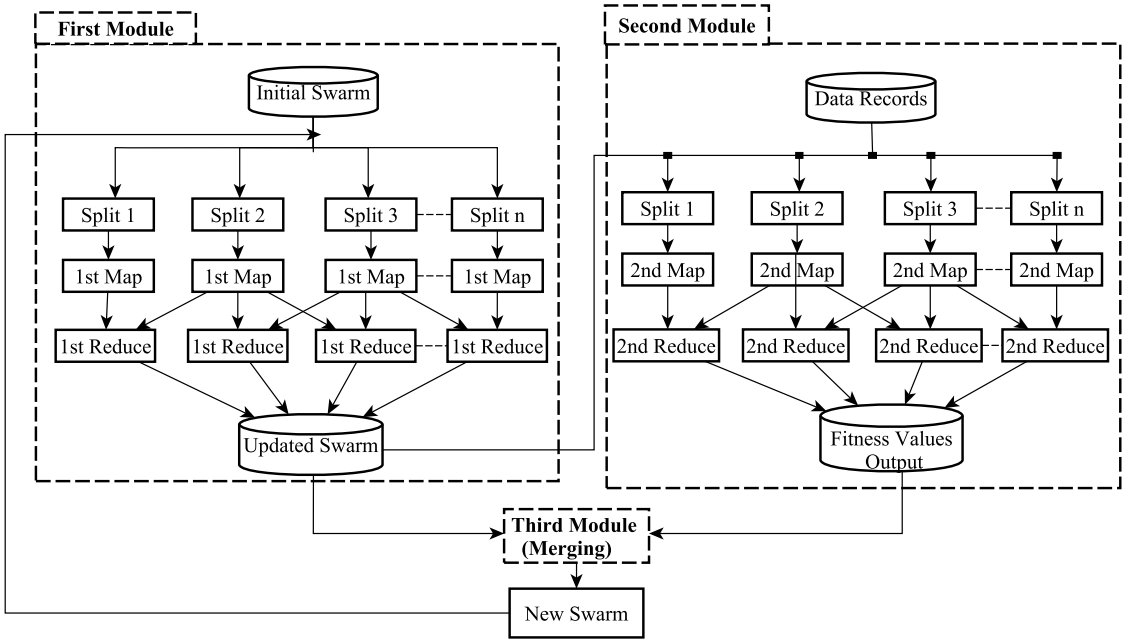


Figure 2.1: MR-CPSO algorithm architecture diagram.

2.2.1. First Module

In the first module, the MapReduce job is launched for updating the particle centroids. The *Map* function receives the particles with identification numbers. However, the particle ID represents the *Map* key and the particle itself represents the value. The *Map* value contains all information about the particle such as CV, VV, FV, BPC and BGC, which are used inside the *Map* function. In the *Map* function, the centroids are updated based on the PSO Equations 1.1 and 1.2. The other

information such as PSO coefficients ($cons_1$,$cons_2$), inertia weight (W), which are used in the PSO equations, are retrieved from the job configuration file. After that, the *Map* function emits the particle with updated centroids to the *Reduce* function. To benefit from the MapReduce framework, we use the number of *Maps* relative to the number of cluster nodes and swarm size. The *Reduce* function in the first module is only an identity reduce function that is used to sort the *Map* results and combine all of them into one output file. Furthermore, the particle swarm is saved in the distributed file system to be used by two other modules. The pseudo-code of the *Map* function and *Reduce* function is shown in Algorithm 2.1.

2.2.2. Second Module

In the second module, the MapReduce job is launched to calculate the new fitness values for the updated swarm. The *Map* function receives the data records with recordID numbers. The recordID represents the *Map* key and the data record itself represents the value. The *Map* and *Reduce* functions work as shown in the Algorithm 2.2 outlining the pseudo code of the second module algorithm. The *Map* function process starts with retrieving the particle swarm from the distributed cache, which is a feature provided by the MapReduce framework for caching files. Then, for each particle, the *Map* function extracts the centroids vector and calculates the distance value between the record and the centroids vector returning the minimum distance with its centroidID. The *Map* function uses the ParticleID with its centroidID that has the minimum distance to formulate a new composite key. Also, a new value is formulated from the minimum distance. After that, the *Map* function emits the new key and new value to the *Reduce* function. The *Reduce* function aggregates the values with the same key to calculate the average distances and assigns it as a fitness value for each centroid in each particle. Then, the *Reduce* function emits the key with average distance to formulate the new fitness values. Then, the new fitness values are

stored in the distributed file system.

Algorithm 2.1 First module.

```
function Map (Key: particleID, Value: Particle)
    particleID=Key
    particle=Value
    extractInfo(CV,VV,BPC,BGC)   ▷ Extract the information from the particle
    for each c_i in CV do                ▷ Generate random numbers r_1 and r_2
        for each j in Dimension do                    ▷ update particle velocity
            newVV_ij= w*VV_ij +(r_1*cons_1)*(BPC-c_ij)+(r_2*cons_2)*(BGC-c_ij)
            newc_ij=c_ij + newVV_ij
        end for
        update(particle,newVV_i, newc_i)
    end for
    Emit(particleID, particle)
end function

function Reduce (Key: ParticleID, ValList: Particle)
    for each Value in ValList do
        Emit(Key, Value)
    end for
end function
```

2.2.3. Third Module (Merging)

In the third module of the MR-CPSO algorithm, the main goal is to merge the outputs of the first and second modules in order to have a single new swarm. The new fitness value (FV) is calculated on the particle level by a summation over all centroids' fitness values generated by the second module. After that, the swarm is updated with the new fitness values. Then, BPC_{FV} for each particle is compared with the new particle fitness value. If the new particle fitness value is less than the current BPC_{FV}, BPC_{FV} and its centroids are updated. Also, the BGC_{FV} with centroids is updated if there is any particle's fitness value smaller than the current BGC_{FV}. Then, the new swarm with new information is saved in the distributed file system to be used as input for the next iteration.

Algorithm 2.2 Second module.

```
function Map (key: RecordID, Value: Record)
    RID=key
    record=value
    read(Swarm)                ▷ Read the particles swarm from the Distributed Cache
    for each particle in Swarm do
        CV=extractCentroids(particle)
        PID=extractPID(particle)
        minDist=returnMinDistance(record,CV)
        centroidID=i                          ▷ i^th centroid contains minDist
        newKey=(PID,centroidID)
        newValue=(minDist)
        Emit(newKey, newValue)
    end for
end function

function Reduce (Key:(PID,centerId),ValList:(minDist,1))
 count=0, sumDist=0, avgDist=0
    for each Value in ValList do
        minDist=extractminDist( Value )
        count=count + 1
        sumDist=sumDist + minDist
    end for
    avgDist=sumDist / count
    Emit(Key, avgDist)
end function
```

2.3. Experiments and Results

In this section, we describe the clustering quality and discuss the running time of the measurements for our proposed algorithm. We focus on scalability as well as the speedup and the clustering quality.

2.3.1. Environment

We ran the MR-CPSO experiments on the Longhorn Hadoop cluster hosted by the Texas Advanced Computing Center (TACC)[1] and on our NDSU[2] Hadoop cluster. The Longhorn Hadoop cluster is one of the common Hadoop cluster that is used by

[1]https://portal.longhorn.tacc.utexas.edu/
[2]http://www.ndsu.edu

researchers. The Longhorn Hadoop cluster contains 384 compute cores and 2.304 TB of aggregated memory. The Longhorn Hadoop cluster has 48 nodes containing 48GB of RAM, 8 Intel Nehalem cores (2.5GHz each), whereas our NDSU Hadoop cluster consists of only 18 nodes containing 6GB of RAM, 4 Intel cores (2.67GHz each) with HDFS 2.86 TB aggregated capacity. For our experiments, we used Hadoop version 0.20 (new API) for the MapReduce framework, and Java runtime 1.6 to implement the MR-CPSO algorithm.

2.3.2. Data Sets

To evaluate our MR-CPSO algorithm, we used both real and synthetic data sets as described in Table 2.1.

Table 2.1: Summary of the data sets.

Data set	**#Records**	**#Dim**	**Size (MB)**	**Type**	**#Clusters**
MAGIC	19,020	10	3.0	*Real*	2
Electricity	45,312	8	6.0	*Real*	2
Poker	1,025,010	10	49.0	*Real*	10
CoverType	581,012	54	199.2	*Real*	7
F2m2d5c	2,000,000	2	83.01	*Synthetic*	5
F4m2d5c	4,000,000	2	165.0	*Synthetic*	5
F6m2d5c	6,000,000	2	247.6	*Synthetic*	5
F8m2d5c	8,000,000	2	330.3	*Synthetic*	5
F10m2d5c	10,000,000	2	412.6	*Synthetic*	5
F12m2d5c	12,000,000	2	495.0	*Synthetic*	5
F14m2d5c	14,000,000	2	577.9	*Synthetic*	5
F16m2d5c	16,000,000	2	660.4	*Synthetic*	5
F18m2d5c	18,000,000	2	743.6	*Synthetic*	5
F30m2d5c	30,000,000	2	1238.3	*Synthetic*	5
F32m2d5c	32,000,000	2	1320.8	*Synthetic*	5

The real data sets that are used are the following:

- MAGIC: represents the results of registration simulation of high energy gamma particles in a ground-based atmospheric Cherenkov gamma telescope using the

imaging technique. It was obtained from UCI machine learning repository[4].

- Electricity: contains electricity prices from the Australian New South Wales Electricity Market. The clustering process identifies two states (UP or DOWN) according to the change of the price relative to a moving average of the last 24 hours. Obtained from MOA[5].

- Poker Hand: is an examples of a hand consisting of five playing cards drawn from a standard deck of 52 cards. Each card is described using 10 attributes and the data set describes 10 poker hand situations (clusters). It was obtained from UCI[4].

- Cover Type: represents cover type for 30 x 30 meter cells from US Forest. The real data set is obtained from the UCI[4]. It has 7 clusters that represent the type of trees.

- Synthetic: two series of data sets with different sizes of records were generated using the data generator developed in [34]. The first series are 9 data sets ranging from 2 million to 18 million data records. The second series are 2 data sets with 30 million and 32 million data records. In order to simplify the names of the synthetic data sets, the data sets' names consist of the specific pattern based on the data records number, the number of dimensions, and the number of the clusters. For example: the F2m2d5c data set consists of 2 million data records, each record is in 2 dimensions, and the data set is distributed into 5 clusters.

[4]http://archive.ics.uci.edu/ml/index.html
[5]http://moa.cs.waikato.ac.nz/datasets/

2.3.3. Evaluation Measures

In our experiments, we used the parallel scaleup [35] and speedup [35] measures calculated using Equations 2.3 and 2.4, respectively. These measures are used to evaluate the performance of our MR-CPSO algorithm. Scaleup is a measure of speedup that increases with increasing data set sizes to evaluate the ability of the parallel algorithm utilizing the cluster nodes effectively.

$$Scaleup = \frac{T_{SN}}{T_{2SN}} \tag{2.3}$$

where the T_{SN} is the running time for the data set with size S using N nodes and T_{2SN} is the running time using 2-fold of S and 2-folds of N nodes.

For the Speedup measurement, the data set is fixed and the number of cluster nodes is increased by a certain ratio.

$$Speedup = \frac{T_2}{T_n} \tag{2.4}$$

where T_2 is the running time using 2 nodes, and T_n is the running time using n nodes, where n is a multiple of 2.

We evaluate the scaleup by increasing the data set sizes and number of cluster nodes with the same ratio.

For the clustering quality, we used the purity measure [36] in Equation 2.5 to evaluate the MR-CPSO clustering correctness.

$$Purity = \frac{1}{N} \times \sum_{i=1}^{k} max_j(\mid C_i \cap L_j \mid) \tag{2.5}$$

where C_i contains all the points assigned to cluster i by MR-CPSO, and L_j denotes the true assignments of the points in cluster j; N is the number of records in the data set.

We used the PSO settings that are recommended by [37, 38]. We used a swarm size of 100 particles and inertia weight W of 0.72. Also, we set the acceleration coefficient constants $cons_1$ and $cons_2$ to 1.7.

2.3.4. Results

We used the real data sets to evaluate the correctness of the MR-CPSO algorithm. We compared the purity results of MR-CPSO with the standard K-means algorithm, which is implemented in the Weka data mining software [39], in order to perform a fair comparison of the purity values. The maximum iterations used for K-means and MR-CPSO is 25.

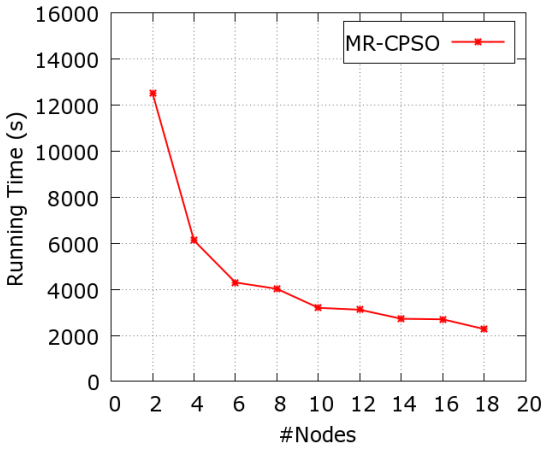
A comparison of the clustering quality in terms of purity with K-means clustering method is shown in Table 2.2. It can been seen from the Table 2.2, MR-CPSO outperforms the K-means technique for all real data sets with purity of 0.65, 0.58, 0.51, and 0.53 for MAGIC, Electricity, Poker, and Cover Type, respectively.

Table 2.2: Purity results.

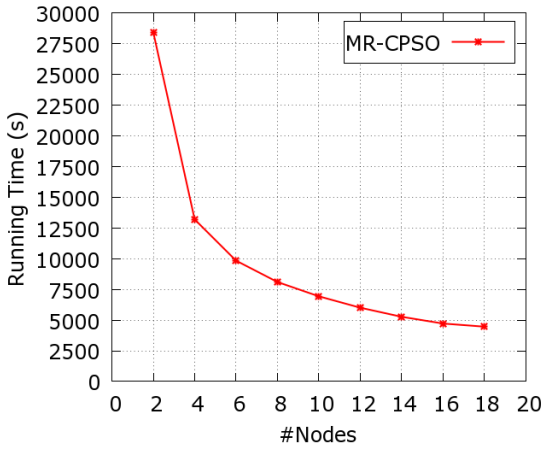
Data Set	**MR-CPSO**	**K-means**
MAGIC	0.65	0.60
Electricity	0.58	0.51
Poker	0.51	0.11
Cover Type	0.53	0.32

We used MR-CPSO for clustering different sizes of synthetic data sets. We ran MR-CPSO with 18 NDSU cluster nodes by increasing the number of nodes in each run by multiples of 2. In each run, we report the running time and speedup of 25 iterations of MR-CPSO. The running times and speedup measures are shown in Figures 2.2 and 2.3, respectively.

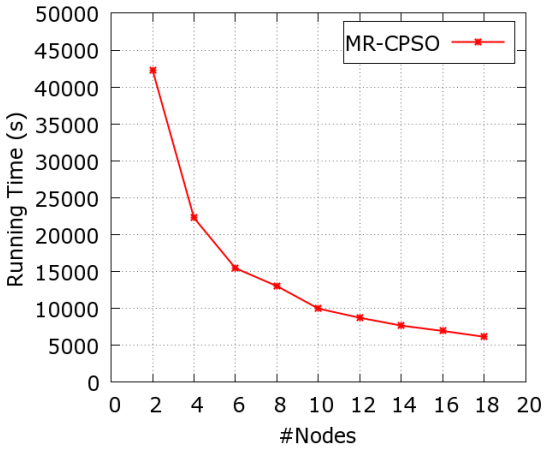
As can be noted from the Figure 2.2, the improvement factor of MR-CPSO's running times for the F2m2d5c, F4m2d5c, F6m2d5c, F8m2d5c, F10m2d5c data sets using 18 nodes are 5.5, 6.4, 6.9, 7.4, 7.8, respectively, compared to the running time



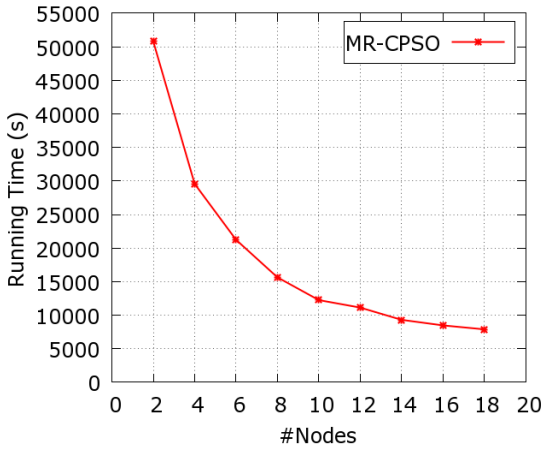
(a) F2m2d5c Running Time



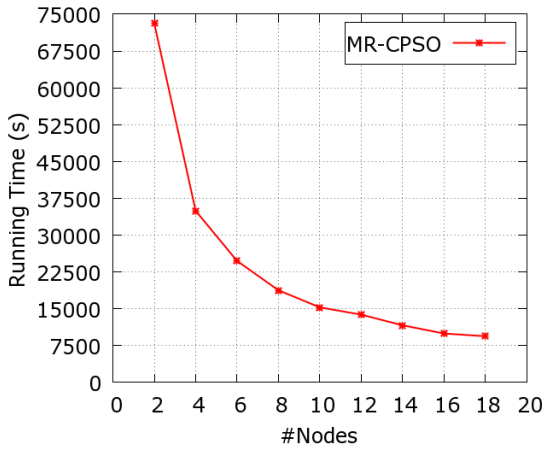
(b) F4m2d5c Running Time



(c) F6m2d5c Running Time

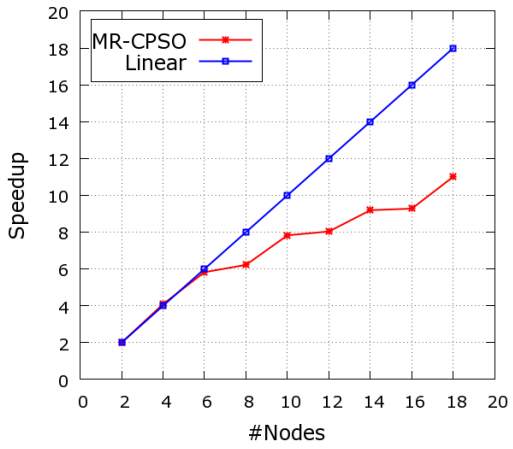


(d) F8m2d5c Running Time

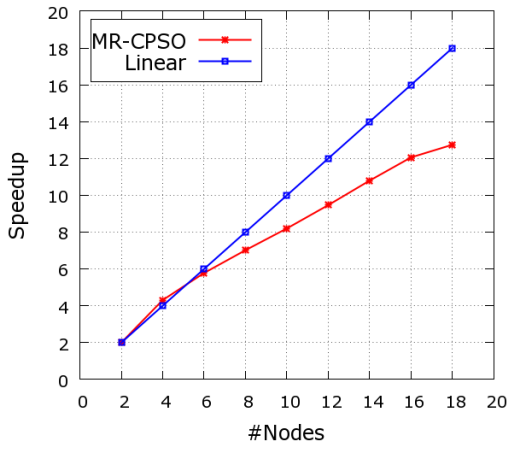


(e) F10m2d5c Running Time

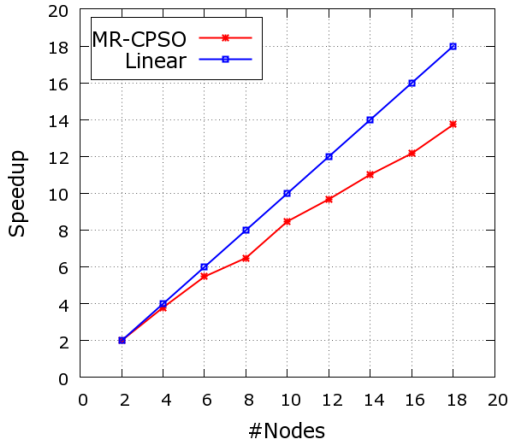
Figure 2.2: Running time results on the synthetic data sets from 2 million to 10 million data records with 18 NDSU Hadoop cluster nodes and 25 iterations of MR-CPSO.



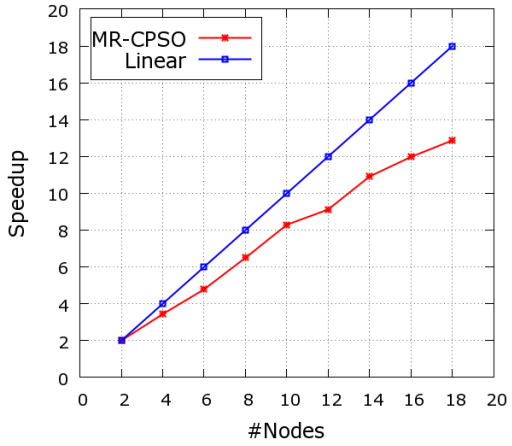
(a) F2m2d5c Speedup



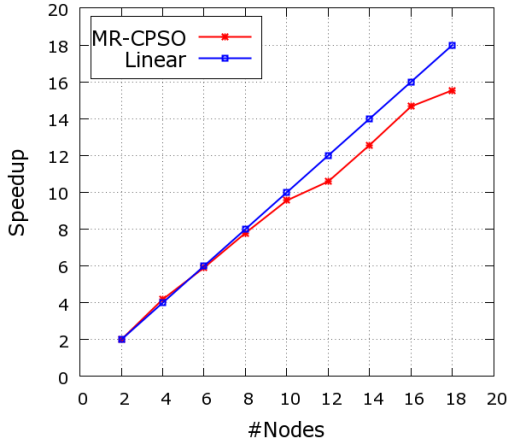
(b) F4m2d5c Speedup



(c) F6m2d5c Speedup



(d) F8m2d5c Speedup



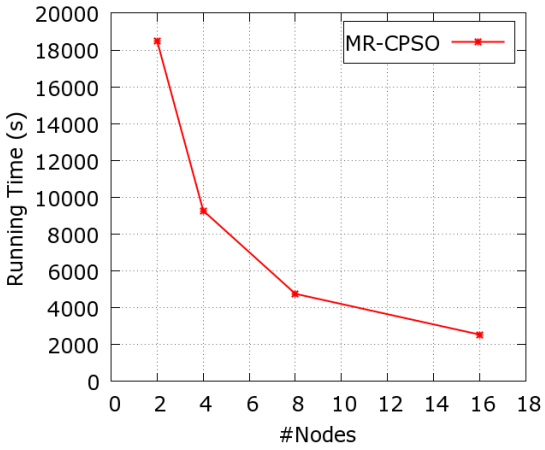
(e) F10m2d5c Speedup

Figure 2.3: Speedup results on the synthetic data sets from 2 million to 10 million data records with 18 NDSU Hadoop cluster nodes and 25 iterations of MR-CPSO.

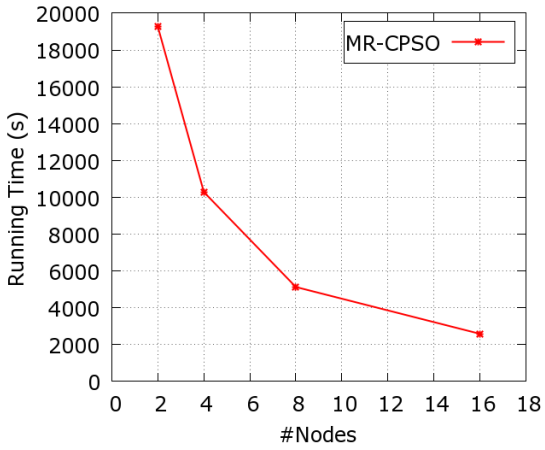
with 2 nodes. The MR-CPSO algorithm demonstrates a significant improvement in running time. Furthermore, the running time of MR-CPSO decreases almost linearly with increasing number of nodes of the Hadoop cluster. In addition, the MR-CPSO speedup in the Figure 2.3 scales close to linear for most data sets. MR-CPSO algorithm with F10m2d5c achieves a significant speedup obtaining very close to the linear speedup.

Figure 2.4 shows the running time and speedup of MR-CPSO with larger data sets using the Longhorn Hadoop cluster. We used 16 nodes as the maximum number of nodes to evaluate the MR-CPSO performance since we have limited resources on the Longhorn cluster. The running time results for the two data sets decreases when the number of nodes of the Hadoop cluster increases. The improvement factor of MR-CPSO running times for the F30m2d5c and F32m2d5c data sets with 18 nodes are 7.27, 7.43 compared to the running time with 2 nodes. The MR-CPSO algorithm shows a significant improvement in running time. The MR-CPSO algorithm with F30m2d5c and F32m2d5c achieve a significant speedup which is almost identical to the linear speedup. Thus, if we want to cluster even larger data sets with the MR-CPSO algorithm, we can accomplish that with a good performance by adding nodes to the Hadoop cluster.

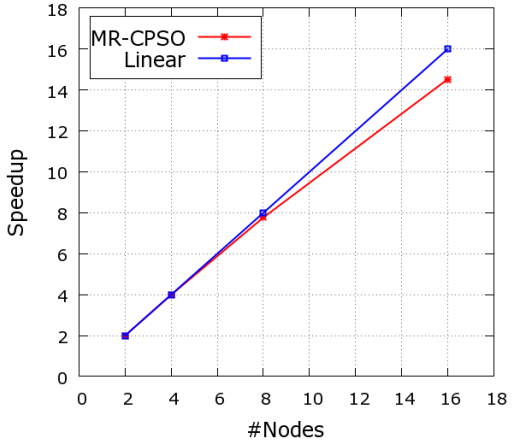
Figure 2.5 shows the scaleup measure of MR-CPSO for increasing double folds of data set sizes (starting from 2, 4, 6, 8 to 18 million data records) with the same double folds of nodes (2, 4, 6, 8 to 18 nodes), implemented on the NDSU Hadoop cluster. Scaleup for F4m2d5c was 0.85, and it captures almost a constant ratio between 0.8 and 0.78 when we increase the number of available nodes and data set sizes with same ratio.



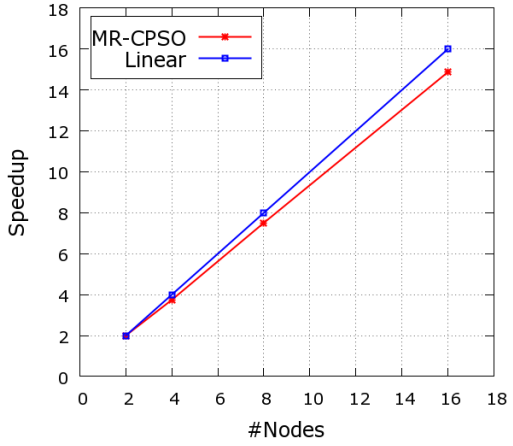
(a) F30m2d5c Running Time



(b) F32m2d5c Runnimg Time



(c) F30m2d5c Speedup



(d) F32m2d5c Speedup

Figure 2.4: Running time and speedup results on the synthetic data sets with Longhorn Hadoop cluster and 10 iterations of MR-CPSO. 2.4(a), 2.4(b) Running times for synthetic data sets for 30 million and 32 million data records. 2.4(c), 2.4(d) Speedup measure for synthetic data sets for 30 million and 32 million data records.

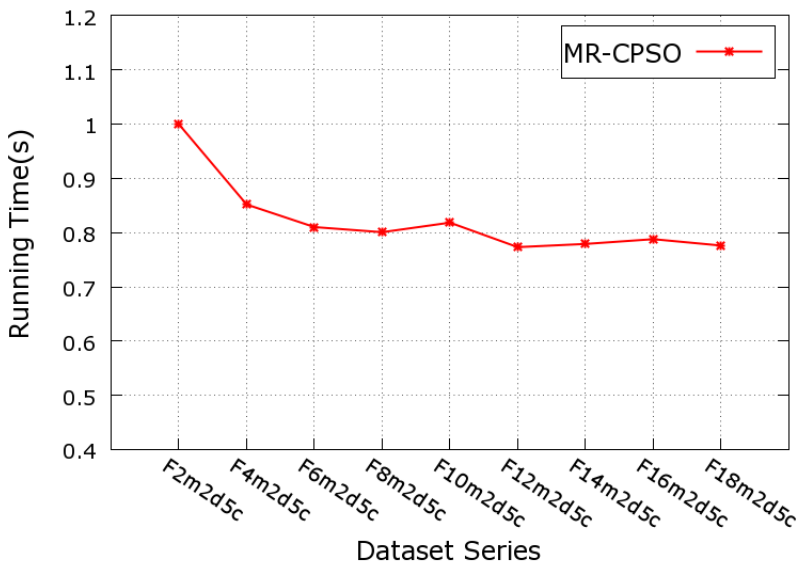


Figure 2.5: MR-CPSO scaleup.

2.4. Conclusion

In this chapter, we proposed a scalable MR-CPSO algorithm using the MapReduce parallel methodology to overcome the inefficiency of PSO clustering for large data sets. We have shown that the MR-CPSO algorithm can be successfully parallelized with the MapReduce running on commodity hardware. The clustering task in MR-CPSO is formulated as an optimization problem to obtain the best solution based on the minimum distances between the data points and the cluster centroids. The MR-CPSO is a partitioning clustering algorithm similar to the k-means clustering approach, in which a cluster is represented by its centroid. The centroid for each cluster is updated based on the particles' velocities. Experiments were conducted with both real-world and synthetic data sets in order to measure the scaleup and speedup of our algorithm. The results reveal that MR-CPSO scales very well with increasing data set sizes, and scales very close to the linear speedup while maintaining good clustering quality. The results also show that the clustering using MapReduce is better than the K-means sequential algorithm in terms of clustering quality.

CHAPTER 3. MAPREDUCE INTRUSION DETECTION SYSTEM BASED ON A PARTICLE SWARM OPTIMIZATION CLUSTERING ALGORITHM

The increasing volume of data in large networks to be analyzed imposes new challenges to an intrusion detection system. Since data in computer networks is growing rapidly, the analysis of these large amounts of data to discover anomaly fragments has to be done quickly. Some of the past and current intrusion detection systems are based on a clustering approach. However, in order to cope with the increasing amount of data, new parallel methods need to be developed in order to make the algorithms scalable. In this chapter, we propose an intrusion detection system based on a parallel particle swarm optimization clustering algorithm using the MapReduce methodology. The use of particle swarm optimization for the clustering task is a very efficient way since particle swarm optimization avoids the sensitivity problem of initial cluster centroids as well as premature convergence. The proposed intrusion detection system processes large data sets on commodity hardware. The experimental results on a real intrusion data set demonstrate that the proposed intrusion detection system scales very well with increasing data set sizes. Moreover, it achieves close to the linear speedup by improving the intrusion detection and false alarm rates.

The rest of this chapter is organized as follows: Section 3.1 briefly introduces intrusion detection systems. Section 3.2 presents the related work in the area of anomaly-detection algorithms based on clustering. In Section 3.3, our proposed IDS-MRCPSO system is introduced. Section 3.4 presents the experimental evaluation, and Section 3.5 presents our conclusions.

3.1. Introduction

Network intrusion detection has been identified as one of the most challenging

needs of the network security community in recent years. This is because of the inflated number of users and the amount of data exchanged which makes it difficult to distinguish the normal data connections from others that contain attacks. This requires the development of intrusion detection systems (IDSs) that can analyze large amounts of data in a reasonable time in order to take appropriate actions against the attacks.

IDSs are classified based on their analysis model and placement approach. In the analysis approach, IDSs are categorized into two classes: misuse and anomaly detection. In the misuse-based class, the IDS checks the network and system activity for a known misuse pattern that was identified beforehand through a pattern matching algorithm.

The anomaly-detection based IDS works differently whereby the decisions are made based on a profile of a normal network or system behavior, often constructed using statistical or machine learning techniques. Each of these approaches offer its strengths and weaknesses. Misuse-based systems generally have very low false positive rates that indicate error rates of mistakenly detected non-intrusion cases. Therefore, this approach is seen in the majority of commercial systems. In addition, the misuse-based systems are unable to identify novel or obfuscated attacks.

On the other hand, anomaly-based IDSs are able to detect new attacks that have not been seen before. However, this model produces a large number of false positives. The reason for this is the inability of current anomaly-based techniques to cope adequately with the fact that in the real world, normal, legitimate computer networks, and system usage changes over time. This implies that any profile of normal behavior needs to be dynamic. Thus, with the exponential growth in the different types of attacks, a pattern-matching algorithm is not trust-worthy in the misuse approach, and therefore, it is recommended to use both in an IDS [40].

The placement approach is usually divided into host-based and network-based systems. An IDS which operates on a computer to detect malicious activity on that host, is called a host-based IDS; whereas an IDS that tries to detect events of interest by analyzing and monitoring network traffic data is called a network-based IDS [41]. Network-based IDSs detect attacks by analyzing network packet traffic along a network segment or switch, enabling the monitoring and protection of multiple hosts by a separate machine. Host-based IDS systems have the ability to determine if an attempted attack was successful or not in a local machine. Network-based systems are able to monitor a large number of hosts with relatively low deployment costs in comparison to host-based systems, and are able to identify attacks to and from multiple hosts.

There are several different data mining techniques that have been used for IDSs in the past, which include supervised (data classification) or unsupervised techniques (data clustering) depending on whether the class labels are known during the learning process or not. Classification-based intrusion detection techniques can be trained from the network traffic data and proved to be very useful. The main weakness of the classification-based IDSs that are unable to identify novel attacks as well as they can not adapt with the network temporal changes.

On the other hand, clustering-based IDSs use an unsupervised learning mechanism to find interesting patterns in a network traffic data without prior knowledge about data labels. Through the learning process, the similarities between the data instances are measured to divide the data objects into different subsets called clusters. High quality clusters denote that the similarities within the same cluster and the dissimilarities between different clusters should be maximized. These techniques can identify new attacks and work with network changes.

Large-scale network traffic analysis applications are too large to be processed by

sequential methods. Traditional intrusion detection methods based on clustering do not scale well with larger sizes of network traffic and are computationally expensive in terms of memory. Furthermore, large-scale network traffic analysis provides a challenge in terms of performance while identifying the anomalies connections, and that is why a parallel algorithm is needed for detecting intrusions.

This chapter presents a parallel intrusion detection system (IDS-MRCPSO) based on the MapReduce framework since it has been confirmed as a good parallelization methodology for many applications. In addition, the proposed system incorporates clustering analysis to build the detection model by formulating the intrusion detection problem as an optimization problem. Furthermore, the proposed system has been tested on a real large-scale intrusion data set with different training subset sizes to show its speedup and scalability, and to present its detection quality.

3.2. Related Work

Anomaly-detection based intrusion detection systems work based on a profile of a normal network or system behavior using statistical or machine learning techniques. Anomaly-detection based on machine learning techniques can be categorized as either supervised or unsupervised depending on whether the class labels are known during the learning process or not. Several techniques have been proposed to tackle the intrusion detection problem using unsupervised algorithms like clustering-based algorithms.

We focus only on closely related work of unsupervised algorithms that were developed to solve anomaly detection. Also, we discuss one unsupervised parallel algorithm that was applied on anomaly detection systems.

Leung et al. in [42] proposed a density-based clustering algorithm by applying the frequent pattern tree on high dimensional data set. Their algorithm was applied on a one million records data set and achieved good detection rates, but it suffered

from high false positive rates.

However in [43], the same authors proposed an anomaly detection technique based on a K-means clustering algorithm. A technique to enhance the initial centers was proposed to avoid a shortcoming of sensitivity of the initial clusters in K-means clustering to enhance the clusters quality. The experiments were applied on 0.4% of the whole data set.

A fuzzy C-means intrusion detection algorithm was proposed in [44], where a weighting means for the degree of record membership with specific clusters was used. The algorithm was tested with five samples of random data, each sample having ten thousand records. The results showed high false positives rates with satisfactory detection rates.

Li et al. in [45] combined the K-means algorithm with the particle swarm optimization to build an intrusion detection system. The algorithm tried to benefit from the PSO characteristics to avoid premature convergence that K-means suffers from. The algorithm achieved relatively better results than the K-means algorithm.

Mazel et al. in [46] introduced an unsupervised approach to detect the network anomalies by combining the subspace clustering with inter-clustering result associations to mark the anomalies from the network traffic flow. The authors build an autonomous intrusion detection system to enhance the network protections against the intrusions. The system was tested with real network traffic and verified that the anomalies can be detected in the distributed network.

Gao et al. in [47] proposed a parallel clustering ensemble algorithm to speed the detection of intrusions in massive network traffic. Their algorithm was applied on a sample of data and achieved improved detection time with having satisfactory detection rate.

The technique proposed in this chapter is different from the techniques explained

above. All algorithms were examined with small sample sizes that were selected randomly from the complete training KDD intrusion data set [48], whereas our proposed technique is applied on the complete training data set for building the learning model.

In particular, we used the whole training data set because the testing data used does not come from the same distribution as the training data. Thus, random sampling affects the intrusion detection system because random sampling only reflects the distribution of a training data sample which leads to possible significant regions of the testing data being left out. For these reasons, using a larger training sample is likely to cover more significant regions and build a stronger detection model. Furthermore, the proposed system enables the use of larger amounts of data to build the detection model due to the parallelization, and this leads to higher detection rates.

As far as we know, our proposed system is the first work on the parallelization of intrusion detection systems using the MapReduce methodology, which is considered an alternative model for parallel processing over the MPI methodology [8].

3.3. Proposed Intrusion Detection System (IDS-MRCPSO)

Analyzing large network traffic data to detect intrusions takes a long time, thus, our proposed system uses the data clustering concept based on the PSO approach. PSO is parallelized using the MapReduce model as to scale with large-scale network traffic. The proposed intrusion detection system consists of three main components: preprocessing component, detector model construction component, and validation component. Figure 3.1 shows the proposed IDS architecture diagram.

The preprocessing component follows three consequent steps: missing value record elimination, categorical feature elimination, and data normalization. First we discard the records that have missing values because we use the records in the distance

equation of the clustering technique, and therefore, a record with a missing value is not usable in the equation.

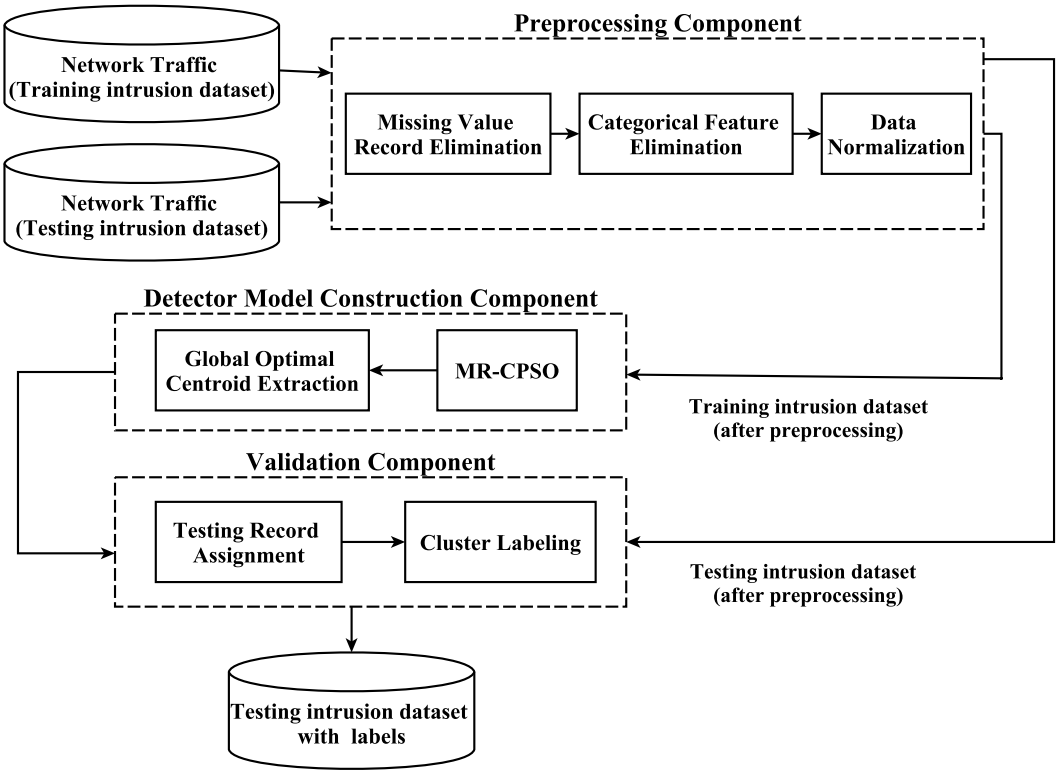


Figure 3.1: Proposed IDS-MRCPSO architecture diagram.

Then, eliminating the categorical features is done by removing any feature with categorical data. The purpose of this process is to use only numerical data in our distance calculation, because for the categorical data the distance calculations are difficult and depend on the data itself. Moreover, we can use the matching technique, but it will not help to distinguish the records in terms of the total distance.

At the end of the preprocessing stage, the normalization process normalizes the data set to avoid the bias problem some larger features values can cause. Furthermore, the normalization process is applied on the training and testing data sets at the same time, because applying normalization on training and testing data sets separately will create two different normalized data sets, since the minimum and maximum are based

on the input data. The normalization process is done using the following equation:

$$X_{ji_{new}} = \frac{X_{ji} - X_{i_{min}}}{X_{i_{max}} - X_{i_{min}}} \tag{3.1}$$

where X_{ji} is the value of record j for feature i; $X_{i_{min}}$ is the minimum value for feature i; $X_{i_{max}}$ is the maximum value for feature i.

The detector model construction stage starts by applying the MR-CPSO algorithm to the data results from the preprocessing stage, where only training data is used.

In chapter 2, we proposed a parallel PSO clustering (MR-CPSO) algorithm that is based on the MapReduce programming model. The experimental evaluation using large-scale data sets proved that MR-CPSO scales very well with increasing data set sizes and achieved reasonable clustering quality. The MR-CPSO algorithm expressed the clustering task as an optimization problem to find the global best solution. The MR-CPSO is a partitioning clustering algorithm which used individual centroids to represent different clusters. The initial centroids are selected randomly from the data set instances, then the centroids are updated iteratively based on the swarm particles' velocities until convergence to the global best centroid vector is acquired which then is used in the validation stage. The best centroid vector is evaluated based on the average minimum distances between the data instances and the selected cluster centroids.

The computational complexity of an intrusion detection system depends on the MR-CPSO algorithm used to generate optimal centroids from the training data. This algorithm has quadratic complexity because it requires computation of pairwise distances for all data set instances. In addition, the validation stage of the detection model is relatively fast, whereby it involves comparing testing data records with a small number of generated centroids.

In MR-CPSO, each particle keep some information which is used in the clustering task such as: centroid vector, velocity vector, fitness value...etc. The particle information is updated in each iteration using the information from the previous iteration. MR-CPSO is divided into three main modules: the first module is responsible for updating the particle's centroid vector, the second module is responsible to evaluate the fitness, and the third module is to merge the outputs of the first and second modules in order to generate a single new swarm.

In the first module, each particle gets an updated centroid vector in each iteration based on PSO movement equations. In this module an individual MapReduce job is triggered, where the *Map* function treats each particle as a Value and particle identification number as a Key. The *Map* function is started by retrieving the *Map* Value which contains all information about the particle. After that, the centroid vector is updated using the previous information based on the PSO equations. At the end, the *Map* function emits the particle with updated centroid vector to the *Reduce* function. The *Reduce* function sorts the *Map* intermediate output according the Key and merges them into one output intermediate file to be the input to the next module.

The second module of MR-CPSO presents another MapReduce job that evaluates the fitness function using the updated swarm particles. The fitness evaluation depends on measuring the distances between all data records and particle centroid vector. The fitness evaluation is calculated using the total sum of squares errors as the following equation:

$$Fitness = \frac{\sum_{j=1}^{k} \sum_{i=1}^{n_j} Distance(R_i, C_j)}{k} \tag{3.2}$$

where n_j denotes the number of records that belong to cluster j; R_i is the i^{th} record; k is the number of available centroids; $Distance(R_i, C_j)$ is the Euclidean distance

between record R_i and the centroid C_j.

The *Map* function in this module treats each record as a Value and each record's identification number as a Key. For each particle in the retrieved swarm, the *Map* function extracts the centroid vector from the particle and calculates the distance value between the *Map* Value (record) and the centroid vector, and then returns the centroid id which contains the minimum distance to emit them after that with particle id to the *Reduce* function.

The *Reduce* function task is to put the values with the same Key together and then calculates the summation of the minimum distances for each particle centroid. After that, the *Reduce* function emits the Key with total distances to use them as new centroid fitness values.

In the third module of MR-CPSO, the new single fitness value is calculated for each particle by summing centroids fitness values generated by the second module. Then, the previous swarm fitness values are changed to the new fitness values. After that, the best personal fitness and its centroids are modified based on a comparison between these values and the new fitness values. In addition, if there is any particle that has a fitness value smaller than the current global best fitness value, the global best fitness is assigned with new smaller fitness value as well as its centroid vector is updated. At the end, the new generated swarm is used for the next iteration.

After the detector model construction stage ends, we extract the global best centroid vector to be used as the detection model in the validation stage. In the validation stage, we used a different record subset called testing data set to evaluate the detection model by calculating the distances between the testing records and the global best centroids vector (detection model). After that, we assigned the testing records to the closest clusters based on the minimum distances. The pseudo-code of the testing records assignment procedure is shown in Algorithm 3.1.

Algorithm 3.1 Testing records assignment.

```
procedure CreateAssignment(model, testData, clustersNo)
    featuresNo=extractNoOfFeatures(testingData)
    recordsNo=extractNoOfRecords(testingData)
    assignmentVector= new Array(recordsNo)
    for i = 1 to recordsNo do
        dist = calcEuclidean(model[1], testData[i])
        minDist=dist
        cID=1
        for j = 2 to clustersNo do
            dist=calcEuclidean(model[j], testData[i])
            if dist ≤ minDist then
                minDist=dist
                cID=j
            end if
        end for
        assignmentCluster[i]=cID
    end for
    return assignmentVector
end procedure
```

Finally, the cluster labeling process is triggered to find the correct labels for output clusters generated from the testing record assignment step.

The assignment of cluster labels is accomplished by the maximum percentage of intersections between the original clusters of the testing data, and the clusters that are generated by applying the testing record assignment.

Figure 3.2 illustrates the cluster labeling process on an example, where the percentage of the normal records in A is $PN_A = \frac{Normal \cap A}{size(A)} = \frac{4}{6}$, and percentage of anomalous records in A is $PA_A = \frac{Anomalous \cap A}{size(A)} = \frac{2}{6}$; the maximum between these values is $\max(PN_A,\ PA_A) = \frac{4}{6}$; thus cluster A is the normal cluster. In the same way, for cluster B, the percentage of normal records is $PN_B = \frac{Normal \cap B}{size(B)} = \frac{2}{6}$, $PA_B = \frac{Anomalous \cap B}{size(B)} = \frac{4}{6}$ is the percentage of anomalous records, and $\max(PN_B,\ PA_B)=\frac{4}{6}$; therefore, cluster B is the anomalous cluster. For cluster C, $PN_C = \frac{1}{3}$, $PA_C = \frac{2}{3}$, and $\max(PN_C,\ PA_C) = \frac{2}{3}$; hence C is the anomalous cluster.

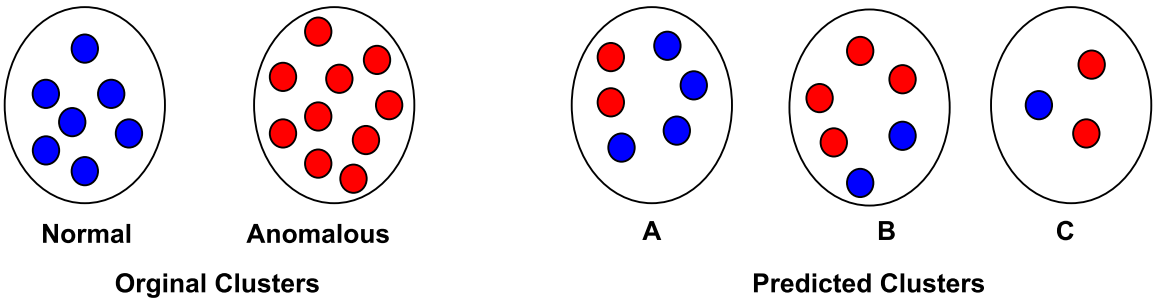


Figure 3.2: Clusters labeling process example.

3.4. Experiments and Results

In this section, we describe the evaluation of the proposed intrusion detection system in terms of detection quality. Furthermore, we discuss the running time and the system speedup of our proposed system.

3.4.1. Environment

We ran the experiments on the Longhorn Hadoop cluster hosted by the Texas Advanced Computing Center (TACC)[1]. The cluster contains 384 compute cores and 2.304 TB of aggregated memory and has 48 nodes containing 48GB of RAM, 8 Intel Nehalem cores (2.5GHz each). For our experiments, we used Hadoop version 0.20 for the MapReduce framework, and Java runtime 1.6 for the system implementation. Furthermore, in order to guarantee a constant level of parallelization, the maximum number of mapper and reducer tasks were set to 8 per node since each node contains 8 cores.

3.4.2. Data Set Description

To evaluate our proposed system, we used a big intrusion detection data set [48] that has never been fully analyzed by any standard data mining algorithms. It was used in 1999 as the benchmark at the Knowledge Discovery and Data Mining (KDD99)[2] competition, however, only portions of the data set were analyzed at a time. This data set contains a standard set of data to be audited that include a wide

[1]https://portal.longhorn.tacc.utexas.edu/
[2]http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

variety of intrusions simulated in a military network environment.

Each record in the data set represents a connection between two IP addresses, starting and ending at defined times and protocol. Every record is represented by 41 features, and each record represents a separate connection and is considered to be independent of any other record. The data is either identified as normal or as one of the 24 different types of attacks. The 24 attacks are grouped into four main types: probing, denial of service (DoS), unauthorized access from a remote machine (R2L), and unauthorized access to root (U2R).

The training data set contains (4,898,431) connection records which is collected during seven weeks of network traffic. Two weeks are used to produce approximately two million connection records as testing data set which is reduced to (311,029) corrected records which are used to evaluate the learning model. Furthermore, it is worth to note that the testing data set contains 7.5% unknown attack types which are not in the training data set.

A preprocessing process is applied on the training and testing data sets by discarding the records that have missing values and reducing the number of features to 38 features by discarding the 3 categorical features such as the protocol and service feature. Furthermore, the normalization process is applied on the training and testing data sets to convert them to normalized ones. In order to evaluate the impact of the training data set size in the detector model construction stage, we extracted 5 different samples from the whole training data set. In this chapter, we used the stratified sampling by randomly selecting the records from the original training data set with keeping the same ratio between the different classes.

In order to simplify the names of the training data set samples, the sample name consist of the specific format based on the percentage of the whole training data set. For example, the TRAIN20 sample consists of 20% of the whole training data set.

The 5 samples are described in Table 3.1.

Table 3.1: Data set samples.

Sample ID	Percentage (%)	Normal	Anomalies	Total
TRAIN20	20%	194,556	785,130	979,686
TRAIN40	40%	389,112	1,570,260	1,959,372
TRAIN60	60%	583,669	2,355,390	2,939,059
TRAIN80	80%	778,225	3,140,520	3,918,745
TRAIN100	100%	972,781	3,925,650	4,898,431

3.4.3. Evaluation Measures

We used the parallel Speedup [35, 49] measure calculated using Equation 3.3 to evaluate the performance of our proposed system. Speedup is measured by fixing the data set with increasing the number of cluster nodes. The speedup measure is calculated as:

$$Speedup = \frac{T_2}{T_n} \tag{3.3}$$

where T_2 is the running time using 2 nodes, and T_n is the running time using n nodes, where n is a multiple of 2.

For the intrusion detection quality, we used the True Positive Rate (TPR) or Detection Rate (DR) measure which is the ratio between the number of correctly detected attacks and the total number of attacks. Another measure used to evaluate intrusion detection systems is the False Positives Rate (FPR) or False Alarm Rate (FAR) which falsely identifies an intrusion detected that is not an intrusion. FPR is a ratio between the number of false positives and the total number of false positives plus the false negatives.

In addition, we evaluated the IDS's effectiveness by the Receiver Operating Characteristic (ROC) [50] curve which is a plot of the TPR against FPR. Therefore, we used the Area Under Curve (AUC) measure [50] as the ROC curve evaluation to combine the TPR and FPR, which is considered a good indicator of their relationship.

The AUC is calculated by the following equation:

$$AUC = \frac{(1 - FPR + TPR)}{2} \tag{3.4}$$

We used the PSO settings that are recommended by [37, 38]. We used a swarm size of 100 particles and adaptive inertia weight with maximum value W of 0.9. Also, we set the acceleration coefficient constants $cons_1$ and $cons_2$ to 1.49.

3.4.4. Results

To evaluate the effectiveness of the IDS-MRCPSO system, we run multiple experiments using different sizes of training data that are given in Table 3.1. In Table 3.2, we report the results of the proposed system based on TPR, FPR, and AUC for different training data set sizes. For this experiment, we set the number of PSO iterations to 50, and the number of clusters to 5 which were empirically determined. We observe that the TPR value of IDS-MRCPSO using the complete training data set (TRAIN100) achieves the best TPR and AUC value compared to other smaller training data sets. In addition, TRAIN100 obtains the lowest FPR results of all training data sets. For example, the IDS-MRCPSO system has a high TPR of 0.939 for TRAIN100, while it has a TPR of 0.903 for TRAIN20. For TRAIN100, the FPR value is 0.013, while for TRAIN20 the FRP is 0.038. The AUC value for TRAIN100 is 0.963 while the AUC value for TRAIN20 is 0.933. Hence, the results show our system can distinguish between the normal data records and anomaly records effectively. The results demonstrate that using larger training data, better results can be achieved.

Figure 3.3 shows the ROC curve using the proposed IDS-MRCPSO system. The figure shows that the best performance and high AUC value are achieved when using TRAIN-100 compared to the other curves.

Table 3.2: Proposed IDS-MRCPSO system results.

Sample ID	TPR	FPR	AUC
TRAIN20	0.903	0.038	0.933
TRAIN40	0.911	0.021	0.945
TRAIN60	0.927	0.015	0.956
TRAIN80	0.935	0.013	0.961
TRAIN100	0.939	0.013	0.963

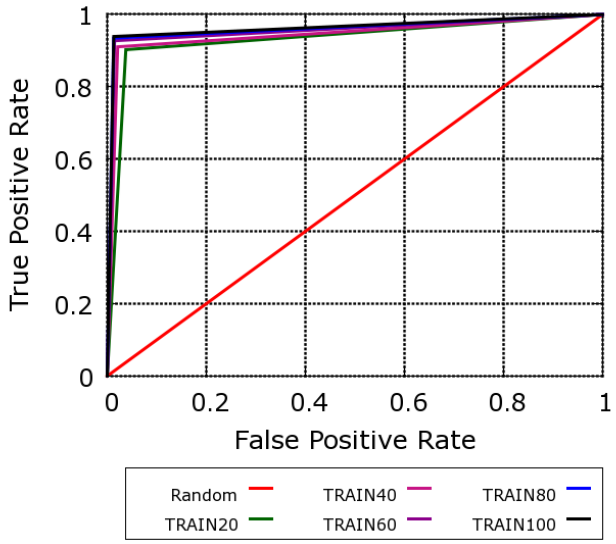
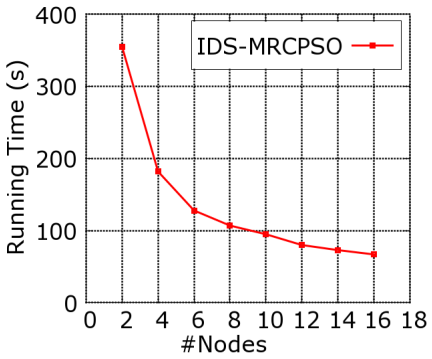


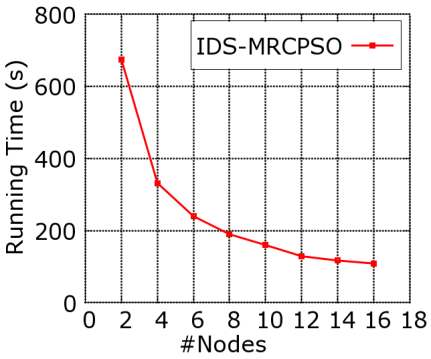
Figure 3.3: ROC results.

To check the scalability of the proposed system, we ran multiple experiments with different number of nodes. In each experiment, we report the running time and speedup on the average of 25 PSO iterations. The running times and speedup measures are shown in Figure 3.4, and Figure 3.5, respectively. In Figures 3.4(a)-3.4(e), the running time results are reported for the 5 training data sets for different number of Hadoop cluster nodes. All subfigures show that the running time improves faster for 2 nodes and 4 nodes than at the end when the number of nodes is 16 nodes. Furthermore, the impact of the training data set on the running time is well observed. The running time on 2 nodes takes 355, 675, 930, 1200 and 1875 seconds for TRAIN20, TRAIN40, TRAIN60, TRAIN80, and TRAIN100, respectively, while

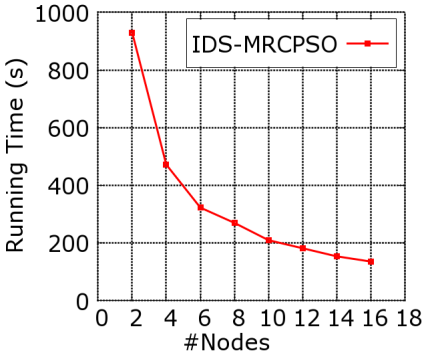
the running time on 16 nodes takes 67, 109, 136, 175 and 250 seconds for the same samples, respectively. As can be seen, the improvement factor of the running times for 16 nodes compared to the running time with 2 nodes are 5.30, 6.19, 6.83, 6.86, and 6.94, respectively.



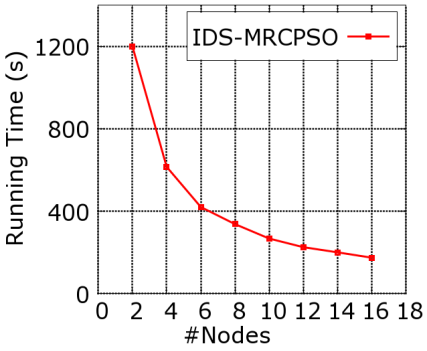
(a) TRAIN20 Running Time



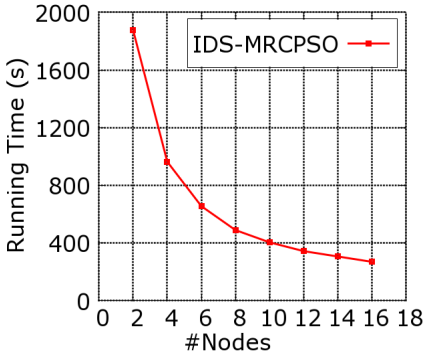
(b) TRAIN40 Running Time



(c) TRAIN60 Running Time

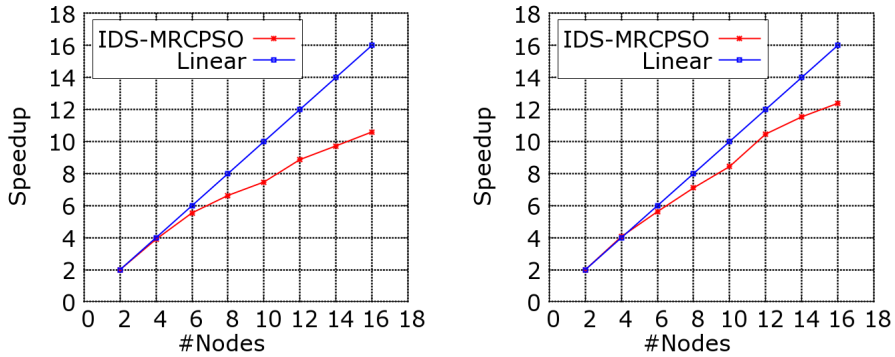


(d) TRAIN80 Running Time



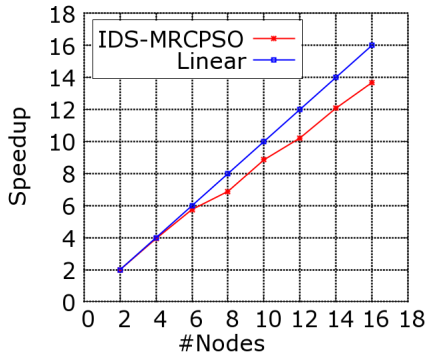
(e) TRAIN100 Running Time

Figure 3.4: 3.4(a)-3.4(e) Running time results for KDD data set samples from 20% to 100% sizes, respectively.

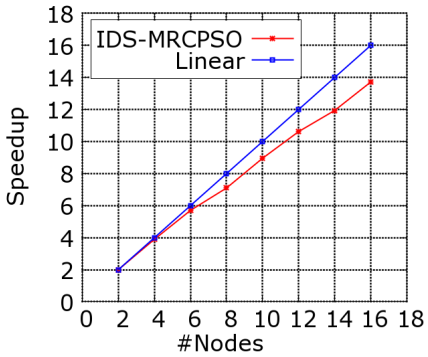


(a) TRAIN20 Speedup

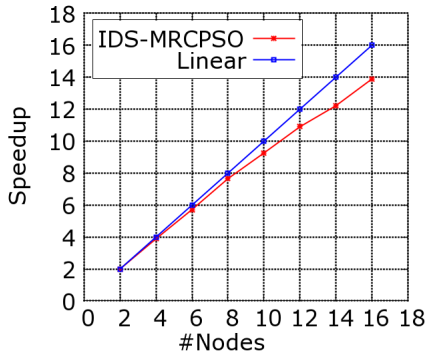
(b) TRAIN40 Speedup



(c) TRAIN60 Speedup



(d) TRAIN80 Speedup



(e) TRAIN100 Speedup

Figure 3.5: 3.5(a)-3.5(e) Speedup results for KDD data set samples from 20% to 100% sizes, respectively.

In Figures 3.5(a)-3.5(e), the speedup results using different training data set sizes with different numbers of nodes are shown. As can be observed from these figures, the speedup for TRAIN20 is very close to the linear speedup using 4, and 6 nodes. It begins to diverge from the linear speedup around 8 nodes that can be

attributed to the overhead of the Hadoop framework such as starting MapReduce jobs and storing intermediate outputs to the distributed file system.

The same trend is observed for TRAIN40, TRAIN60, TRAIN80 and TRAIN100. For TRAIN40, the speedup is very close to the linear one using 2 to 8 nodes, but it starts to diverge from the linear line with little difference compared to TRAIN20. For TRAIN60 and TRAIN80, the speedup is close to the linear one with 10 and 12 nodes, and it then starts to have a larger difference for larger numbers of nodes. We can summarize that the overhead of the Hadoop framework is reduced for larger data sets and the speedup is closer to the linear one. Furthermore, the speedup scales close to linear for most training data sets samples. The proposed system with TRAIN100 achieves a significant speedup getting very close to the linear speedup. The speedup results showed reasonable scalability for the proposed system.

3.5. Conclusion

In this chapter, we proposed an IDS-MRCPSO system for intrusion detection using the MapReduce methodology to solve the management of large-scale network traffic. We have shown that the intrusion detection system can be parallelized efficiently with the MapReduce methodology. Experiments were performed on a real intrusion data set in order to measure the system speedup. The experimental results reveal that IDS-MRCPSO is efficient with increasing training data set sizes, and scales very close to the optimal speedup by improving the detection results. Furthermore, we used the whole training data to build the detection model to avoid the random sampling effects, thus, this technique covers more significant regions of the training data set, and builds a stronger detection model. The results validate that using larger training data leads to better detection rates by keeping the false alarm very low.

CHAPTER 4. A MAPREDUCE BASED GLOWWORM SWARM OPTIMIZATION APPROACH FOR MULTIMODAL FUNCTIONS

In optimization problems, such as highly multimodal functions, many iterations involving complex function evaluations are required. Glowworm Swarm Optimization (GSO) has to be parallelized for such functions when large populations capturing the complete function space, are used. However, large-scale parallel algorithms must communicate efficiently, involve load balancing across all available computer nodes, and resolve parallelization problems such as the failure of nodes. In this chapter, we outline how GSO can be modeled based on the MapReduce parallel programming model. We describe MapReduce and present how GSO can be naturally expressed in this model, without having to explicitly handle the parallelization details. We use highly multimodal benchmark functions for evaluating our MR-GSO algorithm. Furthermore, we demonstrate that MR-GSO is appropriate for optimizing difficult evaluation functions, and show that high function peak capture rates are achieved. We show with the experiments that adding more nodes would help to solve larger problems without any modifications to the algorithm structure.

The remainder of this chapter is organized as follows: Section 4.1 presents the introduction to the mutimodal function optimization using GSO algorithm. Section 4.2 presents the related work in the area of parallel optimization algorithms. In Section 4.3, our proposed MR-GSO algorithm is introduced. Section 4.4 presents the experimental evaluation, and Section 4.5 presents our conclusions.

4.1. Introduction

The GSO algorithm has been used in many applications such as the hazard sensing in ubiquitous environments [51], mobile sensor network and robotics [14], because of its implementation simplicity and the need to tune a small number of

parameters [14, 51, 52]. Some functions such as multimodal functions are functions with many local maxima also referred to as peaks. Multimodal function optimization is not aiming to find the global maximum only, but rather all maxima based on some constraints. The peak count increases for high dimensional spaces, therefore, each function evaluation requires a long time to compute in order to find optimal target peaks at the end of the optimization process.

To optimize such functions, the number of individuals must be increased to share more local information for locating more peaks. To solve the high computation time in these situations, the algorithm must be parallelized in an efficient way to find the maxima in an acceptable amount of time.

Parallel algorithms suffer from a wide range of problems such as inefficient communication, or unfair load balancing, which makes the process of scaling the algorithms to large numbers of processors very difficult. Also, node failure affects the parallel algorithms, thus, reduce the algorithm's scalability. Therefore, any parallel algorithm developed should handle large amounts of data and scale well by increasing the compute nodes while maintaining high quality results.

In this chapter, the MapReduce methodology is utilized to create a parallel glowworm swarm optimization algorithm. The purpose of applying MapReduce to glowworm swarm optimization goes further than merely being a hardware utilization. Rather, a distributed model is developed, which achieves better solutions since it is scalable with a reduced overall computation time.

4.2. Related Work

The parallelization of optimization algorithms has received much attention to reduce the run time for solving large-scale problems [53, 54]. Parallel algorithms make use of multiple processing nodes in order to achieve a speedup as compared to running the sequential version of the algorithm on only one processor [35]. Many parallel

algorithms have been proposed to meet the difficulties of implementing optimization algorithms.

Many of the existing algorithms in the literature apply the (MPI) [8]. In [54], a parallel genetic algorithm was proposed using the MPI library on a Beowulf Linux Cluster with the master slave paradigm. In [53], an MPI based parallel particle swarm optimization algorithm was introduced. However, MPI is not the best choice for parallelization because of the weakness of having to handle the failure of nodes.

In [55], MRPSO incorporated the MapReduce model to parallelize particle swarm optimization by applying it on computationally data intensive tasks. The authors presented a radial basis function as the benchmark for evaluating their MRPSO approach, and verified that MRPSO is a good approach for optimizing data-intensive functions.

In [56], the authors made an extension of the genetic algorithm with the MapReduce model, and successfully proved that the genetic algorithm can be parallelized easier with the MapReduce methodology. In [57], the authors proposed a MapReduce based ant colony approach. They show how ant colony optimization can be modeled with the MapReduce framework. They designed and implemented their algorithm using Hadoop.

Comparing our proposed algorithm to the algorithms listed above, all MapReduce implementations were used to optimize single objective functions, whereas in our proposed algorithm, the algorithm searches for multiple maxima for difficult multimodal functions. To the best of our knowledge, MR-GSO is the first work on the parallelization of glowworm swarm optimization. Furthermore, it is the first work using the MapReduce methodology, which is considered an alternative model for parallel processing over the MPI methodology [8]. GSO can be naturally expressed with MapReduce, and therefore, easily be parallelized in order to be able to solve

computationally expensive multimodal functions with high dimensionality.

4.3. Proposed MapReduce GSO Algorithm (MR-GSO)

The grouping nature of glowworm swam optimization makes it an ideal candidate for parallelization. Based on the sequential procedure of glowworm optimization discussed in the first chapter, we can employ the MapReduce model. The MR-GSO consists of two main phases: Initialization phase, and MapReduce phase.

In the initialization phase, an initial glowworm swarm is created. For each glowworm i, a random position vector (X_i) is generated using uniform randomization within the given search space. Then, the objective function J is evaluated using the X_i vector. After that, the luciferin level (L_i) is calculated by Equation 1.3 using the initial luciferin level L_0, $J(X_i)$, and other given constants. The local decision range r_d is given an initial range r_0. After the swarm is updated with this information, the glowworms are stored in a file on the distributed file system as a <Key,Value> pair structure, where Key is a unique glowworm ID i and Value is the glowworm information. The initial stored file is used as input for the first MapReduce job in the MapReduce phase.

The representation structure of the <Key,Value> pairs are used in the MR-GSO algorithm as shown in Figure 4.1. The main glowworm components are delimited by semicolon, while the position X_i vector component is delimited by comma, where m is the number of dimensions used. In the second phase of MR-GSO, an iterative process of MapReduce jobs is performed where each MapReduce job represents an iteration in the glowworm swarm optimization. The result of each MapReduce job is an updated glowworm swarm with updated information, which is then used as the input for the next MapReduce job.

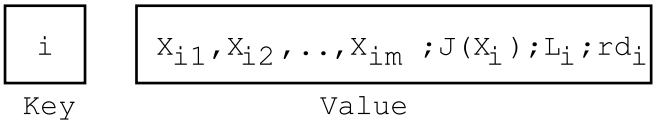


Figure 4.1: Glowworm representation structure.

In each MapReduce job, the algorithm focuses on the time consuming stages of the luciferin level update and glowworm movement to benefit from the power of the MapReduce model. In the movement stage, each glowworm i extracts the neighbor group $N_i(t)$ based on Equation 1.4, which requires distance calculations and luciferin level comparisons between each glowworm and other swarm members to locate the neighbor group. This process is executed N^2 times, where N is the swarm size. The neighbor group finding process is accomplished by the *Map* function that is part of a MapReduce job.

Before the neighbor group finding process is done in the *Map* function, a copy of the stored glowworm swarm ($TempSwarm$) is retrieved from the distributed file system, which is a feature provided by the MapReduce framework for storing files. In addition, the other information such as the GSO constants s, ρ, γ, β, nt, and rs that are used in the GSO movement equations, are retrieved from the job configuration file.

After that, the neighbor group finding process is started when the *Map* function receives <Key,Value> pairs from the MapReduce job driver, where Key is the glowworm ID i and the Value is the glowworm information. However, the *Map* function processes the Value by breaking it into the main glowworm components (X_i, $J(X_i)$, L_i, and rd_i), which are used inside the *Map* function. Then, a local iterative search is performed on $TempSwarm$ to locate the neighbor group using Equation 1.4. After that, the neighbor probability values are calculated based on Equation 1.5 to find the best neighbor using the roulette wheel selection method. At the end of the *Map* operation, the *Map* function emits the glowworm ID i with its Value and glowworm ID i with

the selected neighbor position vector (X_j) to the *Reduce* function. The *Map* function works as shown in Algorithm 4.1 outlining the pseudo code of the *Map* operation. As an intermediate step in the MapReduce job, the emited intermediate output from the mapper function is partitioned using the default partitioner by assigning the glowworms to the reducers based on their IDs using the modulus hash function.

Algorithm 4.1 Map function.

```
function Map (Key: GlowwormID, Value: Glowworm)
    glowwormID=Key
    glowwormValue=Value
    i = glowwormID
    extractInfo(X_i,J_i,L_i,rd_i)          ▷ Extract the information from the Glowworm
    read(TempSwarm)          ▷ Read the copy from the glowworm swarm from the
Distributed Cache
    for each glowworm j in TempSwarm do
        X_j=extractPosition(glowworm)
        L_j=extractluciferin(glowworm)
        EDist=returnEDistance(X_i,X_j)
        if EDist <rd_i and L_j >L_i then
            NeighborsGroup.add(j)
        end if
    end for
    if NeighborsGroup.size() <0 then
        for each glowworm j in NeighborsGroup do
            prob[j]=calculateProbability(i,j)   ▷ calculate the probabilities from the
NeighborsGroup using Equation 1.4
        end for
    end if
    nj=selectBestNeighbor(prob)                          ▷ using the roulette wheel
    X_j=extractPosition(nj)
    newValuenb=createValue(NeighborsGroup.size(),X_j)
    Emit(glowwormID, newValuenb)
    Emit(glowwormID, glowwormValue)
end function
```

The *Reduce* function in the MapReduce job is responsible for updating the luciferin level L_i which is considered the most expensive step in the glowworm optimization, since in this stage the objective function is evaluated for the new glowworm

position. The luciferin level updating process is started when the *Reduce* function receives <Key,ListofValues> pairs from the *Map* function where Key is the glowworm ID and the $ListofValues$ contains the glowworm value itself and its best neighbor position (X_j). The reduce function extracts the neighbor position vector (X_j) and glowworm information (X_i, $J(X_i)$, L_i, and rd_i). Then, the updating of the glowworm position vector is done using Equation 1.6. After that, the objective function is evaluated using the new glowworm position vector, and then the luciferin level is updated using Equation 1.3. Also, rd_i is updated using Equation 1.7. At the end, the *Reduce* function emits the glowworm ID i with new updated glowworm information. The pseudo-code of the *Reduce* function is shown in Algorithm 4.2.

Algorithm 4.2 Reduce function.

```
function Reduce (Key:glowwormID,ValList)
    glowwormID=Key
    i = glowwormID
    for each Value in ValList do
        if Value is the Neighbor case then
            extractInfo(X_j)        ▷ Extract the Neighbor information from the Value
            extractInfo(nbSize)                ▷ Extract the nbSize from the Value
        else
            glowworm_i=NULL
            extractInfo(X_i,J_i,L_i,rd_i)      ▷ Extract the information from the current
glowworm
            fill(glowworm_i,X_i,Jx_i,L_i,rd_i)
        end if
    end for
    newX=calculateNewX(X_i,X_j)    ▷ calculate the new position for glowworm i
using Equation 1.6
    newJx=calculateNewJx(newX) ▷ update luciferin level for glowworm i using
objective function formula J
    newL=calculateNewX(L_i,newJx)       ▷ update luciferin level for glowworm i
using Equation 1.3
    newrd=calculateNewrd(rd_i,nbSize)     ▷ calculate the new rd for glowworm i
using Equation 1.7
    glowworm_i.update(newX,newJx,newL,newrd)
    Emit(glowwormID, glowworm_i)
end function
```

At the end of the MapReduce job, the new glowworm swarm replaces the previous swarm in the distributed file system, which is used by the next MapReduce job.

4.4. Experiments and Results

In this section, we describe the optimization quality and discuss the running time of the measurements for our proposed algorithm. We focus on scalability in terms of speedup and the optimization quality.

4.4.1. Environment

We ran the MR-GSO experiments on the NDSU[2] Hadoop cluster. The NDSU Hadoop cluster consists of only 18 nodes containing 6GB of RAM, 4 Intel cores (2.67GHz each) with HDFS 2.86 TB aggregated capacity. For our experiments, we used Hadoop version 0.20 (new API) for the MapReduce framework, and Java runtime 1.6 to implement the MR-GSO algorithm.

4.4.2. Benchmark Functions

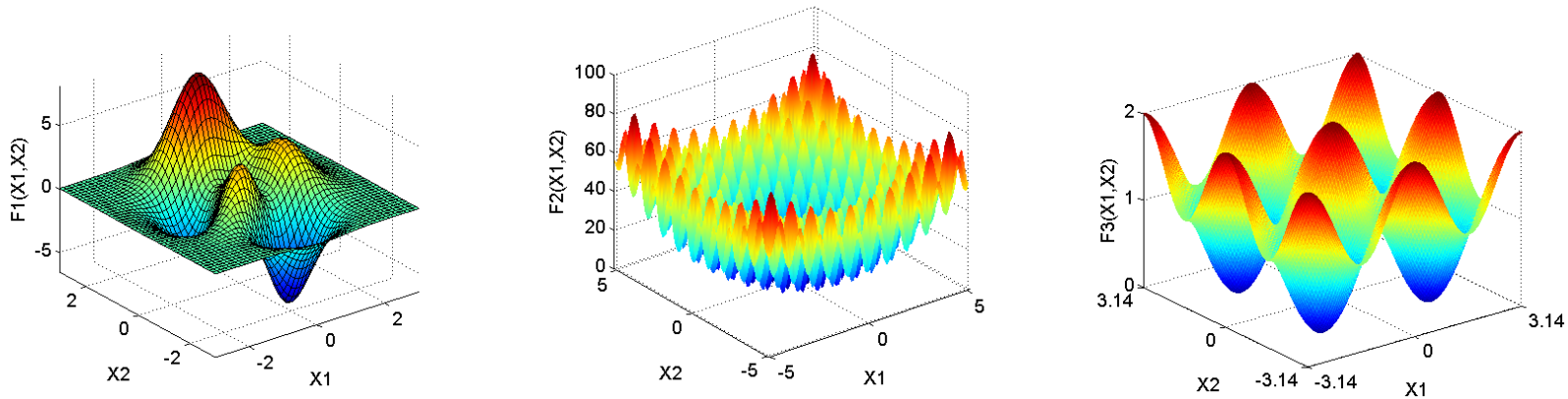
To evaluate our MR-GSO algorithm, we used three standard multimodal benchmark functions. The benchmark functions that are used are the following:

- F_1: the Peaks function in Equation 4.1 is a function of two variables, obtained by translating and scaling Gaussian distributions. It has multiple peaks which are located at (0,1.58), (0.46,0.63), and (1.28,0) with different peak function values. The function has the following definition:

$$\begin{aligned} F_1(X_1, X_2) = 3(1 - X_1)^2 e^{-[{X_1}^2 + (X_2+1)^2]} \\ -10(\frac{X_1}{5} - X_1^3 - X_2^5) e^{-[X_1^2 + X_2^2]} \\ -(\frac{1}{3}) e^{-[(X_1+1)^2 + X_2^2]} \end{aligned} \tag{4.1}$$

Figure 4.2(a) shows the Peaks function visualized for two dimensions.

[2]http://www.ndsu.edu



(a) F_1 (b) F_2 (c) F_3

Figure 4.2: Benchmark functions. 4.2(a) F_1: Peaks function. 4.2(b) F_2: Rastrigin function. 4.2(c) F_3: Equal-peaks-A function.

- F_2: the Rastrigin function is a highly multimodal function with the locations of the minima and maxima regularly distributed. This function presents a fairly difficult problem due to its large search space and its large number of local minima and maxima. We restricted the function to the hyphercube $-5.12 \leq X_i \leq 5.12, i = 1, ..., m$. The function has 100 peaks for 2 dimensions within the given range. The function has the following definition:

$$F_2(X_i) = 2m + \sum_{i=1}^{m} [X_i{}^2 - 10\cos(2X_i)] \tag{4.2}$$

Figure 4.2(b) shows the Rastrigin function visualized for two dimensions.

- F_3: the Equal-peaks-A function is a highly multimodal function in the m-dimensional search space. All local maxima of the Equal-peaks-A function have equal function values. The function search space ($-\pi \leq X_i \leq \pi$) is used, where, i = 1,...,m. The function has 3^m peaks such as for m=2 dimensions, the function has 9 peaks within the given range. We use the function F_3 to test higher dimensional search spaces such as m = 2, 3 and 4. The function has the following definition:

$$F_3(X_i) = \sum_{i=1}^{m} [\cos^2(X_i)] \tag{4.3}$$

Figure 4.2(c) shows the Equal-peaks-A function visualized for two dimensions.

4.4.3. Evaluation Measures

In our experiments, we used the parallel Speedup [35] measure to evaluate the performance of our MR-GSO algorithm, which is calculated using the following equation:

$$Speedup = \frac{T_2}{T_n} \tag{4.4}$$

where T_2 is the running time using 2 nodes, and T_n is the running time using n nodes,

where n is a multiple of 2.

The speedup is obtained by fixing the swarm size while increasing the number of cluster nodes to evaluate the algorithm's ability to scale with increasing numbers of the cluster nodes. For the optimization quality, we use the Peaks Captured Rate (PCR) and the average minimum distance to the peak locations (D_{avg}) [58]. The peak is considered captured if there are three glowworms near it with distance less than or equal ϵ. In this chapter, we used the distance $\epsilon = 0.05$ recommended in [58].

PCR is given by the following equation:

$$PCR = \frac{Number\ of\ Peaks\ Captured}{Number\ of\ All\ Peaks} \times 100\% \tag{4.5}$$

The average minimum distance to the peak locations D_{avg} is given by the following equation:

$$D_{avg} = \frac{1}{N} \times \sum_{i=1}^{N} \min_{\{1 \leq j \leq Q\}} \{\delta_{i1} ... \delta_{iQ}\} \tag{4.6}$$

where δ_{ij} is the Euclidean Distance between the location of glowworm X_i and S_j; X_i and S_j are the locations of glowworm i and peak j, respectively, and Q is the number of available peak locations; N is the number of glowworms in the swarm.

The best result for these measures will have a high PCR and low D_{avg}. For example, if we get a low D_{avg} and a low PCR, this means that the glowworms gathered in only a few peaks, and did not capture the other peaks. A high PCR, close to 100%, means that MR-GSO captured most of the peaks, whereas a low D_{avg}, close to zero, implies that all glowworms are close to the peaks, and thus, this ensures a gathering of the glowworms at the peak locations.

We used the GSO settings that are recommended in [52]. We used the luciferin decay constant $\rho = 0.4$; the luciferin enhancement constant $\gamma = 0.6$; the constant parameter $\beta = 0.08$; the parameter used to control the number of neighbors $nt = 5$;

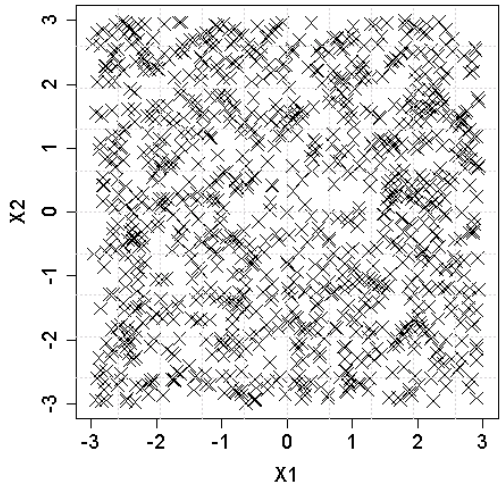
the initial luciferin rate $L_0 = 5.0$; the step size $s = 0.03$. In addition, the local decision range r_d and the radial sensor range r_s are problem based values. In our experiments, the local decision range rd is kept constant (rs=rd=r_0) throughout the optimization process. Preliminary experiments were done to decide whether to use an adaptive or constant rd. The constant rd achieved better results, since it ensures that the glowworm moves even if it has many neighbors. If there are many neighbors around, then the glowworm keeps moving towards the peaks, unlike the adaptive rd, where if the glowworm has many neighbors, it does not move and therefore, the new rd is 0 based on Equation 1.7. The best r_0 values for the given benchmark functions are chosen based on preliminary experiments.

4.4.4. Results

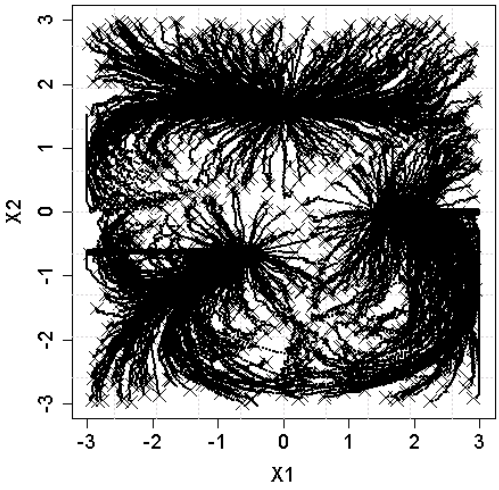
To evaluate the MR-GSO algorithm, the experiments are done measuring the PCR, D_{avg}, running time, and speedup for the mentioned benchmarks.

Figure 4.3 shows the MR-GSO optimization quality results visualized for the F_1 benchmark for two dimensions. Figure 4.3(a) shows the initial state of the glowworm swarm distributed in the search space using the random uniform distribution. The second part (Figure 4.3(b)) shows the glowworms during their movement towards the closest peak. At the end, all glowworms are gathered at the 3 peaks as shown in Figure 4.3(c). For this function, the results show that the MR-GSO algorithm is able to locate all 3 peaks, and therefore, achieves a PCR of 100%. In addition, the average minimum distance for the glowworms is very good with $D_{avg} = 0.0193$, which is fairly close to zero.

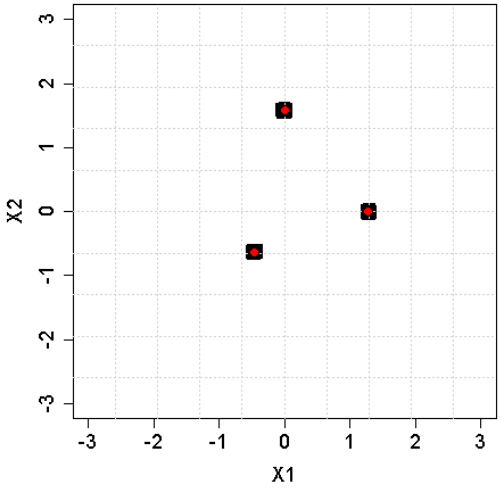
Figure 4.4 shows the MR-GSO optimization quality results visualized for the $F2$ benchmark for two dimensions. The results show that the MR-GSO algorithm is able to locate 96 out of 100 peaks (PCR=96%). Furthermore, D_{avg} is very low with a value of 0.031.



(a) Initial State

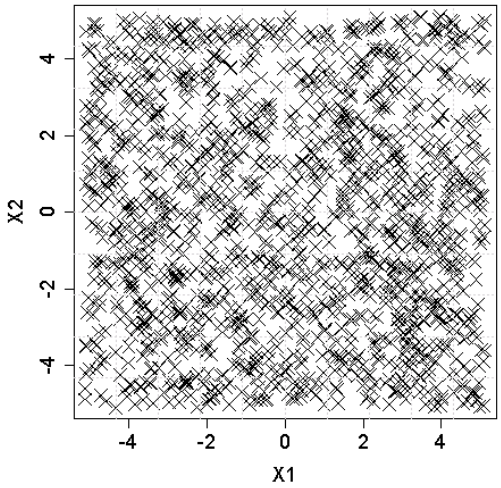


(b) Glowworms Movements

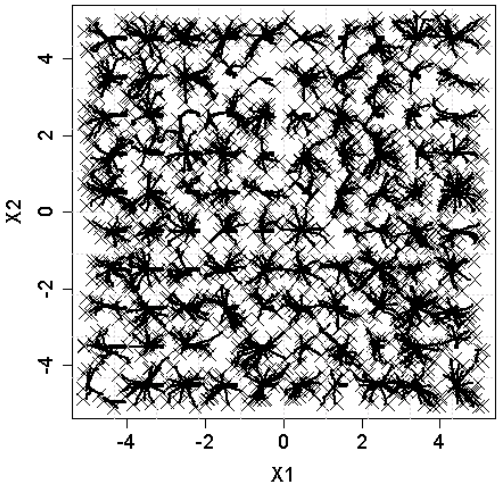


(c) Glowworms Final Locations with Peaks Locations

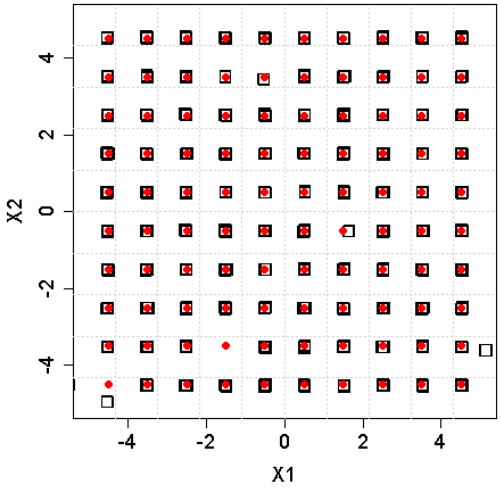
Figure 4.3: Optimization process for the peaks function (F_1) with swarm size= 1000, number of iterations=200, and r_0=1.0: the glowworms start from an initial random location and move to one of the function peaks. 4.3(a) The initial random glowworm locations. 4.3(b) The movements of the glowworms throughout the optimization process. 4.3(c) The final locations of glowworms (small squares) after the optimization process with the peak locations (red solid circles).



(a) Initial State



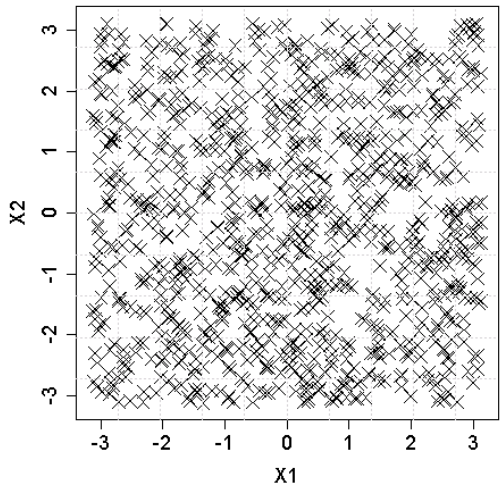
(b) Glowworms Movements



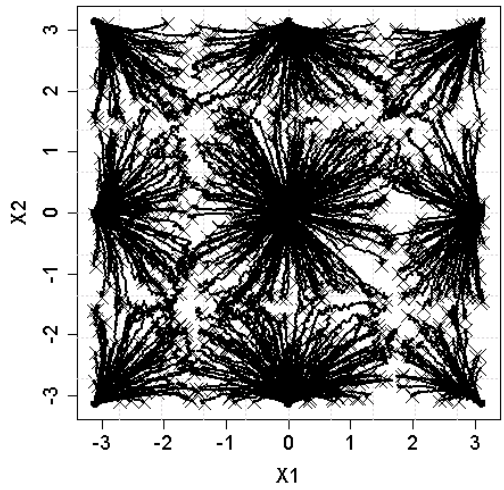
(c) Glowworms Final Locations with Peaks Locations

Figure 4.4: Optimization process for the Rastrigins function (F_2) with swarm size= 1000, number of iterations=200, and r_0=0.5: the glowworms start from an initial random location and move to one of the function peaks. 4.4(a) The initial random glowworm locations. 4.4(b) The movements of the glowworms throughout the optimization process. 4.4(c) The final locations of glowworms (small squares) after the optimization process with the peak locations (red solid circles).

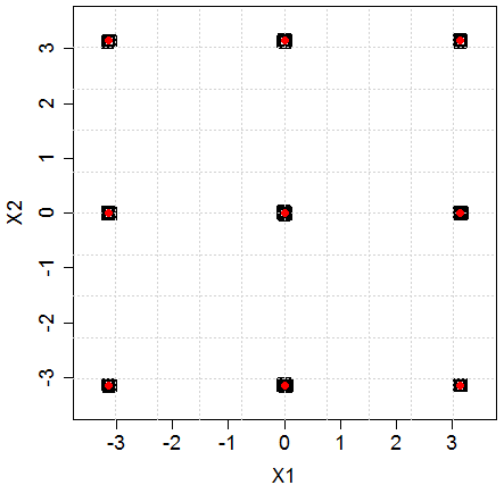
The results for the F_3 benchmark with two dimensions are given in Figure 4.5. The MR-GSO algorithm is able to locate all 9 peaks (PCR=100%) for this function. Also, a good value of D_{avg} is obtained with a low value equal to 0.017.



(a) Initial State



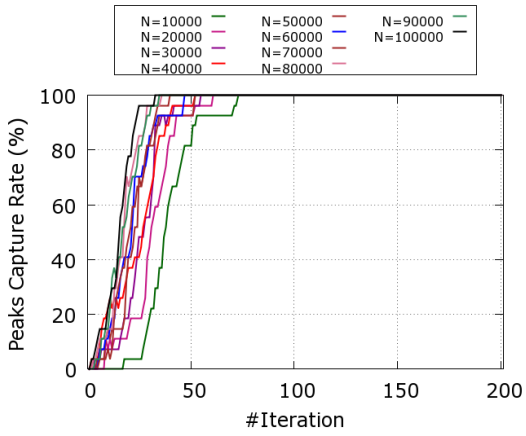
(b) Glowworms Movements



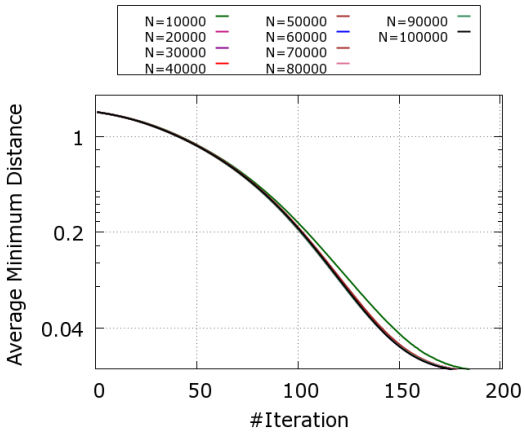
(c) Glowworms Final Locations with Peaks Locations

Figure 4.5: Optimization process for the Equal-peaks-A function (F_3) with swarm size=1500, number of iterations=200, and r_0=1.5: the glowworms start from an initial random location and move to one of the function peaks. 4.5(a) The initial random glowworm locations. 4.5(b) The movements of the glowworms through the optimization process. 4.5(c) The final locations of glowworms (small squares) after the optimization process with the peak locations (red solid circles).

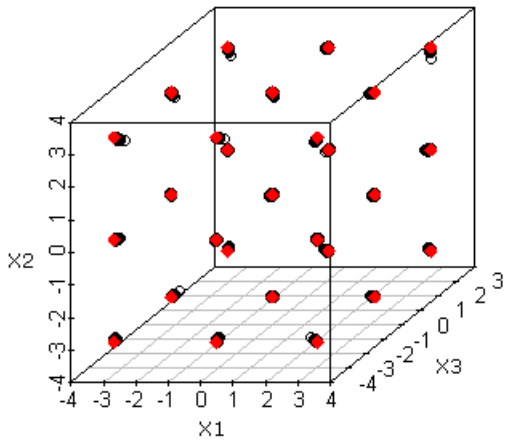
The optimization quality results for the F_3 function with three dimensions are shown in Figure 4.6.



(a) Peaks Capture Rate



(b) Average Minimum Distance



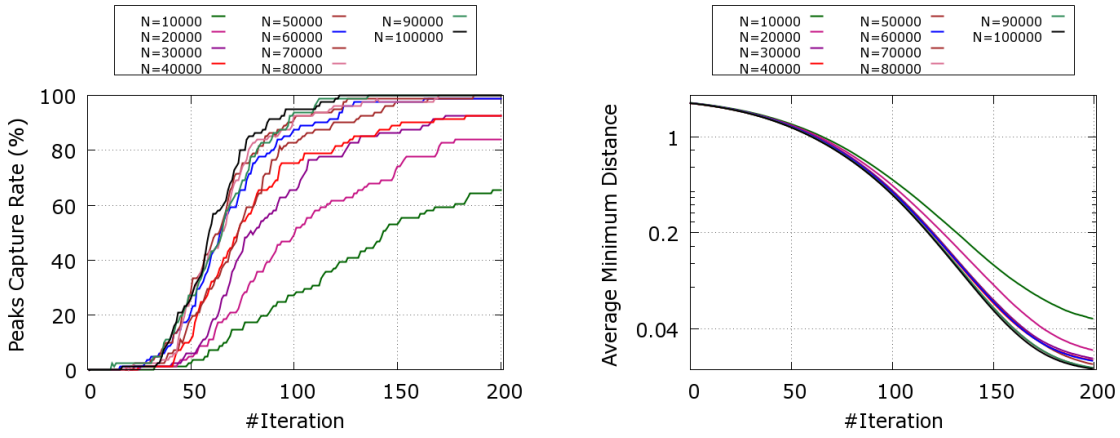
(c) Glowworms Final Locations with Peaks Locations

Figure 4.6: Optimization process for the Equal-peaks-A function (F_3) using 3-dimensions with 200 iterations, and r_0=2.0. 4.6(a) The Peaks capture rate for increasing swarm sizes. 4.6(b) The average minimum distance for increasing swarm sizes. 4.6(c) The final locations of glowworms (small squares) after the optimization process with peak locations (red solid circles).

The PCR and D_{avg} for each iteration using different numbers of swarm sizes (starting from 10,000 to 100,000) are presented. As can be noted from Figure 4.6(a), the PCR is improving for increasing swarm sizes. In addition, the number of iterations

needed to capture all peaks is reduced, such as, the PCR converges to 100% with a swarm size of 10,000 at iteration 73, while with a swarm size of 100,000 the PCR converges at iteration 36. Also, Figure 4.6(b) shows that the average minimum distance is improved when the swarm size is increased while maintaining low values for all swarm sizes. Figure 4.6(c) visualizes an example of the final glowworm location with peak locations for a swarm size of 30,000, where the PCR is 100% and D_{avg} is 0.0186.

The optimization quality results for the F_3 function with four dimensions are shown in Figure 4.7. Figure 4.7(a) clarifies the impact of the swarm size on the PCR. However, 200 iterations capture 65% of the peaks with a swarm size of 10,000, while with a swarm size of 100,000 the PCR converges to 100% at iteration 123. Also, Figure 4.7(b), using the log scale for the y axis, shows that larger swarm sizes give better average minimum distance results maintaining low values for all swarm sizes.

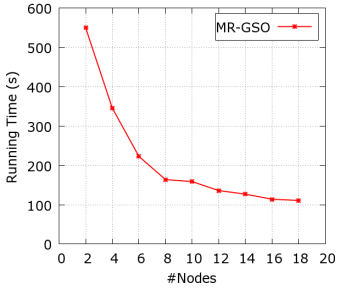


(a) Peaks Capture Rate (b) Average Minimum Distance

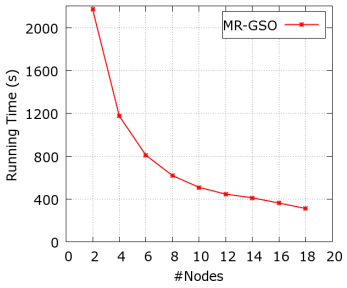
Figure 4.7: Optimization process for the Equal-peaks-A function (F_3) using 4-dimensions with 200 iterations, and r_0=2.0. 4.7(a) The Peaks capture rate. 4.7(b) The average minimum distance.

We ran MR-GSO with 18 cluster nodes by increasing the number of nodes in each run by multiples of 2. In each run, we report the running time and speedup (average of 25 iterations) of MR-GSO. The running times and speedup measures are

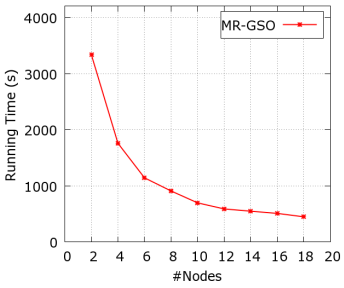
shown in Figure 4.8. Figures 4.8(a), 4.8(b) and 4.8(c) show the running times for the 3 swarm sizes of 100,000, 200,000, and 300,000 glowworms. As can be seen by all subfigures, the running time improves faster at the beginning than the end when increasing the number of nodes. Furthermore, the impact of the swarm size on the running time is well observed. Running the algorithms on 2 nodes takes 550, 2170, and 3335 seconds for 100,000, 200,000, and 300,000 glowworms, respectively.



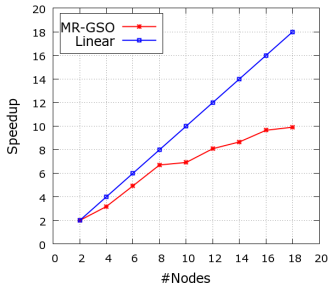
(a) Running Time with N=100,000



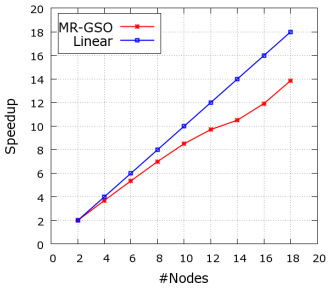
(b) Running Time with N=200,000



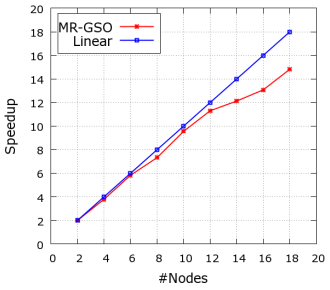
(c) Running Time with N=300,000



(d) Speedup with N=100,000



(e) Speedup with N=200,000



(f) Speedup with N=300,000

Figure 4.8: The running time and speedup results for the Equal-peaks-A function (F_3) with 4 dimensions. 4.8(a), 4.8(b) and 4.8(c): The Running time with N=100,000, N=200,000 and N=300,000, respectively. 4.8(d), 4.8(e) and 4.8(f): The Speedup with N=100,000, N=200,000 and N=300,000, respectively.

In Figures 4.8(d), 4.8(e) and 4.8(f), the speedup results using different swam sizes with different numbers of nodes are shown, highlighting the scalability of the algorithm. As can be inferred from the figures, the speedup for N=100,000 was very close to the linear speedup (optimal scaling) using 4, 6, and 8 nodes. It diverges from the linear speedup because of the overhead of the Hadoop framework, which

results from the management of starting MapReduce jobs, starting mappers and reducers operations, and serializing/deserializing intermediate outputs, and storing the outputs to the distributed file system.

The same behavior is observed for N=200,000 and N=300,000. For N=200,000 the speedup is very close to the linear one using 2 to 12 nodes, but then it diverges from the optimal line with a smaller difference compared to N=100,000. For N=300,000, the speedup is close to the linear one with 12 nodes, then it starts to have a larger difference for larger numbers of nodes, but comparing this difference with the one using N=200,000 and N=100,000 is much smaller. Therefore, we can conclude that the overhead of the Hadoop framework can be avoided when using larger numbers of swarm sizes, and thus the speedup is closer to the optimal one. In addition, the improvement factor of MR-GSO's running times for the swarm sizes of N=100,000, N=200,000 and N=300,000 are 4.95, 6.93, 7.41 respectively, compared to the running time using 2 nodes.

4.5. Conclusion

In this chapter, we proposed a scalable MR-GSO algorithm using the MapReduce parallel methodology to overcome the computational inefficiency of glowworm swarm optimization when difficult multimodal functions are to be optimized. Since large-scale parallel algorithms must communicate efficiently, balance the load across the available computer nodes, and resolve parallelization problems such as the failure of nodes, MapReduce was chosen since it provides all these features. We have shown that the glowworm swarm optimization algorithm can be successfully parallelized with the MapReduce methodology.

Experiments were conducted with three multimodal functions in order to measure the peak capture rate, average minimum distance to the peak locations, and the speedup of our algorithm. The peak capture rates obtained as well as the

average minimum distance values for the three benchmark functions are higher than the ones provided in previous experiments conducted without a parallel framework. This shows the benefit of the parallelization effect on the solution quality. Adding more glowworms reduces the number of iterations required and leads to an overall improvement in optimization quality. In addition, the scalability analysis revealed that MR-GSO scales very well with increasing swarm sizes, and scales very close to the linear speedup while maintaining optimization quality.

CHAPTER 5. A NEW CLUSTERING APPROACH BASED ON GLOWWORM SWARM OPTIMIZATION

High-quality clustering techniques are required for the effective analysis of the growing data. Clustering is a common data mining technique used to analyze homogeneous data instance groups based on their specifications. The clustering based nature-inspired optimization algorithms have received much attention as they have the ability to find better solutions for clustering analysis problems. Glowworm Swarm Optimization (GSO) is a recent nature-inspired optimization algorithm that simulates the behavior of the lighting worms. GSO algorithm is useful for a simultaneous search of multiple solutions, having different or equal objective function values.

In this chapter, a clustering based GSO is proposed (CGSO), where the GSO is adjusted to solve the data clustering problem to locate multiple optimal centroids based on the multimodal search capability of the GSO. The CGSO process ensures that the similarity between the cluster members is maximized and the similarity among members from different clusters is minimized. In addition, the proposed algorithm can discover the numbers of clusters without needing to provide the number in advance. Furthermore, three special fitness functions are proposed to evaluate the goodness of the GSO individuals in achieving high quality clusters. In addition, when the clustering is done for a big data, the calculation of the coverage and the intra-distance is very computationally expensive. This chapter also proposes MRCGSO, which is a parallel implementation of CGSO that enhances CGSO's scalability for large datasets. This is done by using the MapReduce methodology, where CGSO is formulated as Map and Reduce functions. The proposed algorithms are tested by artificial and real-world data sets. The improved performance of the CGSO algorithm compared to four popular clustering algorithms is demonstrated on most datasets. Furthermore, the speedup results reveal that MRCGSO can efficiently be used for

clustering large data sets.

The remainder of this chapter is organized as follows: Section 5.1 presents the related work in the area of clustering analysis based on nature-inspired optimization algorithms. In Section 5.2, our proposed CGSO and MRCGSO algorithms are introduced. Section 5.3 presents the experimental evaluation, and Section 5.4 presents our conclusions.

5.1. Related Work

Many clustering techniques are available in the literature [4, 11, 12] such as K-means [10], DBSCAN [11], Furthest First [59], and Learning Vector Quantization (LVQ) algorithm [60] for unsupervised clustering. Due to space constraints, we focus only on closely related work of clustering based nature-inspired optimization algorithms.

The clustering based nature-inspired optimization algorithms have received much attention to find better solutions for clustering analysis problems. The clustering problem in these algorithms is mapped to an optimization problem to locate the optimal solution based on different similarity metrics. Several clustering based nature-inspired optimization algorithms have been proposed to meet the challenges of clustering analysis problems.

In [61], the authors proposed a solution to the clustering analysis problem where the genetic algorithm capability was used. Their results showed that the genetic based clustering algorithm provides a good performance that is better than the K-means algorithm for different data sets. In [62], the Ant Colony Optimization (ACO) was used to perform the clustering analysis. The authors mainly formulated the problem by simulating the ant movement to group the data instances according to their similarity which is expressed by the available pheromone trails to guide ants to the optimal solutions. From their performance comparison results with other

stochastic algorithms, their results in terms of the quality were better than the other techniques.

Clustering algorithm based Particle Swarm Optimization (PSO) was introduced by Omran et al. in [19] to solve the image clustering problem. The results of their algorithm showed that PSO is applicable to solve the clustering problems. Another work applied PSO in clustering analysis, proposed in [33], where the problem discussed was document clustering. The authors compared their results with some state-of-the-art techniques, and concluded that the PSO algorithm is applicable to locate compact clusters.

Most of the existing nature-inspired optimization clustering algorithms locate the global solution for the given optimization problem, whereas our proposed algorithm locates multiple solutions, having different or equal objective function values. In addition, for most algorithms, the number of clusters as a parameter is required in advance to guide the clustering process. However, in several practical applications, the determination of the number of the clusters before exploring the data set is impossible. Some other nature-inspired algorithms have suffered from the slow convergence and, the clusters quality is low in particular when the data set is noisy. Furthermore, the authors faced some problems to produce well separated clusters. Our proposed algorithm in this chapter uses the modified GSO algorithm to solve the clustering analysis problem to tackle the slow convergence problem and the problem of determining the number of clusters in advance.

5.2. Proposed Algorithm

Our approach is a partitioning-based clustering which is motivated by the notion that instances are gathered around the centroids. K-means is one of the partitioning-based clustering techniques, where the centroids are extracted based on the weighted average of the data instances. The weighted average extraction method could be

efficient if the data set is divided into organized shaped clusters. However, it is not efficient if the data set contains arbitrary shaped clusters. In our proposed algorithm, we are formulating the clustering problem as a multimodal optimization problem to extract the centroids based on glowworms' movement.

The proposed algorithm partitions the given data set into sets of clusters, such that each glowworm in the swarm tries to cover larger numbers of data set instances. Furthermore, each glowworm moves toward the glowworms that cover a larger amount of data instances and has smaller distances between the data instances in the local region for that glowworm which is controlled by r_s. In the next subsections, we provide a formal description of the clustering problem as well as the core components used in our proposed algorithm. Then, we discuss the proposed clustering algorithm.

5.2.1. Preliminaries

The clustering algorithm is applied on data set, D, consisting of n instances with d-dimensions, each instance is represented by x_i, $i = 1...n$. Given D, a clustering algorithm tries to extract a set of clusters $C = \{C_1, C_2, ..., C_k\}$, each is represented with a point called centroid, such as $c = \{c_1, c_2, ..., c_k\}$, where k is the number of centroids in the c centroid set. Furthermore, the clustering algorithm tries to maximize the similarity of the instances in the same cluster, and to minimize the similarity of instances from different clusters. In addition, each cluster should have at least one instance assigned to it, and the different clusters should be disjoint such that $\bigcap_{i..k} C_i = \{\}$ and $\bigcup_{i..k} C_i = D$, which ensures that there is no empty cluster. The Sum Squared Errors (SSE) fraction is calculated using the following equation:

$$SSE = \sum_{j=1}^{k} \sum_{i=1}^{|C_j|} (Distance(x_i, c_j))^2 \tag{5.1}$$

Another fraction is used in our proposed algorithm called Inter-Distance, which is

calculated by the following equation:

$$InterDist = \sum_{i=1}^{k} \sum_{j=i}^{k} (Distance(c_i, c_j))^2 \tag{5.2}$$

In this chapter, we use the Euclidean distance to calculate the $Distance$.

The swarm S used in the GSO optimization process consists of m glowworms, where each glowworm is represented by a vector, g_j, $j = 1...m$. Each g_j has 5 components: luciferin level (L_j), fitness function value (F_j), d-dimensional position vector (p_j), coverage set (cr_j) which is the set of the data instances that are covered by g_j, and intra-distance ($intraD_j$) between the cr_j set members and g_j position. The g_j should cover at least one data instance in its local range. The local range is a constant and is equal to the radial sensor range r_s, which is the same for all glowworms in swarm S. Furthermore, keeping the local range constant throughout the clustering process ensures that a glowworm keeps moving towards the optimal glowworms in all cases, even if it does not have neighbors or if it has many neighbors around.

5.2.2. Clustering based GSO Algorithm - CGSO

In recent years, GSO has been proven to be effective to solve optimization problems [52]. In data clustering, it can be formulated as an optimization problem that finds the optimal centroids of the clusters rather than to find optimal data partitions. The strength of optimization motivates us to apply GSO for finding the optimal solutions for the clustering problems. GSO is distinguished from other optimization techniques (that locate one local or global optimal solution), by finding multiple optimal solutions. The found solutions either have equal values for the dedicated objective function or not.

In our proposed clustering algorithm CGSO, the GSO objective is adjusted to locate multiple optimal centroids such that each centroid represents a sub-solution and

the combination of these sub-solutions formulate the global solution for the clustering problem. The proposed CGSO consists of four main phases: initialization phase, luciferin level update, glowworm movement, and candidate centroids set construction.

In the initialization phase, first an initial glowworm swarm S of size m is created. For each glowworm g_j, a random position vector (p_i) is generated using uniform randomization within the given search space within the minimum and the maximum values that are calculated from the data set D. Then, the luciferin level (L_j) is initialized using the initial luciferin level L_0. The fitness function value F_j is initialized to zero. The local range r_s is set to an initial constant range r_0. Secondly, after initializing the swarm, the set of data instances cr_j which are covered by g_j, is extracted from data set D, and the $intraD_j$ is calculated using the following equation:

$$intraD_j = \sum_{i=1}^{|cr_j|} Distance(cr_{ji}, g_j) \tag{5.3}$$

where cr_{ji} is the data instance i which is covered by g_j; $|cr_j|$ is the number of data instances which is covered by g_j. Then, in the last step of the initialization phase, the swarm-level fractions SSE and $InterDist$ are calculated.

To initialize SSE, we extract the glowworms list that covered the highest number of data instances (the glowworms have the maximum $|cr_j|$ sizes). These glowworms should be disjointed from each other, where each glowworm is located in a different region (the distance between any pair of these glowworms should be greater than r_s). The extracted glowworm list is considered the initial set of the candidate centroid c. After that, the candidate centroid set c is used to calculate the initial SSE using Equation 5.1. The same initial set c is also used to calculate the $InterDist$ which is calculated by Equation 5.2. After the initialization phase, an iterative process is performed to find optimal glowworms that represent the clustering problem centroids.

The result of each iteration is an updated swarm with updated candidate centroids set c. In the luciferin level update phase, firstly, the fitness function F is evaluated to assign new F_j values for each glowworm using the glowworm position and other information.

Three different fitness functions are proposed to evaluate the goodness of the glowworm. For all proposed fitness functions, each glowworm tries to maximize the coverage percentage from the data instances $|cr_j|$ by keeping the intra-distances $intraD_j$ among the covered data instances and the glowworm as minimum. Furthermore, we used normalized fractions for the $|cr_j|$ and $intraD_j$ by dividing the total number of data instances n and $\max_j(intraD_j)$, respectively, to avoid the biased state between the two fractions. The fitness functions are different from each other depending on the swarm-level fractions (SSE, and $InterDist$) that are used. The first fitness function is given by the following equation:

$$F1(g_j) = \frac{\frac{1}{n}|cr_j|}{SSE \times \frac{intraD_j}{\max_j\ (intraD_j)}} \tag{5.4}$$

In $F1(g_j)$, and beside the purpose of maximizing $|cr_j|$ and minimizing $intraD_j$, we incorporate SSE swarm-level fraction to be minimized between the candidate centroids set as a whole. The second fitness function is given by the following equation:

$$F2(g_j) = \frac{InterDist \times \frac{1}{n}|cr_j|}{\frac{intraD_j}{\max_j\ (intraD_j)}} \tag{5.5}$$

In $F2(g_j)$, we incorporate $InterDist$ swarm-level fraction in the process, and this fraction should be maximized among the candidate centroids set c. The third fitness

function is given by the following equation:

$$F3(g_j) = \frac{InterDist \times \frac{1}{n}|cr_j|}{SSE \times \frac{intraD_j}{\max\limits_{j}\ (intraD_j)}} \tag{5.6}$$

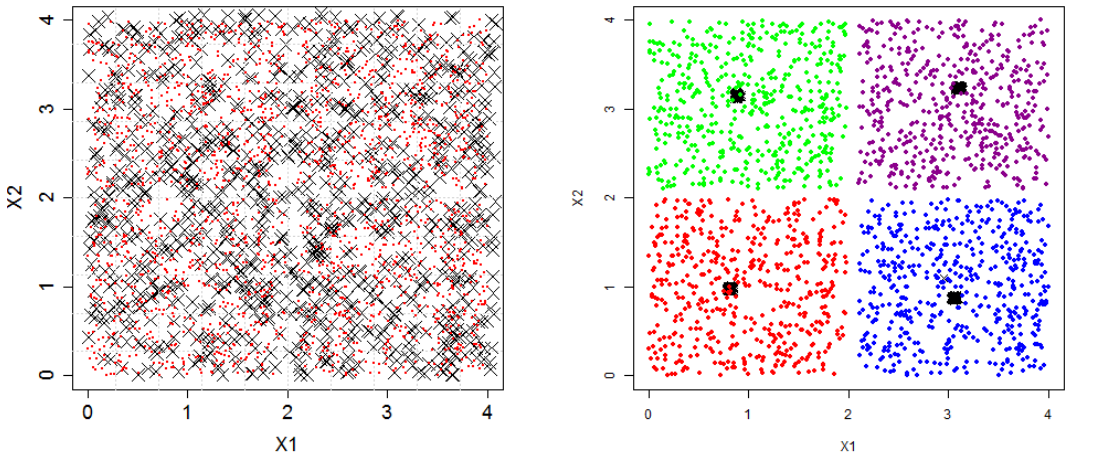
In $F3(g_j)$, a combination between maximization of the $InterDist$ and minimization of the SSE fractions is added to the $F3(g_j)$ at the same time. After the fitness function evaluation for glowworm g_j, the luciferin level L_j is updated using Equation 1.3. Then, each glowworm g_j locates the neighborhood group using Equation 1.4, and the neighbor probability values are calculated based on Equation 1.5 to find the best neighbor using the roulette wheel selection method. Then, the glowworm is moved towards the best neighbor by updating its p_j vector by Equation 1.6 using the best neighbor position. After that, $|cr_j|$, and $intraD_j$ are updated based on the new glowworm g_j positions.

The candidate centroid set c is reconstructed based on the highest fitness values (F_j), and not like the way they are extracted during the initialization phase, which is based on the highest number of data instances (the glowworms have the maximum $|cr_j|$). The same rule is used during the internalization phase to construct candidate centroid set c, where all c members should be disjoint from each other such as each glowworm should locate in different regions and the distance between the glowworm pairs should be greater than range r_s. After that, the candidate centroid set c is used to calculate the new value for SSE which is calculated by Equation 5.1. In addition, the same candidate centroids set c is also used to calculate $InterDist$, which is calculated by Equation 5.2. Then, the fitness function is reevaluated using the new information. The iterative process is continued until the size of the candidate centroid set c becomes less than a specific threshold (minimum number of centroids is given), or the maximum number of iterations is achieved. The candidate centroid

set c decreases throughout the iterative process, and after the clustering process is completed, the candidate centroid set is used to evaluate the clustering results.

5.2.3. Illustrative Example

Figure 5.1 shows an illustrative example of the CGSO clustering algorithm process to visualize the clustering results. An artificial data set with 2 dimensional instances is generated with 4 balanced clusters such as each cluster formulates a square. Figure 5.1(a) shows the initial swarm state distributed in the search space using the random uniform distribution and the scattered artificial data set in the same search space. At the end, all glowworms are gathered at the 4 optimal centroids as shown in Figure 5.1(b).



(a) Initial State

(b) Glowworms Final Locations with Peaks Locations

Figure 5.1: Clustering process for the artificial data set with swarm size=1,000, maximum number of iterations=200, and r_s=1.2: the glowworms start from an initial random location and move to one of the centroids. 5.1(a) The initial random glowworm locations (small black crosses) with data set instances (red points). 5.1(b) The final locations of glowworms (small squares) after the clustering process with 4 centroids, each cluster in the data set has a different color based on the minimum distances to the centroid.

5.2.4. Proposed MRCGSO Approach

CGSO solves the clustering problem by obtaining the best solution based on coverage of the centroids plus the intra-distance of each cluster. In CGSO, each glowworm competes to be a centroid and tries to cover the largest amount of data records, which means the highest coverage with the minimum intra-distance. Therefore, when the clustering is done for big data, the calculation of the coverage and the intra-distance is very computationally expensive. MRCGSO, solves this issue by parallelizing CGSO. This is done by using the MapReduce methodology, where CGSO is formulated as Map and Reduce functions.

In MRCGSO, Each glowworm g_j in the swarm has the following information:

- Luciferin level (L_j)
- Fitness function value (F_j)
- M-dimensional position vector (p_j)
- Coverage set size (cr_j): the number of the data instances that are covered by g_j
- Intra-distance ($intraD_j$): the distance between the cr_j set members and g_j position
- Local decision range (r_d): the range of the glowworm to find the covered data records inside this range, and to find the glowworm's neighbors.

This information is updated in each iteration based on the previous swarm state. MRCGSO consists of three main steps, which are: initialization, coverage and distance computations, fitness evaluation, and swarm movement. The initialization step includes a random initialization for the swarm members, and assigns each glowworm a position between the minimum and maximum values of the search space based on the data set. Moreover, each glowworm has to cover at least one data record, thus

ensuring that the generated position of the glowworm is not an outlier, and that there are no empty candidate clusters.

The map and reduce functions are implemented for the coverage and distance step since these are based on the data records and therefore, this is time consuming step of the algorithm. After initialization, the swarm is written into the distributed cache. The mapper accesses the HDFS to read the data set, which is partitioned based on the number of mappers used. Each mapper retrieves the swarm from the distributed cache and then calculates the sub-covered records and the sub-intra-distances for each glowworm of the swarm based on the data chunk it is associated with. After that, the mapper emits the result of each glowworm in the <key, value> format, where the value components are delimited by semicolons: <glowworm-id , sub-coverage ; sub-intra-distance>. The sub-coverage is the number of data records covered by the glowworm, and the sub-intra-distance in the summation of the distances between the glowworm and the covered data records.

The reducer receives the intermediate output from the mappers and calculates the sum of sub-coverages and sub-intra-distances from all mappers in order to find the glowworm's cumulative coverage and cumulative intra-distance for the whole data set. Then, the reducer emits the results in the <key, value> format: <glowworm-id , coverage ; intra-distance>, where the coverage is the number of data instances covered by the glowworm from the whole data set, and the intra-distance is the sum of distances between the glowworm and the covered data instances of the whole data set. The reducer's output is saved on the HDFS. The number of mappers and reducers are multiples and based on the data set size and the available hardware resources. The last step in MRCGSO is the fitness evaluation and swarm movement where the reducers' outputs are extracted from the HDFS, and then the fitness of each glowworm is calculated. After that, the same CGSO process is applied for updating

the glowworm position based on its glowworm neighbors.

5.3. Experiments and Results

This section presents a performance analysis to investigate the efficiency of the CGSO algorithm in the data clustering. We present the results obtained using the CGSO algorithm on well-known data sets to conduct a reliable comparison. In addition, the proposed MRCGSO algorithm has been tested on large scale synthetic data sets with different sizes to show its speedup and scalability..

Furthermore, the comparisons between the three introduced fitness functions are presented to show the algorithm's robustness. In addition, we present the comparison of the CGSO with other four well-known clustering algorithms: K-Means clustering [10], average linkage agglomerative Hierarchical Clustering (HC) [12], Furthest First (FF) [59], and Learning Vector Quantization (LVQ) [60], which have been used in the literature and we analyze their performance. Finally, the time complexity and algorithm convergence are discussed.

5.3.1. Environment

We ran the CGSO experiments on a PC containing 6GB of RAM, 4 Intel cores (2.67GHz each). The MRCGSO experiments were run on the Longhorn Hadoop cluster hosted by the Texas Advanced Computing Center (TACC)[1]. For our experiments, we used Hadoop version 0.20 (Java-based API) for the MRCGSO implementation, and Java runtime 1.6 to implement the CGSO algorithm. Furthermore, the WEKA [63] open source tool is used for comparisons.

[1]https://portal.longhorn.tacc.utexas.edu/

5.3.2. Data Sets Description

We present the results obtained using the CGSO on 7 typical data sets which are used in the literature. The first 5 data sets are obtained from the UCI database repository[1]. Furthermore, we used 2 artificial data sets from ELKI[2], and use them to visualize the clustering results. To evaluate the scalability of MRCGSO, series of synthetic data sets with different sizes of records were generated using the data generator developed in [34]. All data sets are described in Table 5.1.

Table 5.1: Summary of the data sets.

Data set	**#Records**	**#Features**	**#Clusters**	**Type**
Iris	150	4	2	*Real*
Ecoli	327	7	5	*Real*
Glass	214	9	6	*Real*
Balance	625	4	3	*Real*
Seed	210	7	3	*Real*
Mouse	490	2	3	*Artificial*
VaryDensity	150	2	3	*Artificial*
F2m2d5c	$2,000,000$	5	2	*Synthetic*
F4m2d5c	$4,000,000$	5	2	*Synthetic*
F8m2d5c	$8,000,000$	5	2	*Synthetic*
F16m2d5c	$16,000,000$	5	2	*Synthetic*

The data sets considered in this chapter are described as follows:

- Iris: this data set contains 3 clusters of 50 records each, where each cluster refers to a type of iris plant. One cluster is linearly separable from the other 2, but the other are overlapped and not linearly separable from each other.

- Ecoli: this data set contains 5 clusters, where each cluster represents different cellular localization sites of E.coli proteins.

- Glass: this data set describes the material concentrations in glasses, where each

[1]http://archive.ics.uci.edu/ml/index.html
[2]http://elki.dbs.ifi.lmu.de/wiki/DataSets

cluster denotes the type of the glass.

- Balance: this data set was generated to model psychological experimental results in three different clusters, where each cluster contains different balance scale direction.

- Seed: this data set represents 3 clusters based on the measurements of geometrical properties of wheat, where each cluster contains different types of wheat.

- Mouse: contains point coordinates in 2 dimensions that represents mouse head. The data set contains 3 gaussian clusters, where the left cluster represents left ear, the right cluster represents the right ear, and the middle cluster represents the mouse face.

- VaryDensity: contains point coordinates in 2 dimensions. The data set contains 3 gaussian clusters with variable density.

- Synthetic: Four data sets ranging from 2 million to 16 million data records are generated. The data sets' names consist of the specific pattern such as for example: the F2m2d5c data set consists of 2 million (2m) data records, each record is in 2 dimensions (2d), and the data set is distributed into 5 clusters (5c).

5.3.3. Evaluation Measures

In our experiments, we use two different measures for the evaluation of the cluster quality: entropy and purity [36]. The purity and the entropy are the standard measures of the clustering quality. Entropy measures how the various semantic classes are distributed within each cluster, and is calculated by the following equation:

$$Entropy = \sum_{j=1}^{k} \frac{\mid C_j \mid}{n} E(C_j) \tag{5.7}$$

where C_j contains all data instances assigned to cluster j by the clustering algorithm, n is the number of data instances in the data set, k is the number of clusters that is generated from the clustering process, and $E(C_j)$ is the individual entropy of cluster C_j which is defined by the following equation:

$$E(C_j) = -\frac{1}{log\ q} \sum_{i=1}^{q} \frac{\mid C_j \cap L_i \mid}{\mid C_j \mid}\ log\ (\frac{\mid C_j \cap L_i \mid}{\mid C_j \mid}) \tag{5.8}$$

where L_i denotes the true assignments of the data instances in cluster i; q is the number of actual clusters in the data set. Similarly to the previous equation, the purity of the clustering is defined as:

$$Purity = \frac{1}{n} \sum_{j=1}^{k} \max_{i}(\mid L_i \cap C_j \mid) \tag{5.9}$$

Smaller entropy values and larger purity values indicate better clustering solutions. The clustering quality is perfect if clusters only contain data instances from one true cluster; in that case the purity and entropy equal 1 and 0, respectively.

In addition, we used the speedup [35] measure calculated using Equation 5.10 to evaluate the scalability performance of the MRCGSO version.

$$Speedup = \frac{T_2}{T_n} \tag{5.10}$$

where T_2 is the running time using 2 nodes, and T_n is the running time using n nodes, where n is a multiple of 2.

We used the GSO settings that are recommended in [52]. We used $\rho = 0.4$; $\gamma = 0.6$; the initial luciferin level $L_0 = 5.0$; the step size $s = 0.03$. The swarm size used in our experiments is equal to 1000 glowworms and the maximum number of iterations is set to 200. Furthermore, since radial sensor range (local range) r_s depends on the data set, preliminary experiments were conducted by varying the r_s values in

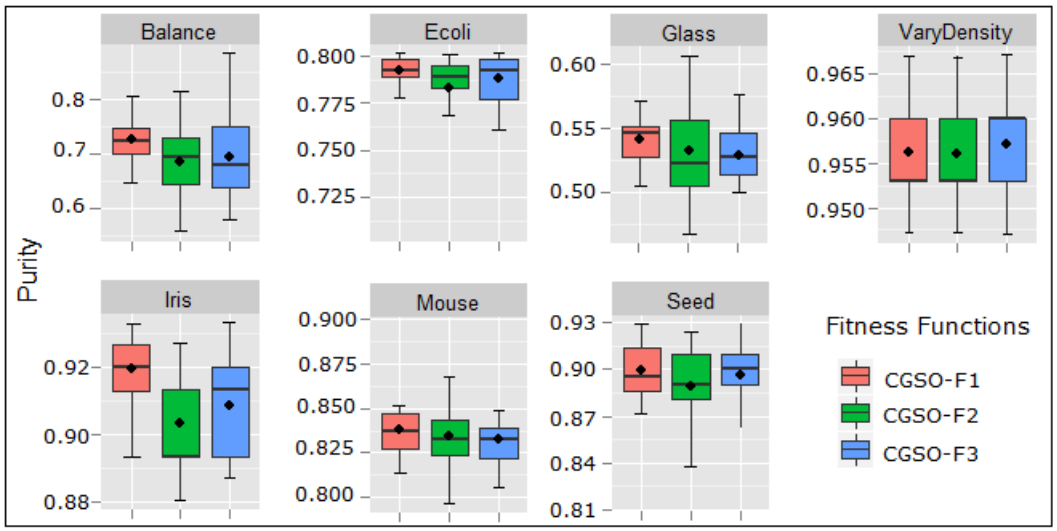
order to choose the best r_s value for each individual data set. The best r_s values that were empirically determined for the Iris, Ecoli, Glass, Balance, Seed, Mouse, and VaryDensity data sets are 1.35, 0.38, 0.38, 0.48, 0.052, and 0.06, respectively.

5.3.4. Results

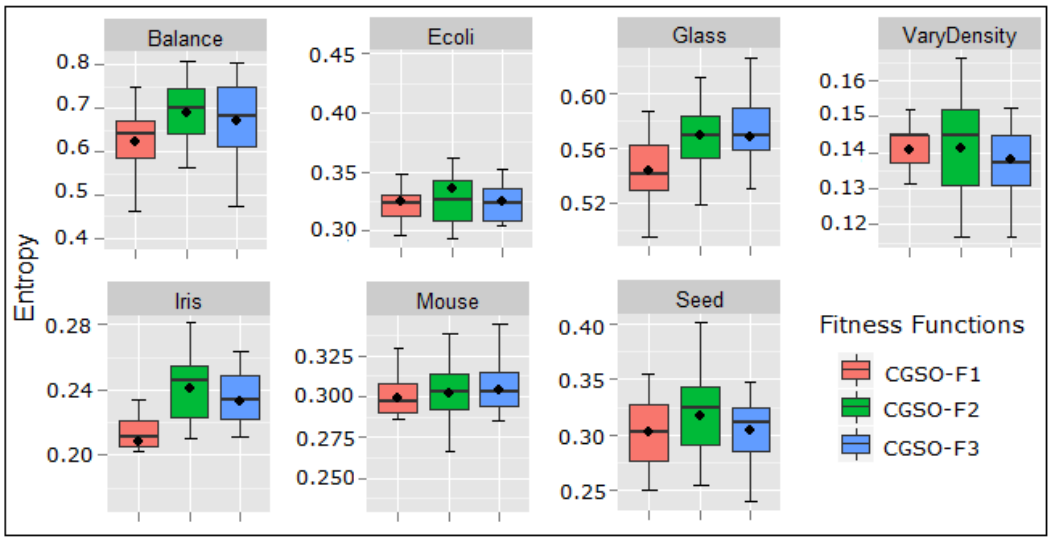
This section presents the comparison among the three proposed fitness functions to evaluate the impact of these functions in our proposed CGSO algorithm. In addition, comparisons with other well-known clustering methods are proposed. In order to abbreviate our proposed algorithm variants which are based on different fitness functions, a specific format is used to distinguish them, such that CGSO-F1, CGSO-F2, CGSO-F3 are our proposed algorithm using F1, F2, and F3, respectively. Furthermore, scalability analysis in terms of speedup for the MRCGSO is discussed in this section.

To evaluate the impact of the fitness functions in our proposed CGSO algorithm, we compared the three variants (CGSO-F1, CGSO-F2, and CGSO-F3) to show the algorithm flexibility and robustness. The purity and entropy results distribution for applying CGSO on the given data sets are shown as the box plots in Figure 5.2.

It can be seen from Figure 5.2(a) that the highest average purity (25 independent runs) is produced using F1 for all data sets (however, it is not statistically significant). Furthermore, it can be noted from Figure 5.2(b), F1 achieved the minimum average entropy for the Iris, Glass, Balance, Seed data sets (however, again not statistically significant). F2 obtained the minimum average entropy for the Ecoli and VaryDensity data sets.



(a) Purity



(b) Entropy

Figure 5.2: Box plots of the purity and entropy results obtained by comparing three different fitness functions (F1, F2, and F3) with different data sets. The small solid circles represent the average of 25 runs, and the bar inside the rectangle shows the median; minimum and maximum values are represented by whiskers below and above the box.

A comparison of the clustering quality in terms of purity and entropy with other clustering methods are shown in Tables 5.2 and 5.3, respectively. For our proposed algorithm, the average and the standard deviation of the purity and entropy results for 25 independent runs for each of the three fitness functions as well as the best

results (within brackets) are presented in Tables 5.2 and 5.3. The highest purity and smallest entropy values in each case are shown in bold. It can been seen from the Table 5.2, CGSO-F1 outperforms all other clustering techniques for most data sets with an average purity of 0.919, 0.792, 0.541, 0.726, 0.900, and 0.956 for Iris, Ecoli, Glass, Balance, Seed, VaryDensity, respectively. The HC obtained the best purity for the Mouse data set (0.91), however, its result was not much different compared to the result achieved by CGSO.

For the entropy results in Table 5.3, where a smaller entropy implies a better result, CGSO-F1 shows competitive performance and outperforms other clustering techniques for most data sets with an average entropy of 0.209, 0.543, 0.622, and 0.302 for Iris, Glass, Balance, and Seed, respectively. The HC obtained the best entropy for the Mouse data set (0.165). The K-Means obtained the best entropy for the Ecoli data set (0.307). Furthermore, CGSO-F3 obtained the best entropy for the VaryDensity data set.

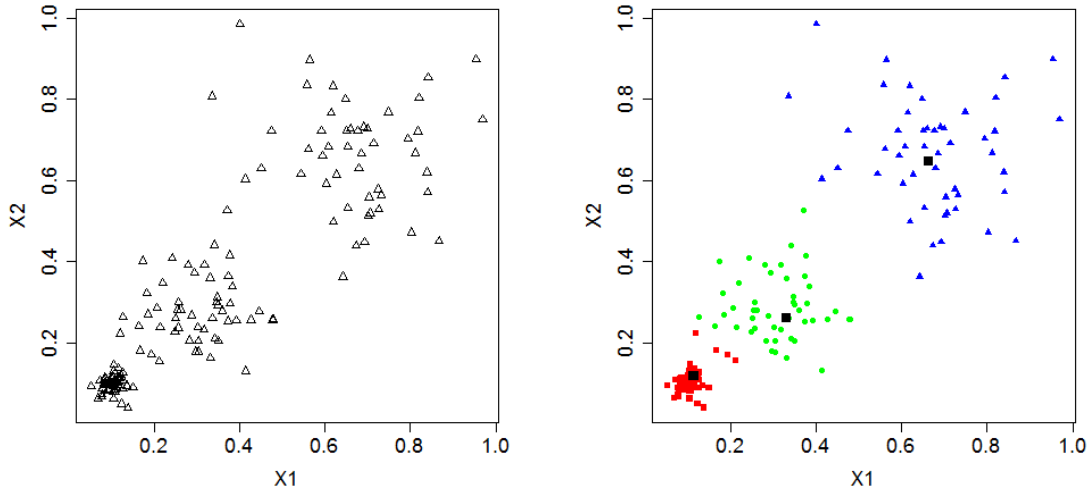
Figures 5.3 and 5.4 show the visualization of clustering quality results (best run is selected from the highest function results) of the VaryDensity and Mouse data sets, respectively. Figure 5.3(b) shows that the clustering quality results of CGSO-F1 (has best results among the three functions), and Figure 5.3(c) shows the clustering quality results of K-means. It can been seen that CGSO-F1 is able to assign the data instances to the correct cluster, with highest purity of 0.963, while K-means' purity result is 0.953. Figure 5.4(b) shows the clustering quality results of CGSO-F3 (has best results among the three functions), and Figure 5.4(c) shows the clustering quality results of K-means. It can be noted that CGSO-F3 is able to assign the data instances to the correct cluster, with highest purity 0.896, while K-means purity results is 0.827.

Table 5.2: Purity results.

Data set	**CGSO-F1**	**CGSO-F2**	**CGSO-F3**	**K-Means**	**HC**	**FF**	**LVQ**
Iris	**0.919** ± 0.090 [0.933]	0.903 ± 0.014 [0.927]	0.909 ± 0.012 [0.933]	0.887	0.887	0.860	0.507
Ecoli	**0.792** ± 0.006 [0.801]	0.779 ± 0.029 [0.801]	0.789 ± 0.012 [0.801]	0.774	0.654	0.599	0.654
Glass	0.541 ± 0.018 [0.570]	0.533 ± 0.036 [0.607]	0.529 ± 0.020 [0.575]	**0.542**	0.463	0.481	0.411
Balance	**0.726** ± 0.039 [0.805]	0.685 ± 0.061 [0.810]	0.694 ± 0.074 [0.882]	0.659	0.632	0.653	0.619
Seed	**0.900** ± 0.016 [0.929]	0.889 ± 0.026 [0.924]	0.897 ± 0.018 [0.929]	0.876	0.895	0.667	0.667
Mouse	0.837 ± 0.013 [0.880]	0.834 ± 0.018 [0.876]	0.833 ± 0.018 [0.896]	0.827	**0.910**	0.800	0.843
VaryDensity	**0.956** ± 0.006 [0.967]	0.956 ± 0.007 [0.967]	0.957 ± 0.006 [0.967]	0.953	0.667	0.667	0.567

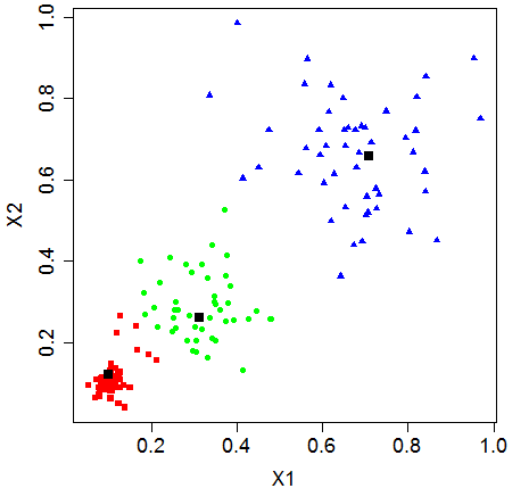
Table 5.3: Entropy results.

Data set	**CGSO-F1**	**CGSO-F2**	**CGSO-F3**	**K-Means**	**HC**	**FF**	**LVQ**
Iris	**0.209** ± 0.018 [0.170]	0.241 ± 0.020 [0.210]	0.233 ± 0.018 [0.176]	0.264	0.230	0.307	0.790
Ecoli	0.325 ± 0.013 [0.295]	0.342 ± 0.050 [0.293]	0.324 ± 0.014 [0.305]	**0.307**	0.522	0.611	0.579
Glass	**0.543** ± 0.023 [0.495]	0.569 ± 0.022 [0.519]	0.568 ± 0.030 [0.507]	0.567	0.662	0.646	0.754
Balance	**0.622** ± 0.078 [0.446]	0.690 ± 0.068 [0.560]	0.669 ± 0.099 [0.395]	0.701	0.739	0.654	0.753
Seed	**0.302** ± 0.031 [0.250]	0.317 ± 0.039 [0.253]	0.305 ± 0.027 [0.239]	0.327	0.298	0.537	0.577
Mouse	0.299 ± 0.015 [0.253]	0.302 ± 0.021 [0.248]	0.304 ± 0.020 [0.234]	0.319	**0.165**	0.351	0.262
VaryDensity	0.141 ± 0.013 [0.116]	0.141 ± 0.017 [0.116]	**0.138** ± 0.016 [0.116]	0.145	0.421	0.466	0.728



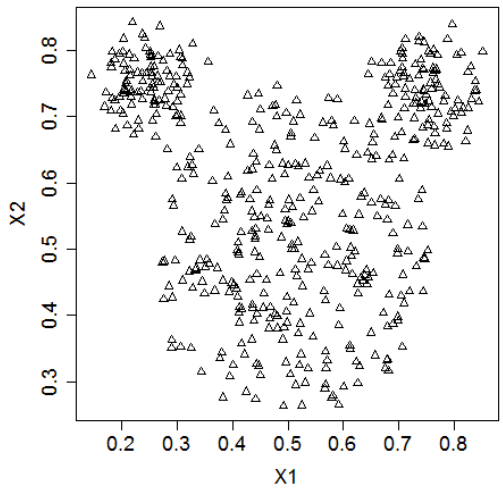
(a) Original Data set

(b) Clustering with CGSO-F1

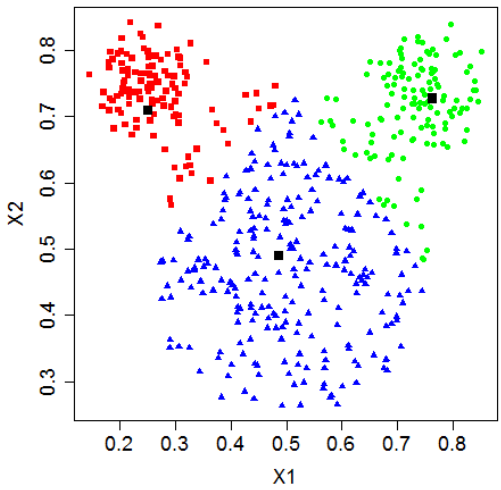


(c) Clustering with K-means

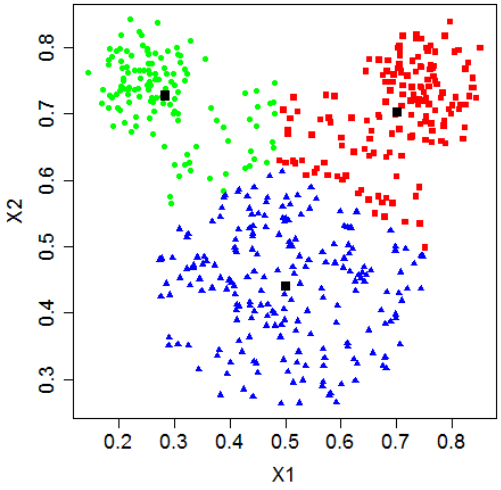
Figure 5.3: Clustering results for the VaryDensity data set, where the black boxes represent the centroids. 5.3(a) The original data set. 5.3(b) The clustering results with CGSO using fitness function F1. 5.3(c) The clustering results with K-means.



(a) Original Data set



(b) Clustering with CGSO-F3

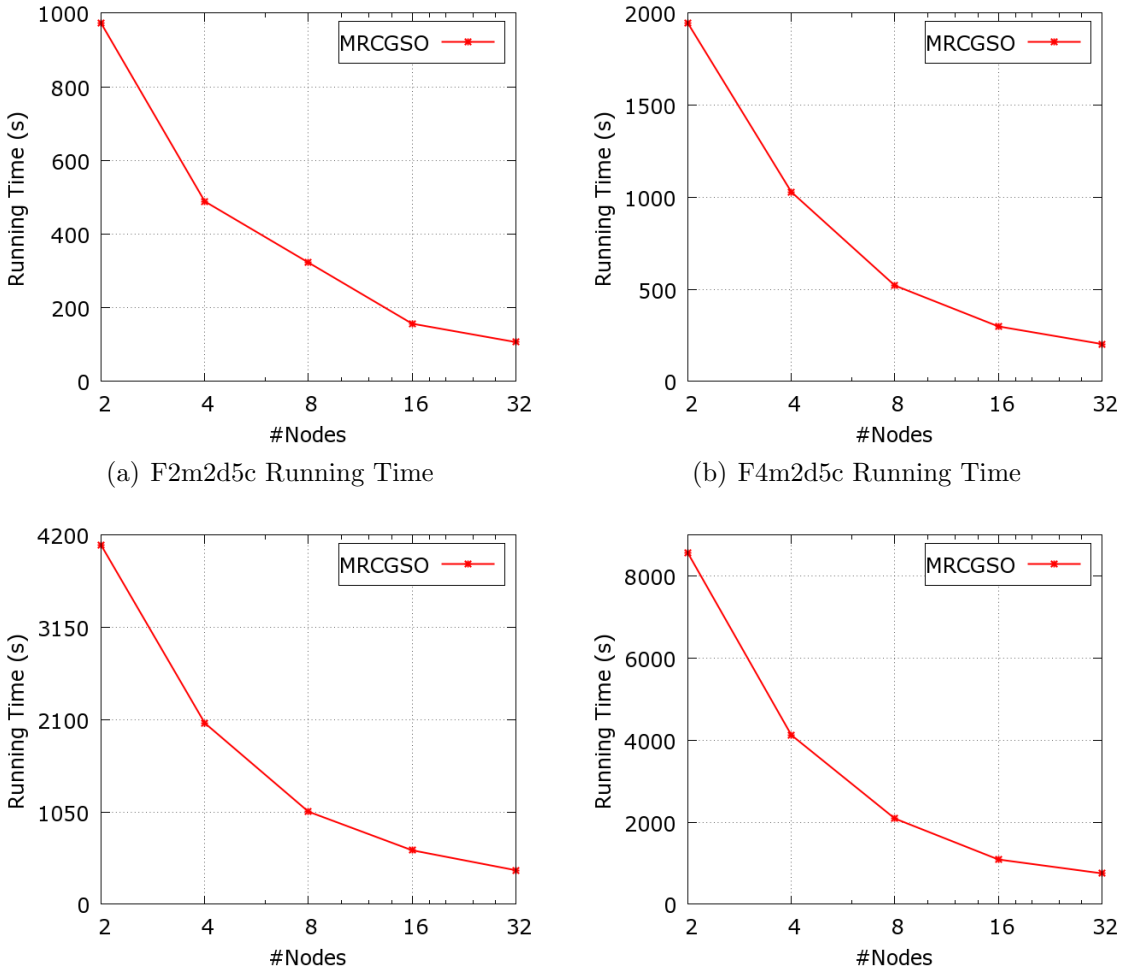


(c) Clustering with K-means

Figure 5.4: Clustering results for the Mouse data set, where the black boxes represent the centroids. 5.4(a) The original Mouse data set. 5.4(b) The clustering results with CGSO using fitness function F3. 5.4(c) The clustering results with K-means.

The running times and speedup measures for MRCGSO are shown in Figures 5.5 and 5.6, respectively. As can be noted from Figure 5.5, the improvement factor of MRCGSO's running times for the F2m2d5c, F4m2d5c, F8m2d5c, F16m2d5c data sets using 32 nodes are 9.09, 9.66, 10.54, 11.35, respectively, compared to the running time with 2 nodes. The MRCGSO algorithm demonstrates a significant improvement in

running time. Furthermore, the running time of MRCGSO decreases almost linearly with increasing numbers of nodes of the Hadoop cluster. In addition, the MRCGSO speedup in the Figure 5.6 scales close to linear for most data sets. The MRCGSO algorithm with F16m2d5c achieves a signicant speedup that is very close to the linear speedup.



(a) F2m2d5c Running Time

(b) F4m2d5c Running Time

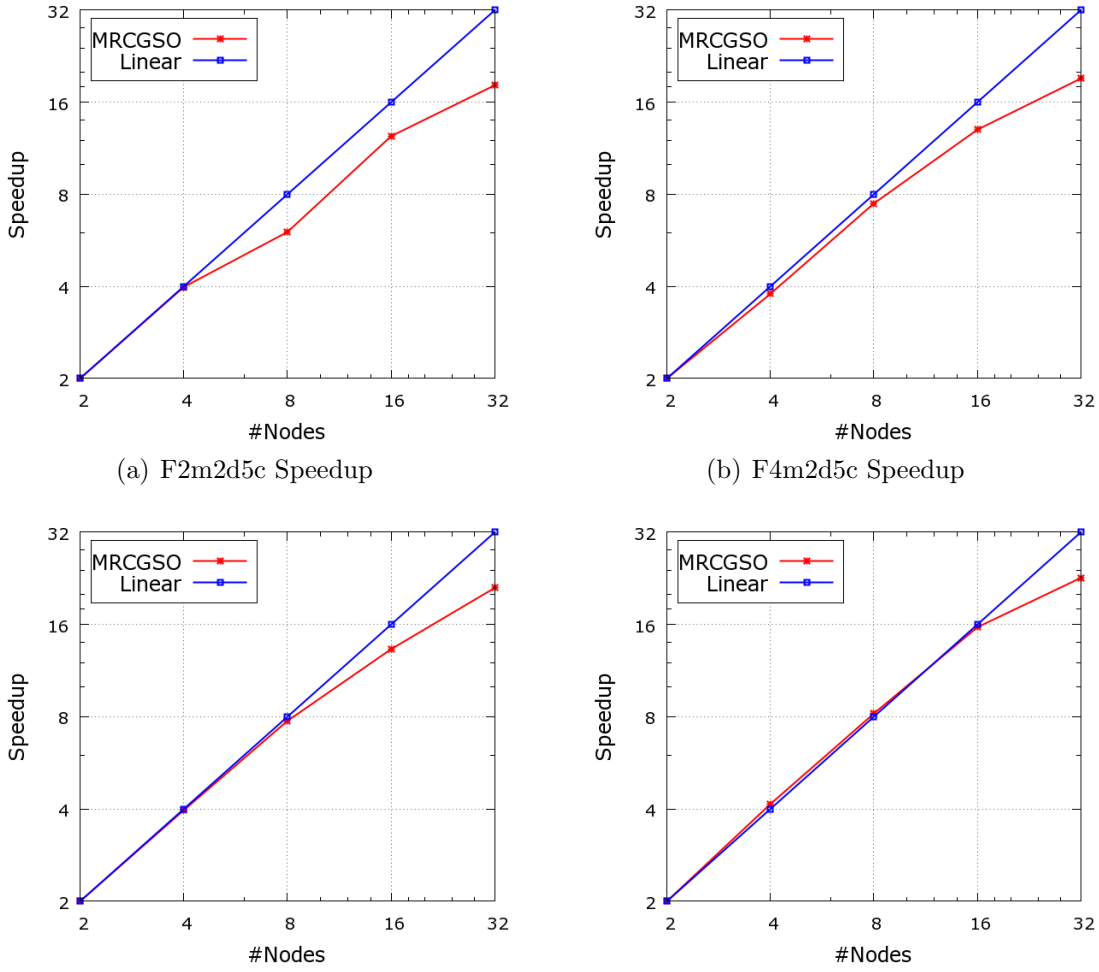
(c) F8m2d5c Running Time

(d) F16m2d5c Running Time

Figure 5.5: Running time results on the synthetic data sets for 2, 4, 8 and 16 million data records for average of 25 iterations of MRCGSO.

5.3.5. Complexity and Convergence Analysis

The overall time complexity of CGSO depends mainly on the amount of time it requires to find the neighborhood set for each glowworm and the amount of time



(a) F2m2d5c Speedup

(b) F4m2d5c Speedup

(c) F8m2d5c Speedup

(d) F16m2d5c Speedup

Figure 5.6: Speedup results on the synthetic data sets for 2, 4, 8 and 16 million data records for average of 25 iterations of MRCGSO.

it requires to retrieve the coverage set (cr_j) from the data set that is covered by individual glowworm g_i as well as the time to calculate $IntraDist_j$ between the glowworm g_i and its coverage set (cr_j). Furthermore, the overall time also depends on the dimensionality of the data set used, as well as the swarm size and the maximum number of iterations. The three proposed fitness functions F1, F2, F3 share the two fractions $|cr_j|$ and $IntraDist_j$ that are distinguished from each other in terms of use of SSE and $InterDist$ swarm-level fractions. The time needed to calculate SSE and $InterDist$ decreases with consequent iterations since the number of the candidate centroid set size $|c|$ is also reduced. Table 5.4 shows the average running time and average number of iterations (over 25 runs) are required to achieve the optimal number of centroids.

We can note that CGSO-F3 has a shorter average running time for Iris, Balance, Seed, and Mouse data sets compared to CGSO-F1, and CGSO-F2. For example, CGSO-F3 needs 10.74 seconds to converge for the Iris data set, whereas CGSO-F1 and CGSO-F2 need 11.58 and 10.74 seconds, respectively, for the same data set. Furthermore, CGSO-F3 converges faster than the other two for the Iris, Balance, Seed, and Mouse data sets. For instance, CGSO-F3 needs 101.04 iterations on average for Iris, whereas CGSO-F1 and CGSO-F2 need 101.8, and 108.92, respectively, for the same data set. In addition, CGSO-F1 has shorter average running time for Ecoli and Glass data sets as well as smallest average number of iterations, for example, CGSO-F1 needs 3.364 seconds and 17.56 iterations on average to converge for the Ecoli data set. Furthermore, CGSO-F2 has a shorter average running time for the VaryDensity data set and has the smallest average number of iterations such as CGSO-F2 needs 7.07 seconds and 110.16 iterations on average to converge.

Table 5.4: Running time and number of iterations.

Data set	CGSO-F1		CGSO-F2		CGSO-F3	
	Average Time (s)	Average #Iter.	Average Time (s)	Average #Iter.	Average Time (s)	Average #Iter.
Iris	10.79	101.80	11.58	108.92	10.74	101.04
Ecoli	3.37	17.56	3.50	18.44	3.82	20.16
Glass	15.95	78.72	18.35	89.88	23.02	82.20
Balance	14.68	96.40	13.87	95.84	13.65	94.04
Seed	5.04	27.36	5.00	27.24	4.82	26.12
Mouse	1.61	17.72	1.79	19.24	1.40	15.20
VaryDensity	7.72	122.40	7.07	110.16	7.88	125.88

5.4. Conclusion

In this chapter, we have presented a new clustering algorithm based on glowworm swarm optimization which takes into account the advantages of the GSO multimodal search capability to locate optimal centroids. The proposed algorithm CGSO can discover the clusters without needing to provide the number of clusters in advance. Experimental results on several real and artificial data sets with different characteristics show that our proposed algorithm is efficient compared to well-known clustering methods that have been used in the literature. In addition, three different fitness functions were proposed to add flexibility and robustness to the proposed algorithm. The average clustering quality, in terms of purity and entropy results over 25 runs, shows that our proposed algorithm is robust since the variances are relatively small. Moreover, we introduced a scalable MRCGSO version using the MapReduce parallel methodology to check the efficiency of the proposed CGSO clustering for large data sets. Results showed that MRCGSO can be successfully parallelized with the MapReduce methodology. Experiments were conducted on synthetic data sets and revealed that MRCGSO scales very well with increasing data set sizes, and achieves very close results to the optimal linear speedup.

CHAPTER 6. CONCLUSION AND FUTURE WORK

In this dissertation, we addressed the impact of the MapReduce framework for building scalable nature-inspired algorithms to solve clustering analysis problems as well as developing scalable algorithms that adapt to large problems. We formulated several nature-inspired algorithms to solve clustering problems using the distributed MapReduce framework, with the aim to minimize execution time and to maximize the optimization quality. MapReduce was chosen since it provides efficient communication, load balancing across the available computer nodes, and it resolves parallelization problems such as node failures.

We first proposed a scalable MR-CPSO algorithm using the MapReduce parallel methodology to improve the particle swarm optimization performance when applied on large scale clustering problems. In MR-CPSO algorithm, we formulated the clustering problem as an optimization problem. The fitness evaluations in the proposed algorithm are based on a fitness function that measures the distance between all data instances and particle centroids by taking the average distance between the particle centroids. MR-CPSO focused on the time consuming operations (the fitness evaluation, and particle centroids updating) by benefiting from the power of the MapReduce model. The experimental results conducted with both real-world and synthetic data sets revealed that MR-CPSO scales very well with increasing data set sizes. Furthermore, MR-CPSO scales very close to the linear speedup while maintaining good clustering quality.

Secondly, an intrusion detection system using the MapReduce methodology (IDS-MRCPSO) was proposed to detect the attacks in large scale networks that exchange large number of connections. The system was developed for the intrusion detection process was parallelized efficiently with the MapReduce methodology. To evaluate the proposed system, we used a large real intrusion detection data set

(KDD99) that has never been fully analyzed by any standard data mining algorithms. The experimental results showed that IDS-MRCPSO is efficient with increasing training data set sizes, and scales very close to the optimal speedup. In addition, we demonstrated that building detection model with larger training sample is likely to cover more significant regions and build a stronger detection model. Also, the results showed that the proposed system achieved better detection rates and low false alarm rates.

To circumvent the problem of the computation time and computational inefficiency of glowworm swarm optimization when difficult multimodal functions are to be optimized, this dissertation presented a parallel MapReduce based scalable glowworm swarm optimization (MR-GSO) approach. Experimental results were conducted with three multimodal functions to measure the peak capture rate, average minimum distance to the peak locations, and the speedup of our algorithm showing the benefit of the parallelization effect on the solution quality. Furthermore, increasing the number of glowworms used reduced the number of iterations required and led to an overall improvement in optimization quality. In addition, MR-GSO scales very well with increasing swarm sizes, and scales very close to the linear speedup while maintaining quality results.

In the last chapter of this dissertation, we proposed a clustering algorithm based on the glowworm swarm optimization (CGSO). In the proposed algorithm, we formulated the clustering problem as a multimodal optimization problem to extract the centroids based on glowworms' movement. In CGSO, the GSO optimization is adjusted to locate multiple optimal centroids such that each centroid represents a sub-solution and the combination of these sub-solutions formulates the global solution for the clustering problem. Experimental results on several real and artificial data sets with different characteristics show that our proposed algorithm is more efficient

compared to well-known clustering methods that have been proposed in the literature. Furthermore, a scalable parallel MapReduce based version (MRCGSO) was proposed. MRCGSO's main objective was to make CGSO scalable in order to be applied on large data sets. Experiments showed that MRCGSO can be successfully parallelized with the MapReduce methodology and the results of the synthetic data sets showed that MRCGSO scales very close to the optimal linear speedup.

In this dissertation, we tried to demonstrate that MapReduce model provides an efficient platform to parallelize the nature-inspired algorithms and applies them to large scale problems like large-scale clustering problems. The research results presented in this dissertation open the way to use the MapReduce model to parallelize other nature-inspired algorithms and expand the application area specially the applications that need to analyze big data sets. Furthermore, we explored the applicability of some nature-inspired algorithms to the development of self-organizing and efficient clustering techniques, which will meet the requirements of scalability and robustness when these algorithms deal with large scale clustering problems.

As for future work, our plan includes experiments with new intrusion data sets to evaluate the proposed intrusion detection system. In addition, we will expand the system to distinguish between the different types of intrusions. Furthermore, we plan to test our proposed algorithms with larger search spaces and higher dimensions as well as using larger swarm sizes. Moreover, we want to address the impact of the nature-inspired algorithm settings on the optimization quality. In addition, our plan includes applying the proposed algorithms on some massive applications such as community detection in social networking. In addition, we want to investigate the optimal configuration of a Hadoop cluster such as number of mappers and reducers to minimize the overhead of launching more mappers and reducers.

In our research we faced some challenges related to preprocessing issues for large

data sets such as feature selection, and data normalization. To tackle these challenges, we intend to develop a MapReduce tool that simplifies the preprocessing of big data sets. In fact, developing MapReduce nature-inspired algorithms is a promising step towards the development of a parallel data mining framework based on MapReduce that can be applied in many fields.

REFERENCES

[1] G. Bell, A. Hey, and A. Szalay. Beyond the data deluge. *Science 323 AAAS*, 39, 2006.

[2] J. Han. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, San Francisco, CA, USA, 2005.

[3] P. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Addison Wesley, 1 edition, May 2005.

[4] A. Baraldi and P. Blonda. A survey of fuzzy clustering algorithms for pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 29(6):778–785, Dec. 1999.

[5] A. Silic, M.-F. Moens, L. Zmak, and B. Basic. Comparing document classification schemes using k-means clustering. 5177:615–624, 2008.

[6] G.-J. Qi, C. C. Aggarwal, and T. S. Huang. On clustering heterogeneous social media objects with outlier links. In *Proceedings of the fifth ACM international conference on Web search and data mining*, WSDM '12, pages 553–562, NY, USA, 2012. ACM.

[7] D. K. Tasoulis, V. P. Plagianakos, and M. N. Vrahatis. Unsupervised clustering of bioinformatics data. In *In European Symposium on Intelligent Technologies, Hybrid Systems and their implementation on Smart Adaptive Systems*, pages 47–53, 2004.

[8] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI: The Complete Reference*. MIT Press Cambridge, MA, USA, 1995.

[9] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. pages 137–150, 2004.

[10] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proc. Fifth Berkeley Symp. on Math. Statist. and Prob.*, volume 1, pages 281–297. Univ. of Calif. Press, 1967.

[11] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD'96*, pages 226–231, 1996.

[12] Y. Zhao, G. Karypis, and U. Fayyad. Hierarchical clustering algorithms for document datasets. *Data Min. Knowl. Discov.*, 10(2):141–168, Mar. 2005.

[13] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of the IEEE ICNN '95*, pages 1942–1948. Australia, 1995.

[14] K. Krishnanand and D. Ghose. Detection of multiple source locations using a glowworm metaphor with applications to collective robotics. In *IEEE Swarm Intelligence Symposium*, pages 84 – 91, Pasadena, CA, USA, june 2005.

[15] M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, USA, 1998.

[16] C. Blum. Ant colony optimization: Introduction and recent trends. *Physics of Life Reviews*, 2:353–373, Dec. 2005.

[17] A. Engelbrecht. *Computational Intelligence: An Introduction*. Wiley, 2007.

[18] J. Handl, J. Knowles, and M. Dorigo. Strategies for the increased robustness of ant-based clustering. In *Engineering Self-Organising Systems*, volume 2977 of

Lecture Notes in Computer Science, pages 90–104. Springer Berlin Heidelberg, 2004.

[19] E. A. Omran M. and S. A. Particle swarm optimization method for image clustering. *International Journal of Pattern Recognition and Artificial Intelligence*, 3:297–322, 2009.

[20] T. Schoene, S. A. Ludwig, and R. Spiteri. Step-optimized particle swarm optimization. In *Proceedings of the 2012 IEEE CEC*. Brisbane, Australia, June 2012.

[21] (2012) Apache Software Foundation, Hadoop MapReduce [Online]. Available: http://hadoop.apache.org/mapreduce.

[22] (2012) Disco MapReduce Framework [Online]. Available: http://discoproject.org.

[23] T. Gunarathne, T. Wu, J. Qiu, and G. Fox. Cloud computing paradigms for pleasingly parallel biomedical applications. In *Proceedings of 19th ACM International Symposium on High Performance Distributed Computing*, pages 460–469. ACM, January 2010.

[24] S. Krishnan, C. Baru, and C. Crosby. Evaluation of mapreduce for gridding lidar data. In *Proceedings of the CLOUDCOM '10*, pages 33–40, Washington, DC, USA, 2010. IEEE Computer Society.

[25] (2012) Hadoop - Facebook Engg, Note [Online]. Available: http://www.facebook.com/note.php?note id=16121578919.

[26] (2012) Yahoo Inc. Hadoop at Yahoo! [Online]. Available: http://developer.yahoo.com/hadoop.

[27] Z. Weizhong, M. Huifang, and H. Qing. Parallel k-means clustering based on mapreduce. In *Proceedings of the CloudCom '09*, pages 674–679, Berlin, Heidelberg, 2009. Springer-Verlag.

[28] L. Guang, W. Gong-Qing, H. Xue-Gang, Z. Jing, L. Lian, and W. Xindong. K-means clustering with bagging and mapreduce. In *Proceedings of the 2011 44th Hawaii International Conference on System Sciences*, pages 1–8, Washington, DC, USA, 2011. IEEE Computer Society.

[29] S. Papadimitriou and J. Sun. Disco: Distributed co-clustering with map-reduce: A case study towards petabyte-scale end-to-end mining. In *Proc. of the IEEE ICDM '8*, pages 512–521, Washington, DC, USA, 2008.

[30] E. Alina, I. Sungjin, and M. Benjamin. Fast clustering using mapreduce. In *Proceedings of KDD '11*, pages 681–689, NY, USA, 2011. ACM.

[31] F. Cordeiro. Clustering very large multi-dimensional datasets with mapreduce. In *Proceedings of KDD '11*, pages 690–698, NY, USA, 2011. ACM.

[32] A. Jain, M. Murty, and P. Flynn. Data clustering: A review. NY, USA, 1999. ACM.

[33] X. Cui, T. Potok, and P. Palathingal. Document clustering using particle swarm optimization. In *IEEE Swarm Intelligence Symposium*, pages 185–191, Pasadena, California, USA, 2005. ACM.

[34] R. Orlandic, Y. Lai, and W. Yee. Clustering high-dimensional data using an efficient and effective data space reduction. In *Proc. ACM 14th Conf. on Information and Knowledge Management*, pages 201–208, Bremen, Germany, 2005.

[35] A. Grama, A. Gupta, G. Karypis, and V. Kumar. *Introduction to Parallel Computing*. Addison-Wesley, USA, 2003.

[36] Y. Zhao and G. Karypis. Evaluation of hierarchical clustering algorithms for document datasets. In *Proceedings of the eleventh CIKM '02*, pages 515–524, NY, USA, 2002. ACM.

[37] H. Shi and R. Eberhart. Parameter selection in particle swarm optimization. In *Proc. 7th Annual Conference on Evolutionary Programming*, pages 201–208, San Diego, CA, 1998.

[38] I. Trelea. The particle swarm optimization algorithm: convergence analysis and parameter selection. In *Information Processing Letters*, 2003.

[39] I. Witten, E. Frank, and M. Hall. *Data Mining: Practical Machine Learning Tools And Techniques, 3rd Edition*. Morgan Kaufmann, 2011.

[40] A. Lazarevic, V. Kumar, and J. Srivastava. Intrusion detection: A survey. *Managing Cyber Treats*, pages 19–78, 2005.

[41] H. Debar and J. Viinikka. Intrusion detection: Introduction to intrusion detection and security information management. *Foundation of Security Analysis and Design III*, 3655:207–236, 2005.

[42] K. Leung and C. Leckie. Unsupervised anomaly detection in network intrusion detection using clusters. In *Proceedings of the Twenty-eighth Australasian conference on Computer Science*, pages 333–342, Newcastle, Australia, 2005. Australian Computer Society.

[43] M. Jianliang, S. Haikun, and B. Ling. The application on intrusion detection based on k-means cluster algorithm. In *Proceedings of the 2009 International*

Forum on Information Technology and Applications, pages 150–152, Washington, DC, USA, 2009. IEEE Computer Society.

[44] W. Jiang, M. Yao, and J. Yan. Intrusion detection based on improved fuzzy c-means algorithm. In *Proceedings of the 2008 International Symposium on Information Science and Engieering*, pages 326–329, Washington, DC, USA, 2008. IEEE Computer Society.

[45] Z. Li, Y. Li, and L. Xu. Anomaly intrusion detection method based on k-means clustering algorithm with particle swarm optimization. In *Proceedings of the 2011 International Conference of Information Technology, Computer Engineering and Management Sciences*, pages 157–161, Washington, DC, USA, 2011. IEEE Computer Society.

[46] J. Mazel, P. Casas, Y. Labit, and P. Owezarski. Sub-space clustering, inter-clustering results association & anomaly correlation for unsupervised network anomaly detection. In *Proceedings of the 7th International Conference on Network and Services Management*, pages 73–80, Paris, France, 2011.

[47] H. Gao, D. Zhu, and X. Wang. A parallel clustering ensemble algorithm for intrusion detection system. In *Proceedings of the DCABES'10 Conference*, pages 450–453, Washington, DC, USA, 2010. IEEE Computer Society.

[48] S. D. Bay, D. Kibler, M. J. Pazzani, and P. Smyth. The uci kdd archive of large data sets for data mining research and experimentation. *SIGKDD Explor. Newsl.*, 2:81–85, December 2000.

[49] Z. Gao, T. Li, J. Zhang, C. Zhao, and Z. Wang. A parallel method for unpacking original high speed rail data based on mapreduce. *Springer Berlin Heidelberg*, 124:59–68, 2012.

[50] W. Zhu, N. Zeng, and N. Wang. Sensitivity, specificity, accuracy associated confidence interval and roc analysis with practical sas implementations. In *In Proceedings of the NorthEast SAS Users Group Conference NESUG10*, 2010.

[51] K. N. Krishnanand and D. Ghose. Glowworm swarm optimization algorithm for hazard sensing in ubiquitous environments using heterogeneous agent swarms. *Soft Computing Applications in Industry*, 226:165–187, 2008.

[52] K. Krishnanand and D. Ghose. Glowworm swarm optimisation: a new method for optimising multi-modal functions. *International Journal of Computational Intelligence Studies*, 1:93–119, 2009.

[53] G. Venter and J. Sobieszczanski-Sobieski. A parallel particle swarm optimization algorithm accelerated by asynchronous evaluations. *Journal of Aerospace Computing, Information, and Communication*, 2005.

[54] M. Ismail. Parallel genetic algorithms (PGAs): master slave paradigm approach using MPI. In *E-Tech 2004*, pages 83 – 87, july 2004.

[55] A. McNabb, C. Monson, and K. Seppi. Parallel PSO using mapreduce. In *IEEE Congress on Evolutionary Computation*, pages 7–14, Sept. 2007.

[56] C. Jin, C. Vecchiola, and R. Buyya. MRPGA: an extension of mapreduce for parallelizing genetic algorithms. In *Proceedings of the 2008 Fourth IEEE International Conference on eScience*, ESCIENCE '08, pages 214–221, Washington, DC, USA, 2008. IEEE Computer Society.

[57] B. Wu, G. Wu, and M. Yang. A mapreduce based ant colony optimization approach to combinatorial optimization problems. In *Natural Computation (ICNC), 2012 Eighth International Conference on*, pages 728 –732, may 2012.

[58] K. Krishnanand and D. Ghose. Glowworm swarm optimization for simultaneous capture of multiple local optima of multimodal functions. *Swarm Intelligence*, 3:87–124, 2009.

[59] D. S. Hochbaum and D. B. Shmoys. A best possible heuristic for the k-Center problem. *Mathematics of Operations Research*, 10(2):180–184, May 1985.

[60] T. Kohonen. Learning Vector Quantization: The Handbook of Brain Theory and Neural Networks. pages 631–634, 2003.

[61] U. Maulik and S. Bandyopadhyay. Genetic algorithm-based clustering technique. *Pattern Recognition*, 33(9):1455 – 1465, 2000.

[62] P. Shelokar, V. Jayaraman, and B. Kulkarni. An ant colony approach for clustering. *Analytica Chimica Acta*, 509(2):187 – 195, 2004.

[63] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *SIGKDD*, 11:10–18, Nov. 2009.