Received: 2 January 2018 | Revised: 25 April 2018 | Accepted: 18 June 2018
DOI: 10.1002/cpe.4872

WILEY

RESEARCH ARTICLE

Distributed Whale Optimization Algorithm based on MapReduce

Yasser Khalil | Mohammad Alshayeji | Imtiaz Ahmad

Department of Computer Engineering, Kuwait University, Kuwait City, Kuwait

Correspondence
Yasser Khalil, Department of Computer Engineering, Kuwait University, Kuwait City, Kuwait.
Email: yasser.khalil@grad.ku.edu.kw

Summary

Whale Optimization Algorithm (WOA) is a recent swarm intelligence based meta-heuristic optimization algorithm, which simulates the natural behavior of bubble-net hunting strategy of humpback whales and has been successfully applied to solve complex optimization problems in a wide range of disciplines. However, when applied to large-size problems, its performance degrades due to the need of massive computational work load. Distributed computing is one of the effective ways to improve the scalability of WOA for solving large-scale problems. In this paper, we propose a simple and robust distributed implementation of WOA using Hadoop MapReduce named MR-WOA. MapReduce paradigm is adopted as the distribution model since it is one of the most mature technologies to develop parallel algorithms which automatically handles communication, load balancing, data locality, and fault tolerance. The design of MR-WOA is discussed in details using the MapReduce paradigm. Experiments are conducted for a set of well-known benchmarks for evaluating the quality, speedup, and scalability of MR-WOA. The conducted experiments reveal that our approach achieves a promising speedup. For some benchmarks, speedup scales linearly with increasing the number of computational nodes.

KEYWORDS

evolution algorithm, Hadoop, MapReduce, meta-heuristic, Whale Optimization Algorithm (WOA)

1 | INTRODUCTION

Evolutionary Algorithms (EAs) are becoming an essential part of solving optimization problems and are increasing in popularity due to the following reasons: they are based on a simple concept, easy to develop, do not need auxiliary information, avoid local optima, and are used in a wide range of applications. In the past few years, a wide variety of nature-inspired meta-heuristic algorithms have emerged. Some of the well-known EAs include Genetic Algorithms (GA),[1,2] Genetic Programming (GP),[3,4] and Simulated Annealing (SA).[5,6] Other respected swarm-based algorithms, including Particle Swarm Optimization (PSO)[7] and Ant Colony Optimization (ACO),[8,9] share common features with EAs and are therefore classified as part of EA group. It is worth mentioning that all EAs are considered as population-based meta-heuristic algorithms. Population-based EAs consist of a population and individuals within the population exchange their local information between one another to achieve their goal. The goal being defined as the best solution found so far.

The Whale Optimization Algorithm (WOA) introduced by Mirjalili and Lewis in 2016 is one of the new population-based meta-heuristic optimization algorithms.[10] More specifically, WOA falls under the swarm-based algorithms. WOA emulates the natural hunting behavior of humpback whales. These types of whales have a unique feeding behavior called bubble-net, where they trap their preys by encircling them in a network of bubbles along a '9'-shaped path and hunt them once they reach the surface. WOA consists of two processes: exploitation and exploration. Furthermore, within the exploitation process, another two processes exist, namely encircling prey and spiral updating position. The former process is used to search for the best global solution or prey (in the context of whales). The latter process, ie, exploration, is used to randomly search for the prey, which helps in avoiding local optima. WOA is considered powerful in reaching the best global solution and escaping local optima in a satisfying time due to its strength in maintaining a good balance between exploitation and exploration. According to Mirjalili and Lewis, results of WOA have demonstrated its effectiveness and are considered competitive when compared with other well-known optimization algorithms.[10]

Recently, researchers have been attracted to WOA and utilized it successfully to solve various optimization problems in diverse fields due to its effectiveness in exploring the search space, avoiding local optima and fast convergence rate. Aziz et al applied WOA to multilevel thresholding image segmentation.[11] Hassanien et al used WOA for the binarization of handwritten Arabic manuscripts.[12] Mostafa et al employed WOA in the medical field segmentation, more specifically for liver segmentation in MRI images.[13] Aljarah et al used WOA for training multilayer perceptron (MLP) neural networks.[14] Mafarja and Mirjalili proposed a hybrid WOA by integrating it with SA for feature selection.[15] Oliva et al enhanced WOA by combining it with chaotic maps that are used for better estimating the parameters of Photo-Voltaic (PV) cells using circuit models.[16] Tharwat et al used WOA with Support Vector Machines (SVMs) for predicting drug toxicity prior its development.[17] Dao et al proposed a planning algorithm used for optimizing mobile robot path planning problems[18]; the algorithm is based on a multi-objective approach and is integrated with WOA. Ling et al applied Lévy flight trajectory to WOA for enhancing its precision in finding best global optima and convergence rate.[19] Abdel-Basset et al[20] integrated WOA based algorithm with Cloudsim toolkit to competently solve the virtual machine placement problem in cloud computing. Zhang et al[21] demonstrated the use of WOA for synthesis of linear antenna arrays, which are widely used in radar, communications, remote sensing, and in many other fields. Abdel-Basset et al[22] introduced a novel modified version of WOA for cryptanalysis of a well-known Merkle-Hellman public key cryptosystem. Other recent fruitful applications of WOA include software testing,[23] wind speed prediction,[24] and power systems. [25,26]

The traditional sequential WOA operated on a single machine has proven its performance in solving optimization problems when compared with other well-known EAs. However, when faced with complex problems, its performance decreases as huge computational efforts and time will be needed. Gong et al[27] discussed that distributed EAs are rapidly gaining ground as they utilize a set of computational resources for enhancing their performance capabilities. The authors also presented a comprehensive study of the well-known distributed EAs and models, where each has its own features and characteristics. In the work of Tan and Ding,[28] a review was provided on distributing swarm-based algorithms more specifically on Graphical Processing Units (GPUs). Furthermore, in the work of Zhan et al,[29] the use of EAs on cloud computing was proposed and a parallelized heterogeneous Differential Evolution (DE) algorithm on a distributed cloud was developed.

Several parallel and distributed approaches exist for developing distributed EAs. On top of the ones mentioned earlier, some other approaches include Message-Passing Interface (MPI),[30] OpenMP,[31] Parallel Virtual Machine (PVM),[32] and Hadoop MapReduce.[33,34] In this research paper, MapReduce paradigm is selected as the distributed approach for partitioning WOA.

MapReduce is a programming model that eases the development of scalable and fault tolerant parallel applications in a distributed environment. This model is based on two main functions, namely Map and Reduce, which work together in a divide-and-conquer mechanism. Parallelization in MapReduce occurs through distributing the workload among a cluster of heterogeneous commodity machines. Apache Hadoop[35,36] is an open-source implementation of MapReduce framework written in Java. Due to the good features that come with MapReduce and its ease of use, researchers are heavily utilizing it in developing applications from a wide range of disciplines, including text mining, machine learning, bioinformatics, log file analysis, financial analysis, and report generation. Moreover, MapReduce is being supported by world's top cloud providers such as Google and Amazon with its Elastic MapReduce (EMR) service. In addition, MapReduce has gained increasing attention in distributing EAs as developers only need to build the target algorithm in a Map and Reduce fashion. This allows them to focus more on the algorithm itself and relieving them from thinking about the low-level details of managing distributed implementations.

In this paper, we present a distributed WOA based on MapReduce framework (MR-WOA). To the best of our knowledge, this is the first distributed implementation of WOA. The following are the main contributions of this paper:

- The MapReduce concept is successfully applied to WOA algorithm.
- Detailed explanation outlining the implementation of MR-WOA.
- Convergence analysis is conducted to determine the accuracy of the best obtained solution and the rate of convergence.
- Speedup analysis is conducted measuring the behavior of MR-WOA against varying number of computational nodes which determine the scalability of our approach.
- Speedup analysis with very large populations.
- Speedup analysis with varying dimensions.

The rest of the paper is structured as follows: in Section 2, we discuss the related work in the area of distributed implementation of optimization algorithms using MapReduce paradigm. Section 3 introduces both MapReduce model and the WOA. Section 4 describes the implementation details of MR-WOA. In Section 5, the experimental setup and results are presented. Finally, conclusion and suggestions for future research are made in Section 6.

2 | RELATED WORKS

Many works exist that attempt to distribute optimization algorithms under different parallel and distributed models, but we will discuss the ones implemented using the Hadoop MapReduce paradigm. Some of the MapReduce-based optimization algorithms that will be briefly reviewed are PSO, GA, Krill Herd,[37] ACO, and some others.

McNabb et al[38] proposed a parallel PSO implementation using MapReduce. Each MapReduce job, representing a single iteration, distributes the entire population among the computational nodes (Mappers) in the cluster. Chang[39] enhanced the PSO by combining it with crossover operation

of GA and implemented it using the MapReduce for efficiently solving the Traveling Salesman Problem (TSP). The model used in the work of Chang[39] is similar to the one used in the work of McNabb et al[38] where each Mapper operates only on a chunk of the population. Moreover, Ghosh[40] implemented two models for parallelizing PSO and their performance results were analyzed. In the first model, each Mapper in the cluster works independently on the entire population, whereas in the second model, the entire population is divided among the Mappers. Results of the analysis showed that both techniques achieve promising results that outperform the sequential PSO. However, it was reported that the performance of the first technique is better than the second, without providing any justification.

Genetic Algorithm is another optimization algorithm that was parallelized using MapReduce. Di Martino et al[41] implemented GA based on MapReduce using the global parallelization model. This model is like the master-slave model. The master performs most of the algorithm and assigns individuals to the slaves for fitness evaluation. The only difference between this model and the sequential model is the use of parallelization for the fitness function. In another work, Geronimo et al[42] developed a MapReduce-based GA using the island model. In the island model, the population gets divided into several sub-populations, so-called "islands." Each sub-population gets processed by single Mapper independently. Two main advantages of the model are the improvement in global search ability and enhancement in population diversity due to migration. The achieved results show that the use of parallel GA saves more than 50% of the time when compared with the sequential GA.

Jin et al[43] implemented an extension of MapReduce for parallelizing GA. The proposed architecture consists of a master node and multiple Mappers and Reducers. On top of the traditional mapper and reducer phases comes an additional reducer phase. This additional phase consists of only one Reducer and is responsible for selecting the global optimum individual. Verma et al[44] stated that the employment of an additional reducer phase is unnecessary. Furthermore, the single master node can affect the scalability of the proposed algorithm. So, in their work, they proposed a MapReduce-based GA using the grid model. The task of the Mappers is to evaluate the population's fitness, and Reducers are held responsible for applying the rest of the algorithm on the sub-population they receive. The scalability experiments showed that performance degrades when the number of requested nodes exceeds the number of physical nodes.

Ludwig[45] used MapReduce to parallelize Krill Herd algorithm. Çolak and Varol[46] proposed a distributed optimization solution of Circular Water Wave (CWW) algorithm based on MapReduce. The results have shown that the algorithm can be implemented in MapReduce. As for future works, they suggested to test their proposed approach on more complex functions.

Wu et al[47] designed and implemented a new ACO algorithm using the Hadoop MapReduce methodology. Two optimization problems were used to evaluate the algorithm: 0-1 knapsack problem and TSP. Two different models were considered for implementation. First, all Mappers operate on the entire population, and no exchange of information occurs between computational nodes. Second, the population gets partitioned into chunks and each Mapper operates on one chunk independently. Results have proven the effectiveness of the proposed implementation.

In addition, several kinds of research exist in distributing algorithms related to the field of data mining using MapReduce. As an example, in the clustering area,[48-51] implemented MapReduce-based Artificial Bee Colony (ABC), PSO, Glowworm Swarm Optimization (GSO), and fuzzy c-means clustering algorithm, respectively. Moreover, in the area of distributed document clustering, a MapReduce-based hybrid PSO K-Means clustering (MR-PKMeans) algorithm was proposed in the work of Judith and Jayakumari.[52] PSO is used in the proposed algorithm due to its global search ability, which helps in improving the algorithm's accuracy. Results of MR-PKMeans confirm improvement in accuracy and also reduction in execution time.

3 | BACKGROUND

In this section, a brief introduction to both MapReduce paradigm and WOA is given, providing readers with the necessary basics for tackling our proposed approach in the following section.

3.1 | Hadoop MapReduce framework

Hadoop is an open source framework implemented in Java and consists of two core components: MapReduce and Hadoop Distributed File System (HDFS) that are related to processing and storage, respectively. MapReduce is a parallel programming framework first introduced by Google and is based on a two phase processing strategies, being performed on data, namely Map and Reduce. These two functions work together in a divide-and-conquer fashion, where the task of the Map function is to parallelize the application, while the Reduce function handles the gathering and merging of intermediate results for final processing and generating the final output. In the MapReduce model, all inputs and outputs including intermediate results are in the form of <key, value> pair. More formally, the Map function accepts a <key, value> pair as an input for processing and outputs a list of intermediate <key, value> pairs:

$$\text{Map(key1, value1)} \rightarrow \text{list(key2, value2)}.$$

The Reduce function then merges all intermediate values holding the same key and results in the final output:

$$\text{Reduce(key2, list(value2))} \rightarrow \text{list(key3, value3)},$$

where key1, value1, key2, value2, key3, and value3 are equivalent to the input key, input value, intermediate key, intermediate value, output key, and output value, respectively.

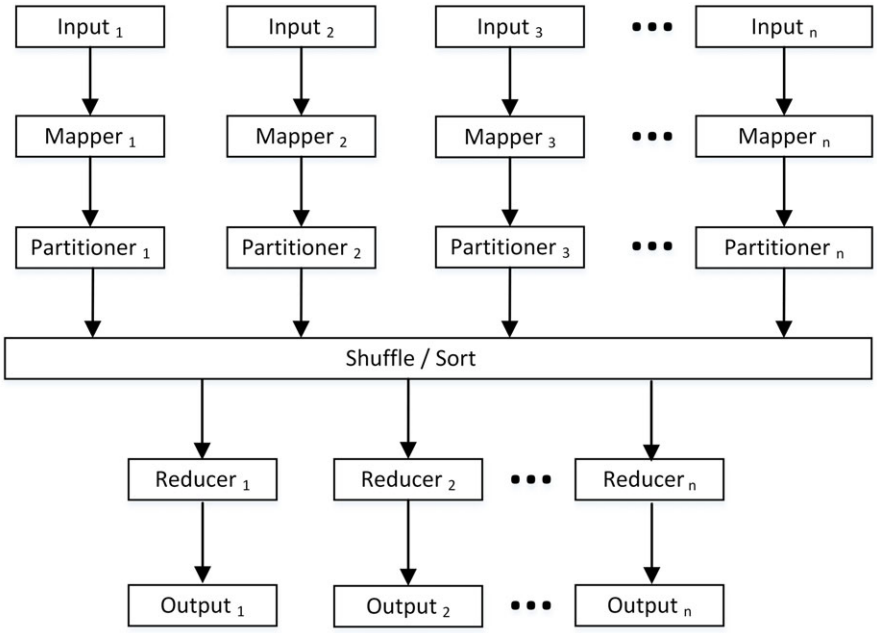

FIGURE 1 Hadoop MapReduce architecture

To run an application using MapReduce, it first needs data to operate on. The input data is stored in a file on HDFS in the <key, value> form. HDFS is the other core component of Hadoop used for storing data. Initially, a master node partitions the input file into blocks called splits, which contains <key, value> pairs called records. Next, the splits get assigned to Mappers on the cluster for processing. Each Mapper operates on a different split independently and in parallel; however, they work on the same tasks. In the mapper phase, each record in the split gets processed individually. This process repeats until all records in the split get processed. After the Mappers are done processing the splits, the intermediate output in <key, value> pairs are passed to the Reducer. In the reducer phase, the Reducers takes the intermediate outputs assigned to it, process them, then outputs multiple sets of <key, value> pairs, and finally stores them on HDFS. Similar to Mappers, Reducers also operate independently and in parallel. The intermediate outputs being inputted to Reducers are basically lists of values, where they get sorted and then gets associated with their keys. This process is called shuffling. The shuffling occurs between the mapper and the reducer phase. Along with the shuffling stage comes the concept of partitioner, where decisions are made regarding which keys should be sent to which Reducer for final processing. Decisions are made according to the following hash function on keys: hash(*key*)%*numR*, where *numR* is the number of Reducers. The overall process of the MapReduce model is depicted in Figure 1. Lastly, in addition to the Map and Reduce function, there exists a Driver function (not shown in Figure 1) that initializes and leads the MapReduce job by binding both Map and Reduce functions.

3.2 | Whale Optimization Algorithm (WOA)

WOA[10] is a newly developed optimization algorithm naturally inspired by the bubble-net hunting strategy of humpback whales. The bubble-net attacking strategy is a behavior that is unique and perceived only in humpback whales. These whales hunt their preys by encircling them with bubbles then swims up towards the surface in a spiral shape to prey on them. WOA consists of two phases that are used for reaching the best solution: exploitation and exploration. In the context of whales, the best solution is defined as the target prey. The following subsections explain in details the mathematical models of each phase.

3.3 | Exploitation phase

Exploitation phase is where whales (search agents) move closer to the prey (current best solution agent). Encircling prey and bubble-net attacking method are other names for exploitation and are used interchangeably. Two approaches exist for exploitation: shrinking encircling mechanism and spiral updating position. The shrinking encircling mechanism is achieved by the mathematical model as in Equation (1) and Equation (2).

$$\vec{D} = |\vec{C} \cdot \vec{X^*}(t) - \vec{X}(t)| \tag{1}$$

$$\vec{X}(t+1) = \vec{X^*}(t) - \vec{A} \cdot \vec{D}. \tag{2}$$

All symbols and variable definitions for the equations presented in this section are defined in Table 1. Over the course of iterations, for both exploitation and exploration, the value of $\vec{a}$ decreases linearly by the equation defined in Table 1, and $\vec{X^*}(t)$ gets updated only in the case where a

TABLE 1 Symbols used in WOA equations

Symbol	Meaning	Symbol	Meaning
t	Current iteration	$\vec{A}$	Coefficient vector. Calculated as follows: $\vec{A} = 2\vec{a} \cdot \vec{r} - \vec{a}$
$\vec{X}(t)$	Position vector of the current search agent being updated	$\vec{C}$	Coefficient vector. Calculated as follows: $\vec{C} = 2 \cdot \vec{r}$
$\vec{X}^*(t)$	Position vector of the best solution agent obtained so far (leader)	$\vec{r}$	Random vector in [0,1]
$\vec{X}(t+1)$	Updated position vector of the agent for the next iteration	b	Constant that defines the shape of the logarithmic spiral
$\overrightarrow{X_{rand}}$	Random position vector (random whale) chosen from the population	l	Random number in [-1,1]
$\vec{D}$	Position vector	p	Random number in [0,1]
$\vec{D'}$	Position vector indicating the distance between the ith search agent and the best solution agent	$\vec{a}$	Vector that linearly decreases from 2 to 0 over the course of iterations. Calculated as follows: $\vec{a} = 2 - t * (2/\text{MaxIterations})$
$\|\|$	Absolute value	$\cdot$	Element-by-element multiplication

better solution agent is found. After exploiting a search agent, its new position will be anywhere between its current position and the position of the best solution agent.

The other approach used in the exploitation phase is the spiral updating position. This approach first calculates the distance between the search agent located at (X, Y) and the best search agent located at (X^*, Y^*). Then, the spiral equation in Equation (3) and Equation (4), which mimics the helix-shaped movement of humpback whales, is used to calculate the new position of the search agent

$$\vec{D'} = |\vec{X^*}(t) - \vec{X}(t)| \tag{3}$$

$$\vec{X}(t+1) = \vec{D'} \cdot e^{bl} \cdot \cos(2\pi l) + \vec{X^*}(t). \tag{4}$$

During exploitation, humpback whales use both shrinking encircling mechanism and spiral updating position simultaneously for surrounding preys. Thus, to model this behavior, an assumption was made that 50% of the time humpback whales use the shrinking model, and in the other 50%, spiral mechanism is used.[10] As a result, the mathematical model becomes as in Equation (5)

$$\vec{X}(t+1) = \begin{cases} \vec{X^*}(t) - \vec{A} \cdot \vec{D}, & p < 0.5 \\ \vec{D'} \cdot e^{bl} \cdot \cos(2\pi l) + \vec{X^*}(t), & p \geq 0.5. \end{cases} \tag{5}$$

3.4 | Exploration phase

In addition to the exploitation phase, an exploration phase also exists in WOA. In exploration phase, search agents randomly explore the search space hoping to find a search agent that has a better optimal solution than the existing one. In contrast to exploitation, search agent in exploration randomly selects a search agent from the population rather than the best solution agent and approaches it. This phase prevents the algorithm from getting stuck in local optima. This mechanism is mathematically modeled as in Equation (6) and Equation (7)

$$\vec{D} = |\vec{C} \cdot \overrightarrow{X_{rand}} - \vec{X}| \tag{6}$$

$$\vec{X}(t+1) = \overrightarrow{X_{rand}} - \vec{A} \cdot \vec{D}. \tag{7}$$

Algorithm 1 presents the pseudo code of the WOA. The algorithm begins with initializing a random set of search agents, ie, the population. Each individual in the initial random population gets evaluated, and the best solution agent is extracted. Then, the following steps are repeated until the stopping criteria is met. First, the coefficients get updated, and random numbers are generated. After that, depending on the values of the random numbers ($\vec{A}$ and p), a decision is made on which equation to be used for updating search agent's position, Equations (2), (7), or (4). Exploration is done if $|\vec{A}| \geq 1$, which forces the search agent to move away from the best solution agent found so far; furthermore, if $|\vec{A}| < 1$, then exploitation is done. Also, the decision between using shrinking or spiral mechanism for exploitation depends on the random value p. Next, after updating all search agents in the population, each individual gets checked whether it exceeded the search space; if yes, it gets modified. Lastly, they get evaluated, and the best solution agent, if found, is extracted. Figure 2 depicts the flowchart of the WOA.

Algorithm 1 Whale Optimization Algorithm

```
Random initialization of whale population
Evaluate fitness of each search agent
X* = best solution search agent
while (iter < maximum number of iterations) do
    for (each search agent) do
        Update a, A, C, l, and p
        if (p < 0.5) then
            if (|A| < 1) then
                Exploitation: update the position of the current search agent using the shrinking mechanism by Equation (2)
            else if (|A| ≥ 1) then
                X_rand = a random search agent from the population
                Exploration: update the position of the current search agent by Equation (7)
            end if
        else if (p ≥ 0.5) then
            Exploitation: update the position of the current search agent using the spiral mechanism by Equation (4)
        end if
    end for
    Amend search agents that exceed the search space
    Evaluate fitness of each search agent
    Update X* (if found)
    iter + +
end while
return X*
```

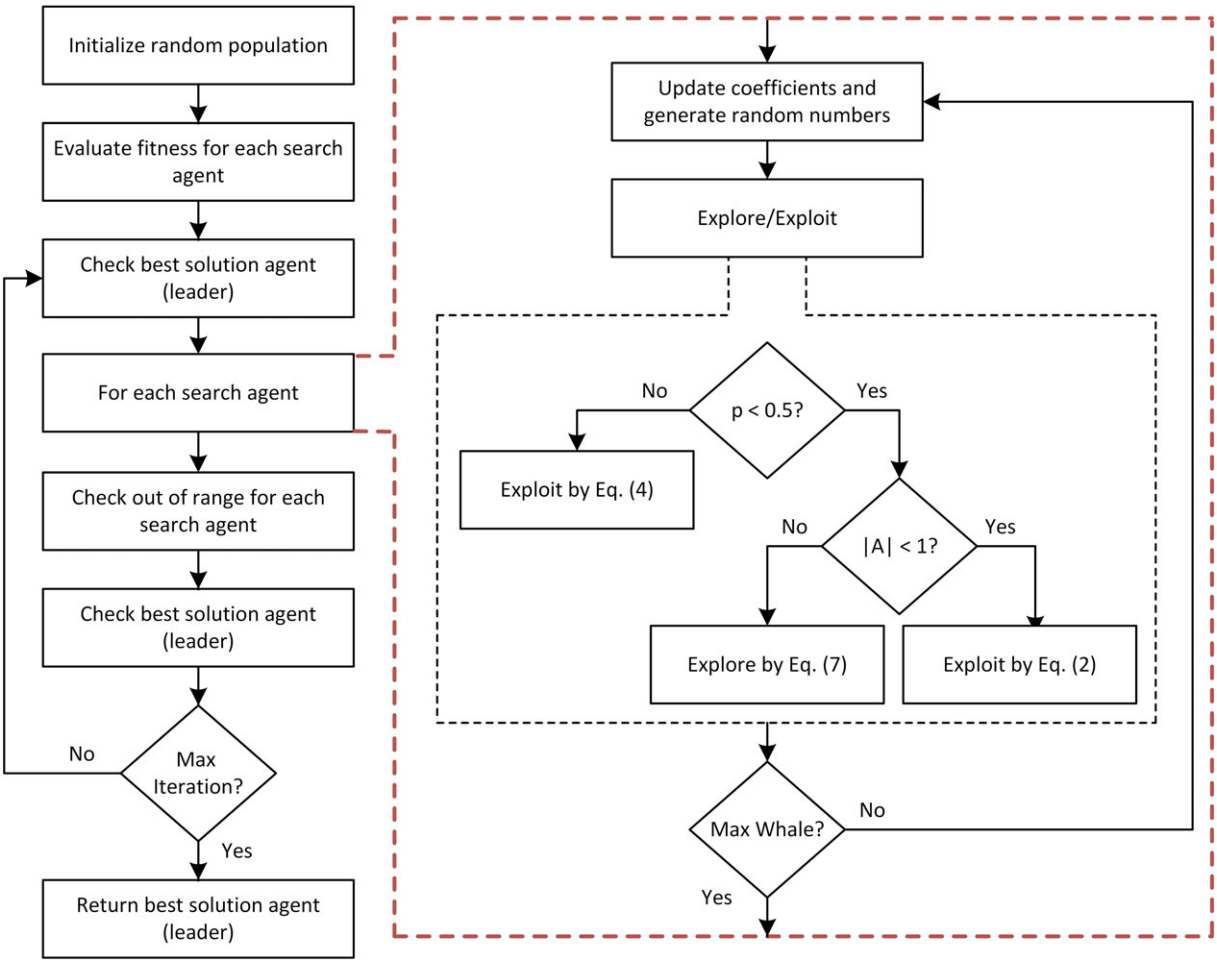

FIGURE 2 Flowchart of WOA

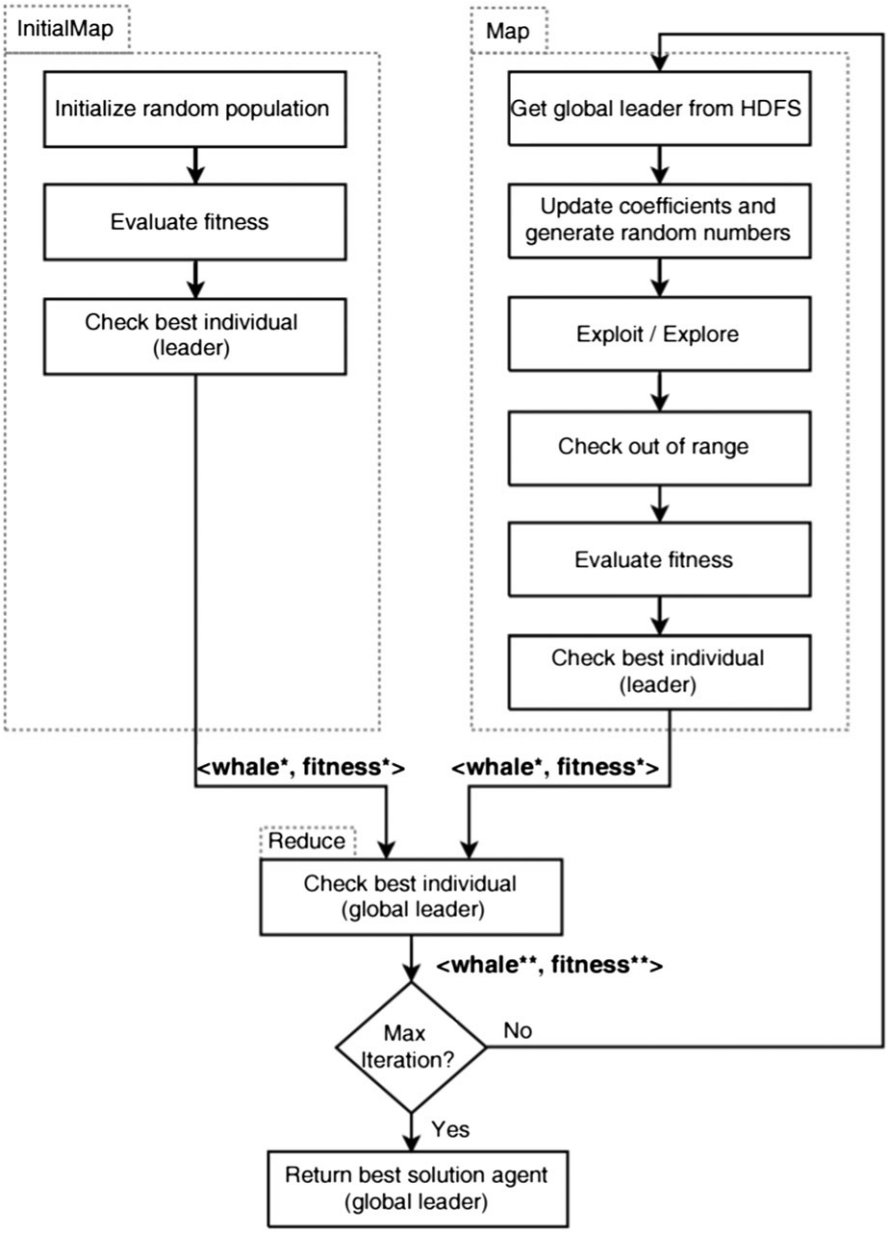

FIGURE 3 Flowchart of MR-WOA

4 | MR-WOA IMPLEMENTATION

In the previous section, a short introduction to both MapReduce paradigm and WOA was given. In this section, we will present and discuss in details our proposed approach, MapReduce-based WOA (MR-WOA).

As mentioned earlier, any application built based on MapReduce consists of at least three functions: Driver, Map, and Reduce. A detailed algorithmic description for MR-WOA's Driver, Map, and Reduce functions are presented in Algorithms 2-5 in a pseudo code fashion. Additionally, a flowchart for MR-WOA is shown in Figure 3. The Driver function (Algorithm 2) begins by reading input parameters used for setting MapReduce job's configuration. The parameters read in our Driver function include the number of Mappers, number of Reducers, number of variables (dimensions of the benchmark functions), the number of iterations (stopping criteria), and the population size. In addition, the Driver function drives the main iteration loop of the algorithm and controls its termination. Each and every iteration in the main loop is encapsulated as a separate MapReduce job. In this section, a single iteration represents a single MapReduce job; hence, the words iteration and MapReduce job are used interchangeably. The Driver function is responsible for calling and executing all chained MapReduce jobs, which internally calls the Map and Reduce functions, respectively. Lastly, the termination of MR-WOA is done by checking whether the stopping criteria has been met or not.

Following the Driver function comes the Map function. In the design of our Map function, the entire WOA is executed, and at the end of the function, the leader, defined by the search agent that owns the best fitness value, is emitted. In each run, each Mapper operates on a chunk of the population, and their leaders are emitted; therefore, if m Mappers exist, then m leaders will be emitted to the Reducer. However, only in the first iteration of our proposed approach, a different Mapper function is used, namely *InitialMap*, presented in Algorithm 3. The purpose of this function

is to randomly initialize the whale's population. Each Mapper running the *InitialMap* function generates a chunk of the population so that the sum of all generated chunks is equal to the desired population size. For example, if *p* population is required and *m* Mappers exist, then each Mapper will generate *p*/*m* search agents. Dummy inputs to the *InitialMap* function were used, as initially no input data exists and the population is randomly created. After initialization, search agents get evaluated and stored in HDFS in the form of <whale, fitness> pairs. All Mappers keep track of their best search agent (whale*) for emitting it, at the end of the function, to the Reducer in the following format: <whale*, fitness* >.

Algorithm 2 Driver function

Input: parameters = {*numMappers*, *numReducers*, *numVariables*, *maxIteration*, *populationSize*}

```
iter = 0
while (iter ≤ maxIteration) do
    Initialize job configuration
    Set parameters in job configuration
    Setup Mapper and Reducer nodes
    if (iter == 0) then
        setMapperClass: InitialMap()
    else
        setMapperClass: Map()
    end if
    setReducerClass: Reduce()
    waitForJobCompletion()
    iter++
end while
```

In the pseudo codes below (Algorithms 3-5) and in the MR-WOA flowchart (Figure 3), the symbols whale**, fitness**, whale*, and fitness* represent the best global leader of the entire iteration, fitness of the global leader, best local leader in a certain Mapper, and fitness of the local leader in that Mapper, respectively.

Algorithm 3 InitialMap function

Input: Key: *dummyWhale*, Value: *dummyFitness*

Output: Key: whale*, Value: fitness*

```
Randomly initialize population
Evaluate fitness
Store individual in HDFS
Emit(whale*, fitness*)
```

For the rest of the MapReduce jobs, starting from the second, a different Map function is utilized to operate on Mappers, namely *Map* (Algorithm 4). This function executes the entire WOA with some additional steps. First, the global best leader (whale**) and its corresponding fitness (fitness**), resulted from the previous MapReduce job, are read from HDFS. Second, either exploitation or exploration is applied on each search agent depending on the random numbers generated (*p* and $|\vec{A}|$). After updating search agent's positions, they are checked whether they went outside their defined search space; if yes, they are amended. Next, their fitness is evaluated and compared with the global leader; if a better solution exists, then the leader of that Mapper (whale*) is updated. Subsequently, each search agent gets stored in HDFS to be used in the next MapReduce job. Finally, each Mapper emits its leader and leader's fitness (<whale*, fitness*>) of the current iteration.

In the traditional WOA, when a search agent is to be explored, it updates its position with respect to a randomly selected search agent from the entire population. However, in our approach, each Mapper operates only on a sub-population rather than the entire population, and if a search agent within a Mapper is to be explored, it will have access only to a chunk of the population. Exploring using sub-populations, where their sizes are big enough, is beneficial to the rate of convergence. All sub-populations search the entire search space independently, and each finds its own leader. From all the leaders found so far, the best one is extracted and used in the next iteration as a reference for all search agents in all Mappers to update their positions, thus enhancing exploration which as a result improves the rate of convergence.

Furthermore, while Mappers are processing search agents, those agents do not have access to the rest of the sub-population. This is because Mappers, in their natural behavior, process their input one record (search agent) at a time. Besides, in the case where search agents are to be explored, they should have access to the rest of the sub-population for efficient exploration. To overcome this issue, all search agents are stored in a temporary variable (defined by *totalWhales* in Algorithm 4) as they are being iterated over by the Mappers. Additionally, the search agents that are to be explored are stored in another temporary variable called *whalesToBeExplored* (Algorithm 4), and their exploration is delayed until the last search agent is processed. As after processing the Mapper's last search agent, then any search agent that needs to be explored can do so as by that time they have access to the entire sub-population.

After the mapper phase comes the reducer phase. In our approach, all MapReduce jobs use the same Reducer function, namely *Reduce* (Algorithm 5). First, the *Reduce* function receives all the leaders and their corresponding fitness (<whale*, fitness* >) emitted from all Mappers. Second, the leaders are iterated over to extract the global best leader and its corresponding fitness (<whale**, fitness** >). Next, the best solution agent gets emitted and stored in HDFS to be used in the next MapReduce job. At the end of the current MapReduce job,

the Driver function then takes over and checks whether the stopping criteria has been satisfied; if yes, the MR-WOA terminates, else another MapReduce job is created and the above process repeats. In our approach, we did not use a partitioner function as only one Reducer is utilized. A graphical representation of a single MR-WOA's iteration is shown in Figure 4.

Algorithm 4 Map function

Input: Key: *whale*, Value: *fitness*

Output: Key: whale*, Value: fitness*

```
1:  whale**, fitness** = Get global leader from HDFS
2:  totalWhales, whalesToBeExplored
3:  if (exploitation) then
4:      Apply exploitation
5:      Check if search space is exceeded and amend it
6:      Evaluate fitness
7:      Update leader if better fitness exists
8:      Store individual in HDFS
9:  else if (exploration) then
10:     Save individual into whalesToBeExplored
11: end if
12: Save individual into totalWhales
13: if (last individual) then
14:     for each (whalesToBeExplored) do
15:         Apply exploration
16:         Check if search space is exceeded and amend it
17:         Evaluate fitness
18:         Update leader if better fitness exists
19:         Store individual in HDFS
20:     end for
21:     Emit(whale*, fitness*)
22: end if
```

Algorithm 5 Reduce function

Input: Key: *whale**, Value: *fitness**

Output: Key: whale**, Value: fitness**

```
1: Iterate over all fitness values and select the best
2: Emit(whale**, fitness**)
```

The <key, value> types used for all inputs and outputs in MapReduce jobs are as follows: key is a whale object, which is an array of *DoubleWritable*, and value represents the whale's fitness of type *DoubleWritable*. *DoubleWritable* is a box class in Hadoop and is equivalent to the Java primitive type *double*.

To compute the complexity of the proposed algorithm MR-WOA, first, we need to compute the complexity of WOA. In the complexity analysis, P represents the population size, I_{max} represents the maximum number of iterations, and M represents the number of mappers. Looking at Algorithm 1, we can compute the complexity of WOA by analyzing the components of the algorithm as follows: The while loop (lines 4-22) is executed I_{max} times. Inside this while loop, the for loop (lines 5-17) is executed for each individual in the population, that is, it is executed P times. The fitness of each individual is computed in the population (line 19), which is outside the for loop, is O(P) also. Therefore, the worst case complexity of WOA is O(P.I_{max}). In the MR-WAO algorithm as shown in Figure 4, each mapper operates on P/M sub-population independently to generate its own P/M local search agents in each iteration. The reducer finds the best search agent among them to find the global best search agent. The parallelism of the MapReduce framework comes from the fact that each map or reduce operation can be executed on a separate processor independently of others. Therefore, the complexity of each MapReduce job is O(P/M), which represents a single iteration. The outputs from a reduce step are used as inputs to another round of MapReduce steps. Since MR-WOA iterates these steps I_{max} times, hence the worst case complexity of the algorithm is O(I_{max}.P/M).

5 | EXPERIMENTAL RESULTS AND ANALYSIS

5.1 | Environment setup

We implemented our proposed approach on a cluster with Java 1.8. Each computer in the cluster has 8-core CPU and 31.75GB memory. The MapReduce cluster is configured with Apache Hadoop 2.7.3 and consists of 80 nodes. A detailed system configuration of the used cluster can be found in Table 2.

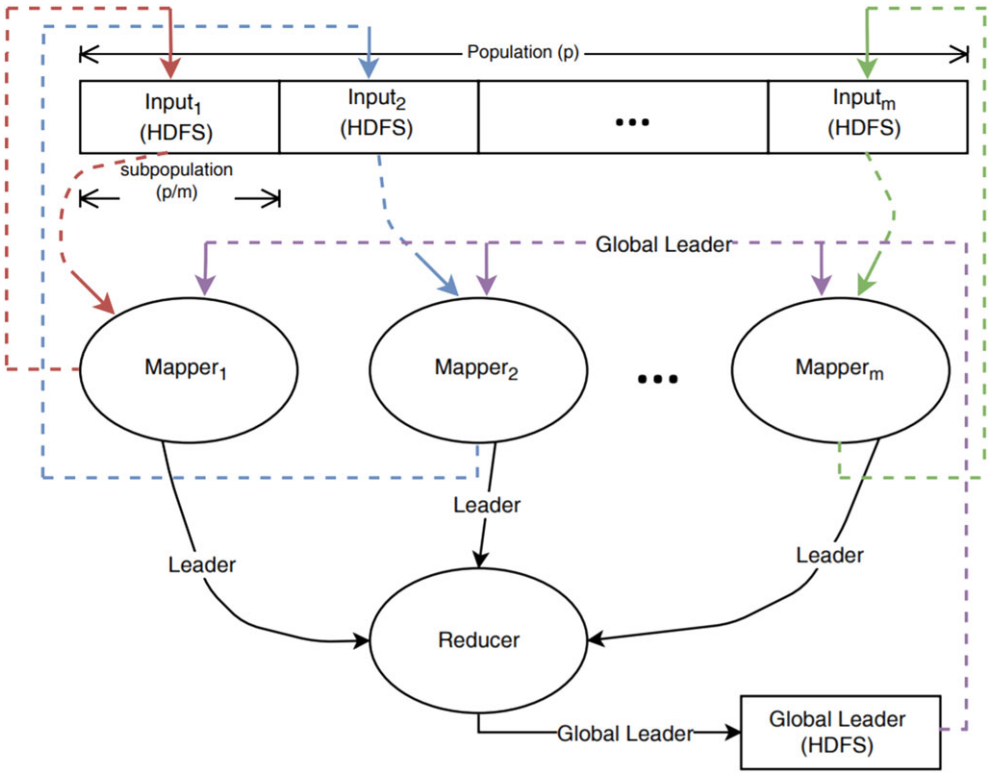

FIGURE 4 Single MR-WOA iteration

TABLE 2 System configuration

Environment	Cluster
Hadoop Distribution	Apache Hadoop 2.7.3
Java SDK	1.8.0
Instance Specification	Operating System: centos7 (x86_64) Memory: 31.75GB Storage: 3.55TB
Number of Nodes	80

5.2 | Experimental results and evaluation

Different sets of experiments are conducted for evaluating the performance of MR-WOA. Additionally, for all experiments, multiple benchmark functions are employed to confirm our approaches' performance in terms of rate of convergence, scalability, and speedup. The benchmarks employed are presented in Table A1, and their graphical representations are shown in Figure A1. In Table A1, dim represents the function's dimensions, range represents the limits of the search space, and f_{min} represents the global optima.

A wide range of reviewed benchmark functions[53,54] are suitable for testing the performance of meta-heuristic algorithms. The functions utilized in order to check the performance of MR-WOA are classified into two groups: high-dimension unimodal functions (f_1, f_2, and f_3) and high-dimension multimodal functions (f_4, f_5, and f_6). As their names imply, high-dimension unimodal functions contain only one global optimum, and they are suited for benchmarking exploitation. On the contrary, high-dimension multimodal functions have several optima, where one is global and the rest are local. This makes them more challenging than unimodal functions as the algorithm has to find the global optima without getting stuck in any of the local optima. Thus, multimodal functions are used to test the algorithm's exploration capability.

Several parameters need to be considered while evaluating the experiments. The parameters include number of whales (population), maximum number of iterations, convergence (fitness), computational time, and number of computational nodes. Moreover, results of the conducted experiments in this paper are an average of 30 independent runs due to the stochastic behavior of MR-WOA.

5.3 | Convergence analysis

In this experiment, we monitor the comparison of the best fitness value achieved vs number of iterations under different number of whales and different sets of benchmark functions. The results of this comparison are presented in Figures 5A-5F. For this analysis, the parameter defined by the number of computational nodes is kept constant at 2 nodes. The rate of convergence depends on two factors the complexity of the utilized benchmark and the number of whales. We can observe from Figures 5A-5F that with increasing number of iterations, the optimal output

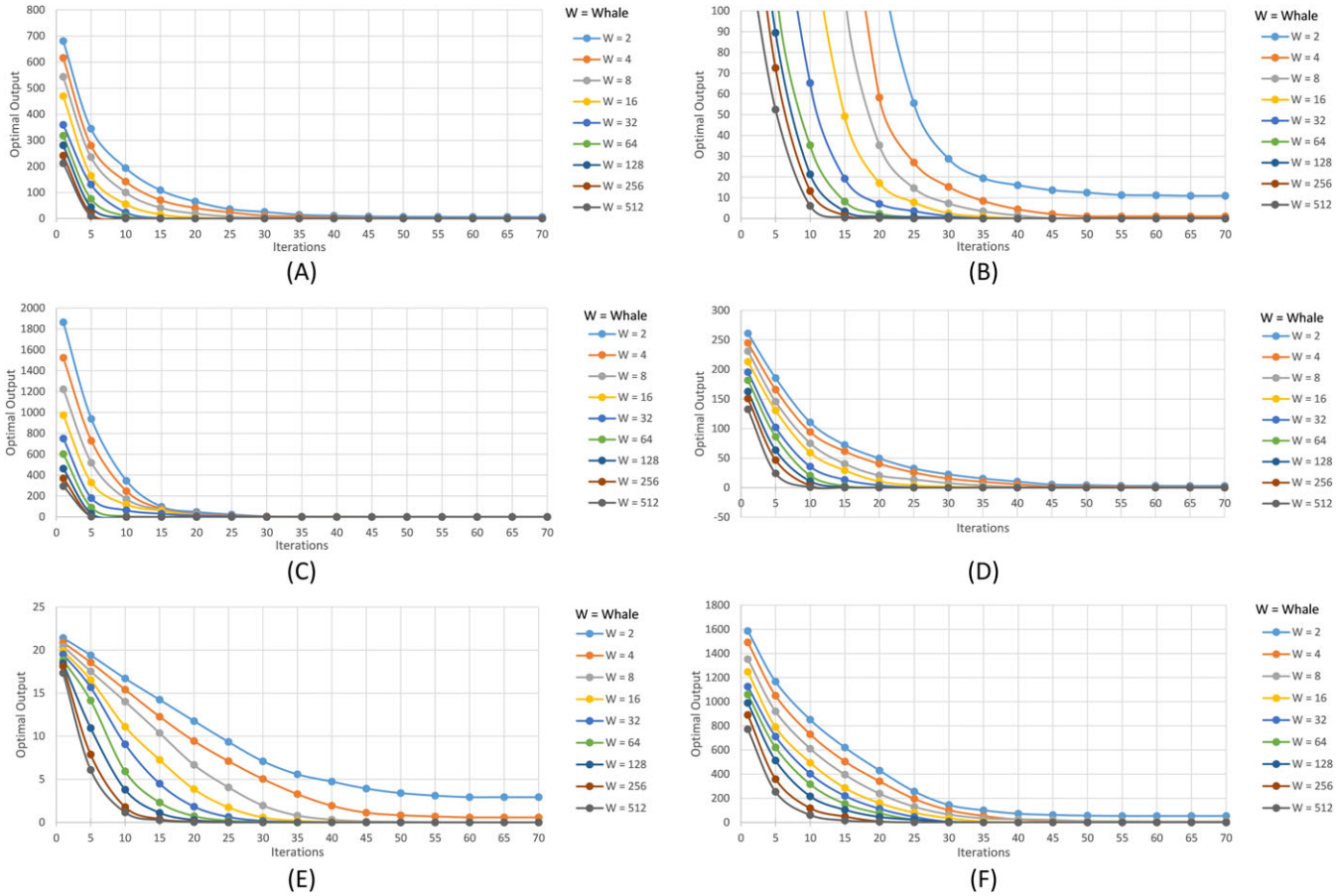

FIGURE 5 Output trace for the benchmark functions (f_1-f_6). A, Sphere (f_1); B, Schwefel2.22 (f_2); C, Quartic (f_3); D, Alpine (f_4); E, Ackley (f_5); E, Rastrigin (f_6)

gradually decreases and approaches the global best fitness value ($f_{\min}$). In addition, at a fixed iteration as the number of whales increases, the achieved optimal output gets closer to $f_{\min}$. The slowest rate of convergence in the Sphere (f_1) function, shown in Figure 5A, occurs after 53 iterations when the population is set to 2, and its fastest convergence rate occurs after 9 iterations when the population is set to 512. While in Ackley function (f_5), shown in Figure 5E, the slowest convergence rate is reached after 59 iterations when the population is 2, which is the slowest among all other benchmarks. However, when the population is set to 512, convergence is reached in 17 iterations. This verifies that convergence rate is dependent on the population size.

Convergence rate is also dependent on the complexity of the applied benchmark in terms of the number of local optima existing in the search space. The higher the number of local optima in a benchmark, then the more time will be needed to avoid getting stuck in all local optima and finding the global best solution. This behavior could be extracted from the figures below. When the population is set to 512, the Sphere function (f_1) belonging to the unimodal group reaches converges after 9 iterations. While in Alpine function, (f_4) belonging to the multimodal group reaches converges after 14 iterations. This shows that unimodal functions reach convergence at a faster rate than in multimodal functions, and thus, convergence rate is slower in more complex functions.

Furthermore, it can be noticed that when the population is very small, for some benchmark functions, the function gets stuck in a local optimum as it converges rather than $f_{\min}$. The population sizes that are small and has the highest probability in getting stuck in a local optimum are 2 and 4 due to their poor exploration capability. The number of whales in such populations are already small, and when they get distributed among computational nodes for processing, the population per node gets even smaller. This reduces the power of exploration enormously due to the lack of individuals for random selection. When the population is 2 and gets distributed among two Mappers, then each Mapper will process only one search agent, and so, no other search agent will be available for random selection. Same applies when the population is 4; each Mapper will process only two search agents, and if a search agent has to be explored, then it has only one other individual to select randomly. On the other hand, for the rest of the population sizes, MR-WOA is capable of escaping all local optima and finding $f_{\min}$ due to their high powers in exploration. Precision and accuracy of our proposed approach in finding the best global fitness value is outstanding when compared with the benchmark's $f_{\min}$. The achieved optimal output for any population, except 2 and 4, is 0, which is the benchmark's $f_{\min}$. In some cases, the achieved optimal output is not 0 but is as low as 0.00005 (0.005%). Another reason behind the high precision and accuracy is the high dimensions of the benchmark functions.

Among all benchmark functions, the convergence rate study shows that 59 iterations is sufficient for the algorithm to produce challenging results. The slowest convergence rate occurs in the Ackley (f_5) function after 59 iterations, and this occurs when the population is smallest (2). Therefore, for the rest of the experiments, the number of iterations will be fixed at 59 as, by that iteration, convergence would have been reached regardless

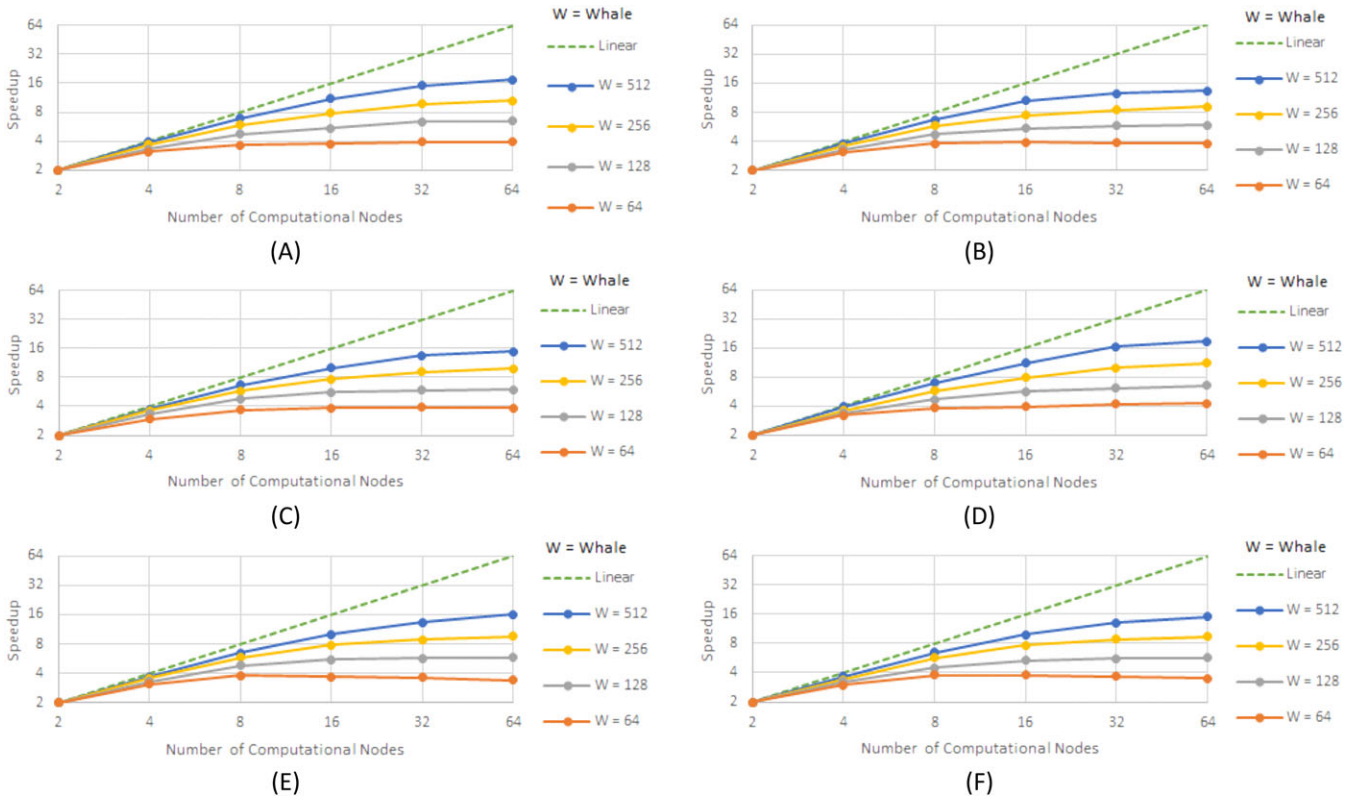

FIGURE 6 Speedup for the benchmark functions (f_1-f_6). A, Sphere (f_1); B, Schwefel2.22 (f_2); C, Quartic (f_3); D, Alpine (f_4); E, Ackley (f_5); F, Rastrigin (f_6)

of both the applied benchmark function and the population. The performance of our MR-WOA in terms of rate of convergence is very competitive when compared with the traditional WOA and other state-of-the-art EAs. Our rate of convergence defined by the number of iterations at which convergence is reached is superior because individuals within the population are forced to explore from a sub-population rather than the whole population, allowing each sub-population to search the entire search space independently, leading to faster convergence rate.

5.4 | Speedup analysis

In this experiment, speedup is calculated from the running time of different population sizes under various number of computational nodes. The speedup is calculated by Equation (8)

$$Speedup = \frac{T_2}{T_n}, \tag{8}$$

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

Figures 6A-6F present the speedup of all benchmark functions for different number of whales with increasing number of computational nodes. The number of iterations is kept constant at 59 based on the previous experiment's results. The straight green line in the speedup figures exhibits the ideal speedup against the increase in the number of computational nodes. The performance of MR-WOA in terms of speedup is defined by the closeness of the achieved speedup to the ideal speedup. The higher the speedup means the better utilization of the parallel computational nodes.

As shown in the Figures, there are minimal differences in the achieved speedup for all benchmark functions. As mentioned before, this is due to the complexities of all benchmark functions falling in the same range. Alpine (f_4), shown in Figure 6D, achieves the best speedup among all other benchmarks. The highest achieved speedup is 18.82, which occurs when the population size is set to 512, and the number of computational nodes is 64. Moreover, it is noticed that when the population is very small and with a low number of computational nodes, the achieved speedup is acceptable; however, as the number of computational nodes increase, the speedup worsens. This is because when the population is small and gets distributed among the computational nodes, each Mapper will only process a sub-population, which is even smaller than the entire population. Thus, the time needed to initialize the Mappers exceeds the actual time needed per Mapper to process the sub-populations, hence limiting the speedup.

As the population size increases, the achieved speedup increases too, and even if the number of computational nodes increases, the speedup will not be limited. This is because the number of whales per Mapper is big enough that the processing time needed, per Mapper, exceeds the initialization time. To verify this, from the Figures, it can be noticed that at a fixed number of computational nodes the speedup rises with the increase in number of whales. For example, when the number of computational nodes is 64 for the Rastrigin (f_6) function and the population size is set to 64, 128, 256, and 512, the achieved speedup is 3.50, 5.70, 9.35, and 15.61, respectively. This growth in speedup with the increase in the number of computational nodes confirms the scalability of our approach.

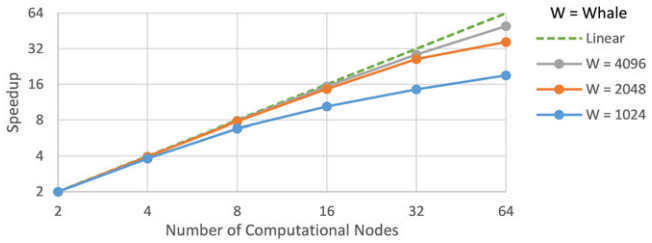

FIGURE 7 Speedup of Alpine function under large population

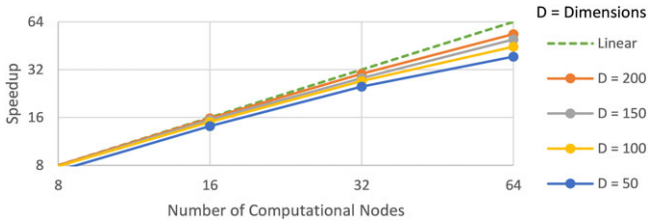

FIGURE 8 Speedup of Alpine function as dimensions increase

5.5 | Speedup analysis for large population

In the previous experiment, we saw that with the increase in number of whales the speedup increases. The function that achieved the highest speedup is Alpine function (f_4). Thus, in this experiment, we will build on the previous experiments by keeping the parameters fixed and measuring the speedup of Alpine function with even bigger population sizes (greater than 512). Figure 7 shows the speedup of Alpine function under large populations. The speedup of our approach still increases with larger populations. The highest speedup achieved is 49.65 when the population is set to 4096 and ran on 64-computational nodes.

5.6 | Speedup analysis for varying dimensions

Another set of experiment was done to examine the effect of increasing the dimensional size on the Alpine benchmark function with respect to speedup. The dimensions tested are 50, 100, 150, and 200. Building on our previous experiments, the whale population is fixed to 4096, number of iterations set to 59 and running it on 64-computational nodes. Results of the experiment are shown in Figure 8. It can be noticed that, as dimensions of the benchmark function increase, the speedup increases too. The increase in dimensions results in the increase of the variables to be optimized, which leads in increasing the running time. The longer running time means lower performance; however, if the job is being shared and executed all in parallel, then the running time will reduce by the parallelization factor. If this experiment is to be executed on fewer number of nodes, then the running time will be higher. The achieved speedups are not perfectly linear as there are several factors that limit the speedup performance such as the network congestion, assignment of tasks to mappers, mapper's failure, transfer of data across the network, and randomness used in the proposed algorithm's for searching purposes. The highest achieved speedup is 53.65 when the dimensions is set to 200.

5.7 | Discussion

A discussion of the experimental results is considered in this subsection. MR-WOA has been tested over 6 different benchmark functions, belonging to two groups. It is observed that the proposed algorithm reaches convergences after a maximum of 59 iterations, regardless of the parameters set. The occurrence of convergence after a small number of iterations reflects on the outstanding search ability of the MR-WOA. Moreover, the proposed algorithm shows excellent accuracy as the achieved results are very close to the optimal solutions of the benchmark functions (f_{min}). These achievements are because of WOA exploration search capability and the two approaches of exploitation implemented in the algorithm granting it the ability to avoid local optima and reach the global optimal solution. Attaining accurate results after short number of iterations proves the ability of MR-WOA to provide a good switch between the exploration and exploitation phases during the optimization process. In addition, MR-WOA has small control parameters, which maintains the low level of complexity, as often regulating parameters can be more complex than the problem itself. Other reasons leading for these outstanding results are: the random solutions used for leading the search in WOA, existence of adaptive mechanisms enabling the algorithm to accelerate exploitation proportional to the number of iterations.

The scalability and speedup of MR-WOA was established due to the ability to parallelize some parts of the WOA algorithm and load sharing across the computation nodes. These also enhanced the search ability as each sub-population devoted to a node was responsible for searching the entire search space and finding their own optimal solution, reducing the risk of getting stuck in a local minimum. Speedup results are shown in Figures 6A-6F. Further experiments were done to check the consequence of increasing the load and the dimensions of the system on the speedup. Increasing the population size of the Alpine function from 512 to 4096 elevated the speedup from 18.82 to 49.65. Moreover, building on top of the previous experiment by raising the dimensions to 200 results in pushing the speedup further to 53.65. This shows that there is room for speedup improvement with bigger problems.

6 | CONCLUSION

In this paper, we present a distributed approach for the whale optimization algorithm based on MapReduce framework (MR-WOA). Different sets of experiments were conducted to illustrate the effectiveness of our approach. In the first experiment, we analyzed both the rate and precision of convergence. From this experiment, the best iteration number was picked to be used for setting the number of iterations in the rest of the experiments. Next, the speedup was measured and analyzed with respect to the different number of whales and different number of computational nodes. The speedup experiments showed that MR-WOA is scalable. In the last experiment, the population sizes were increased to see the effect of large population sizes on speedup. The best achieved speedup was up to 53.65 when our proposed approach ran on 64-computational nodes, the population is set to 4096 dimensions set to 200 and for 59 iterations. These experiments revealed that MR-WOA scales very well with increasing number of computational nodes and achieved a speedup that is close to the linear speedup. The future work includes measuring the effectiveness of MR-WOA on a range of much harder problems such as TSP and developing a Spark-based distributed implementation of WOA to further enhance its scalability as compared with MR-WOA. Spark is expected to give a higher scalability than MapReduce as it processes data in-memory while MapReduce stores data on disk after running each job.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their valuable and thoughtful comments which definitely improved overall quality of the manuscript.

ORCID

Yasser Khalil http://orcid.org/0000-0002-6632-6068

REFERENCES

1. Goldberg DE. *Genetic Algorithms in Search, Optimization and Machine Learning*. 1st ed. Boston, MA: Addison-Wesley Longman Publishing; 1989.
2. Holland JH. Genetic algorithms. *Sci Am*. 1992;267(1):66-72.
3. Eiben AE, Smith JE. *Introduction to Evolutionary Computing*. Berlin, Germany: Springer-Verlag Berlin Heidelberg; 2003.
4. Koza JR. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA; Massachusetts Institute of Technology; 1992.
5. Černý V. Thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm. *J Optim Theory Appl*. 1985;45(1):41-51.
6. Kirkpatrick S, Gelatt Jr CD, Vecchi MP. Optimization by simulated annealing. *Science*. 1983;220(4598):671-680.
7. Kennedy J, Eberhart R. Particle swarm optimization. In: Proceedings of the IEEE International Conference on Neural Networks; 1995; Perth, Australia.
8. Dorigo M. Optimization, learning and natural algorithms [PhD thesis]. Milano, Italy: Politecnico di Milano; 1992.
9. Dorigo M, Birattari M, Stutzle T. Ant colony optimization. *IEEE Comput Intell Mag*. 2006;1(4):28-39.
10. Mirjalili S, Lewis A. The whale optimization algorithm. *Adv Eng Softw*. 2016;95:51-67.
11. Aziz MAE, Ewees AA, Hassanien AE. Whale optimization algorithm and moth-flame optimization for multilevel thresholding image segmentation. *Expert Syst Appl*. 2017;83(C):242-256.
12. Hassanien AE, Elfattah MA, Aboulenin S, Schaefer G, Zhu SY, Korovin I. Historic handwritten manuscript binarisation using whale optimisation. Paper presented at: 2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC); 2016; Budapest, Hungary.
13. Mostafa A, Hassanien AE, Houseni M, Hefny H. Liver segmentation in MRI images based on whale optimization algorithm. *Multimed Tools Appl*. 2017;76(23):24931-24954.
14. Aljarah I, Faris H, Mirjalili S. Optimizing connection weights in neural networks using the whale optimization algorithm. *Soft Comput*. 2018;22(1):1-15.
15. Mafarja MM, Mirjalili S. Hybrid whale optimization algorithm with simulated annealing for feature selection. *Neurocomputing*. 2017;260:302-312.
16. Oliva D, Aziz MAE, Hassanien AE. Parameter estimation of photovoltaic cells using an improved chaotic whale optimization algorithm. *Appl Energy*. 2017;200:141-154.
17. Tharwat A, Moemen YS, Hassanien AE. Classification of toxicity effects of biotransformed hepatic drugs using whale optimized support vector machines. *J Biomed Inform*. 2017;68:132-149.
18. Dao T-K, Pan T-S, Pan J-S. A multi-objective optimal mobile robot path planning based on whale optimization algorithm. Paper presented at: 2016 IEEE 13th International Conference on Signal Processing (ICSP); 2016; Chengdu, China.
19. Ling Y, Zhou Y, Luo Q. Lévy flight trajectory-based whale optimization algorithm for global optimization. *IEEE Access*. 2017;5:6168-6186.
20. Abdel-Basset M, Abdle-Fatah L, Sangaiah AK. An improved Lévy based whale optimization algorithm for bandwidth-efficient virtual machine placement in cloud computing environment. *Clust Comput*. 2018:1-16.
21. Zhang C, Fu X, Leo LP, Peng S, Xie M. Synthesis of broadside linear aperiodic arrays with sidelobe suppression and null steering using whale optimization algorithm. *IEEE Antennas Wirel Propag Lett*. 2018;17(2):347-350.
22. Abdel-Basset M, El-Shahat D, El-henawy I, Sangaiah AK, Ahmed SH. A novel whale optimization algorithm for cryptanalysis in Merkle-Hellman cryptosystem. *Mob Netw Appl*. 2018:1-11.

23. Harikarthik SK, Palanisamy V, Ramanathan P. Optimal test suite selection in regression testing with testcase prioritization using modified Ann and Whale optimization algorithm. *Clust Comput*. 2017:1-10.
24. Wang J, Du P, Niu T, Yang W. A novel hybrid system based on a new proposed algorithm–multi-objective whale optimization algorithm for wind speed forecasting. *Appl Energy*. 2017;208:344-360.
25. Raj S, Bhattacharyya B. Optimal placement of TCSC and SVC for reactive power planning using whale optimization algorithm. *Swarm Evol Comput*. 2017;40:131-143.
26. Hasanien HM. Performance improvement of photovoltaic power systems using an optimal control strategy based on whale optimization algorithm. *Electr Power Syst Res*. 2018;157:168-176.
27. Gong Y-J, Chen W-N, Zhan Z-H, et al. Distributed evolutionary algorithms and their models: a survey of the state-of-the-art. *Appl Soft Comput*. 2015;34:286-300.
28. Tan Y, Ding K. A survey on GPU-based implementation of swarm intelligence algorithms. *IEEE Trans Cybern*. 2016;46(9):2028-2041.
29. Zhan Z-H, Liu X-F, Zhang H, et al. Cloudde: a heterogeneous differential evolution algorithm and its distributed cloud version. *IEEE Trans Parallel Distributed Syst*. 2017;28(3):704-716.
30. Schikuta E. Message-passing-interface-forum: MPI: a message-passing interface standard. Techn. Ber. Knoxville, Tennesee: University of Tennessee; 1994.
31. Board O. OpenMP application program interface (version 4.5). Technical Report. OpenMP; 2015.
32. Sunderam VS. PVM: a framework for parallel distributed computing. *Concurrency Computat Pract Exp*. 1990;2(4):315-339.
33. Lämmel R. Google's MapReduce programming model-revisited. *Sci Comput Program*. 2008;70(1):1-30.
34. Polato I, Ré R, Goldman A, Kon F. A comprehensive view of Hadoop research-a systematic literature review. *J Netw Comput Appl*. 2014;46:1-25.
35. Borthakur D. The Hadoop distributed file system: Architecture and design. *Hadoop Proj Website*. 2007;11(2007):21.
36. White T. *Hadoop: The Definitive Guide*. Sebastopol, CA: O'Reilly Media; 2012.
37. Gandomi AH, Alavi AH. Krill herd: a new bio-inspired optimization algorithm. *Commun Nonlinear Sci Numer Simul*. 2012;17(12):4831-4845.
38. McNabb AW, Monson CK, Seppi KD. Parallel PSO using MapReduce. Paper presented at: 2007 IEEE Congress on Evolutionary Computation (CEC); 2007; Singapore, Singapore.
39. Chang J-C. Modified particle swarm optimization for solving traveling salesman problem based on a Hadoop MapReduce framework. Paper presented at: 2016 International Conference on Applied System Innovation (ICASI); 2016; Okinawa, Japan.
40. Ghosh PS. Parallelization of particle swarm optimization algorithm using Hadoop Mapreduce. *Circulation*. 2016;701:8888.
41. Di Martino S, Ferrucci F, Maggio V, Sarro F. Towards migrating genetic algorithms for test data generation to the cloud. In: Tilley S, Parveen T, eds. *Software Testing in the Cloud: Perspectives on an Emerging Discipline*. Hershey, PA: IGI Global; 2012.
42. Geronimo LD, Ferrucci F, Murolo A, Sarro F. A parallel genetic algorithm based on Hadoop MapReduce for the automatic generation of JUnit test suites. Paper presented at: 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation; 2012; Montreal, Canada.
43. Jin C, Vecchiola C, Buyya R. MRPGA: an extension of MapReduce for parallelizing genetic algorithms. Paper presented at: 2008 IEEE Fourth International Conference on eScience (ESCIENCE); 2008; Indianapolis, IN.
44. Verma A, Llorà X, Goldberg DE, Campbell RH. Scaling genetic algorithms using MapReduce. Paper presented at: 2009 Ninth International Conference on Intelligent Systems Design and Applications; 2009; Pisa, Italy.
45. Ludwig SA. Running krill herd algorithm on Hadoop: a performance study. Paper presented at: 2016 IEEE Congress on Evolutionary Computation (CEC); 2016; Vancouver, Canada.
46. Çolak ME, Varol A. Optimization with MapReduce and CWW algorithm. *Int J Adv Electron Comput Sci*. 2016;3(9):78-81.
47. Wu B, Wu G, Yang M. A MapReduce based ant colony optimization approach to combinatorial optimization problems. Paper presented at: 2012 8th International Conference on Natural Computation; 2012; Chongqing, China.
48. Al-Madi N, Aljarah I, Ludwig SA. Parallel glowworm swarm optimization clustering algorithm based on MapReduce. Paper presented at: 2014 IEEE Symposium on Swarm Intelligence; 2014; Orlando, FL.
49. Aljarah I, Ludwig SA. Parallel particle swarm optimization clustering algorithm based on MapReduce methodology. Paper presented at: 2012 Fourth World Congress on Nature and Biologically Inspired Computing (NaBIC); 2012; Mexico City, Mexico.
50. Banharnsakun A. A MapReduce-based artificial bee colony for large-scale data clustering. *Pattern Recogn Lett*. 2017;93:78-84.
51. Ludwig SA. MapReduce-based fuzzy c-means clustering algorithm: implementation and scalability. *Int J Mach Learn Cybern*. 2015;6(6):923-934.
52. Judith JE, Jayakumari J. Distributed document clustering analysis based on a hybrid method. *China Commun*. 2017;14(2):131-142.
53. Jamil M, Yang X-S. A literature survey of benchmark functions for global optimisation problems. *Int J Math Model Numer Optimisation*. 2013;4(2):150-194.
54. Yang X-S. Appendix A: test Problems in Optimization. In: Yang X-S, ed. *Engineering Optimization: An Introduction with Metaheuristic Applications*. Hoboken, NJ: John Wiley & Sons; 2010;261-266.

How to cite this article: Khalil Y, Alshayeji M, Ahmad I. Distributed Whale Optimization Algorithm based on MapReduce. *Concurrency Computat Pract Exper*. 2018;e4872. https://doi.org/10.1002/cpe.4872

APPENDIX

BENCHMARK FUNCTIONS AND GRAPHS

TABLE A1 Benchmark functions

Name	Function	Dim	Range	f_{min}
Sphere	$f_1(x) = \sum_{i=1}^{n} x_i^2$	100	[-5.12,5.12]	0
Schwefel2.22	$f_2(x) = \sum_{i=1}^{n} \lvert x_i \rvert + \prod_{i=1}^{n} \lvert x_i \rvert$	100	[-10,10]	0
Quartic	$f_3(x) = \sum_{i=1}^{n} x_i^4 + \text{random}(0, 1)$	100	[-1.28,1.28]	0
Alpine	$f_4(x) = \sum_{i=1}^{n} \lvert x_i \sin(x_i) + 0.1x_i \rvert$	100	[-10,10]	0
Ackley	$f_5(x) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n} \cos(2\pi x_i)\right) + 20 + e$	100	[-32,32]	0
Rastrigin	$f_6(x) = \sum_{i=1}^{n} \left[x_i^2 - 10\cos(2\pi x_i) + 10\right]$	100	[-5.12,5.12]	0

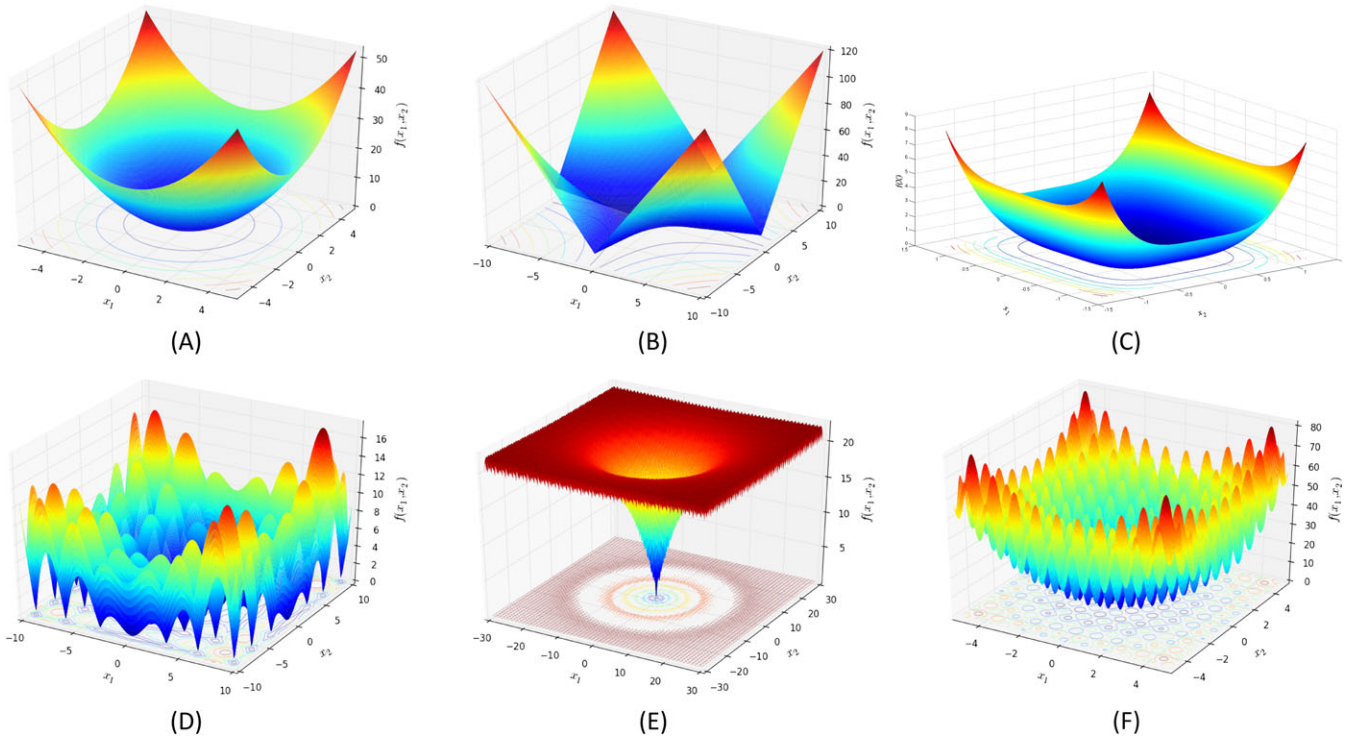

FIGURE A1 Graphs of the benchmark functions (f_1-f_6). A, Sphere (f_1); B, Schwefel2.22 (f_2); C, Quartic (f_3); D, Alpine (f_4); E, Ackley (f_5); F, Rastrigin (f_6)