A Modified Cuckoo Search Algorithm for Data Clustering

Preeti Pragyan Mohanty, Institute of Technical Education and Research, Siksha ‘O’ Anusandhan University, Bhubaneswar, India*
Subrat Kumar Nayak, Institute of Technical Education and Research, Siksha ‘O’ Anusandhan University, Bhubaneswar, India

ABSTRACT

Clustering of data is one of the necessary data mining techniques, where similar objects are grouped in the same cluster. In recent years, many nature-inspired clustering techniques have been proposed, which have led to some encouraging results. This paper proposes a modified cuckoo search (MoCS) algorithm. In this work, an attempt has been made to balance the exploration of the cuckoo search (CS) algorithm and to increase the potential of the exploration to avoid premature convergence. This algorithm is tested using 15 benchmark test functions and is proven as an efficient algorithm in comparison to the CS algorithm. Further, this method is compared with well-known nature-inspired algorithms such as ant colony optimization (ACO), artificial bee colony (ABC), particle swarm optimization (PSO), particle swarm optimization with age group topology (PSOAG), and CS algorithm for clustering of data using six real datasets. The experimental results indicate that the MoCS algorithm achieves better results as compared to other algorithms in finding optimal cluster centers.

KEYWORDS

Cuckoo Search Algorithm, Data Clustering, Intra-Cluster Distance

1. INTRODUCTION

Clustering is a method of grouping an enormous amount of data into different groups. The data in the same group exhibit similar properties, while the data in different groups exhibit different properties. This technique is used for finding patterns among the data in each group. Over the past few years, clustering has played a key role in various fields of research such as image analysis, machine learning, data mining, pattern recognition, information retrieval, statistics, biology, medical sciences, market research, etc. (Ahalya & Pandey, 2015).

The traditional clustering algorithms are mostly classified into two prime types, i.e., hierarchical clustering and partitional clustering (Leung et al., 2000; Xu & Tian, 2015). The outcome of the hierarchical clustering approach is a tree-like structure representing the clustering process, where the dataset is partitioned into different groups. In this type of clustering, the data objects present in one group cannot be reassigned to another group (Armano & Farmani, 2016). Moreover, this clustering can be performed even if the number of groups is not known. The major disadvantage of this technique is that it fails to separate the overlapping groups as the information about the shape and size of groups...
This paper deals with the partitional clustering problem, where the number of clusters of a dataset is known beforehand. The partitional clustering approach aims to optimize some dissimilarity criteria such as minimizing the intra-cluster distance between the objects in one group and maximizing the inter-cluster distance between different groups.

Generally, partitional clustering algorithms consist of centroid-based algorithms. One of the popular centroid-based algorithms is the \( k \)-means algorithm proposed by Stuart Lloyd in 1957 (Lloyd, 1982). The main aim of the \( k \)-means clustering algorithm is to partition the objects into \( k \) groups by randomly choosing \( k \) number of data objects as initial centroids. Though this algorithm is easy to implement, still it gets stuck at the local minimum as it is sensitive to the initial position of the groups.

The nature-inspired metaheuristic algorithms such as Firefly Algorithm (FA) (Yang, 2008), Ant Colony Optimization (ACO) (Dorigo, 1992), Cuckoo Search (CS) (Yang & Deb, 2009), Particle Swarm Optimization (PSO) (Kennedy & Eberhart, 1995), Glowworm Swarm Optimization (GSO) (Krishnanand & Ghose, 2005), Artificial Bee Colony (ABC) (Karaboga et al., 2005), etc. have been able to improve upon the disadvantages of \( k \)-means algorithm. These algorithms are used to solve optimization problems that do not have a specific satisfactory solution and generate near-optimal results (Nesmachnow, 2014). In these algorithms, a population of candidate solutions is randomly generated and the best population for the next generation is selected based on some fitness values. These algorithms simultaneously optimize these candidate solutions to generate a globally optimized solution (Jiang et al., 2013).

Many researchers have proposed many variations of nature-inspired metaheuristic algorithms for clustering, which have resulted in inefficient solutions. It is impossible to cover all the algorithms proposed for clustering. In this paper, we confine our literature to some of the swarm intelligence algorithms proposed for clustering. PSO is a swarm intelligence algorithm inspired by a flocking of birds or swarm of fish. In this algorithm, the particles move towards their best previous positions and towards the best particle in order to achieve the global optimal solution. The PSO based clustering approach proposed by Cura (2012) follows this technique and achieves better outcomes on the basis of objective function values, error rate and computation time when compared to other swarm-intelligence techniques. The Ant Colony Optimization (ACO) for clustering the data into \( k \) clusters was proposed by Shelokar et al. (2004). This method involves distributed agents that imitate the way real ants go for searching their food from their nest. This algorithm was tested on different datasets yielding better computational solutions in less time. Zhang et al. (2010) used the ABC algorithm for clustering. This algorithm involves employed bees, onlooker bees and scout bees who get involved in the process of searching for their food. In this paper, Deb’s method is adapted for selecting a food source instead of the greedy search process for selecting the food source. This method gives encouraging results when tested with different datasets. A recently developed algorithm and a variation of the PSO algorithm, called Particle Swarm Optimization with Age Group topology (PSOAG) is used for clustering of data (Jiang et al., 2013). In this method, an approach for maintaining population diversity is defined. The PSOAG algorithm proves as an effective algorithm for data clustering as it has better intra-cluster distance, better clustering accuracy and low computation time as compared to PSO, ACO, ABC, and DE algorithm.

Saida et al. (2014) used the CS algorithm for clustering of data. CS algorithm is based on an aggressive breeding strategy of cuckoos laying their eggs in the nest of other birds. The cuckoo birds go for searching their food with the help of a random walk method called the Levy flight approach. The Levy Flight approach is a random walk in step lengths with a high tailed probability distribution. Also, these cuckoo eggs when discovered by the host birds are either thrown away or the nest is abandoned depending on a certain probability. Although the CS algorithm is good at exploring the search space, still it is slow in exploiting the solutions (Long et al., 2014). Hence, a Modified Cuckoo Search (MoCS)
algorithm is proposed in this paper to balance the exploration and exploitation of the search space and improve the convergence performance of CS algorithm. Here, a novel method has been suggested that automatically updates the potential of exploration to avoid premature convergence. Moreover, the efficiency of MoCS algorithm for numerical optimization is tested using fifteen benchmark test functions. Also, the MoCS algorithm is used for clustering and the results are compared with CS, PSO, ACO, ABC and PSOAG algorithms based on six standard real datasets.

The rest of the paper is organized as follows. In Section 2, the clustering problem is presented. In Section 3, the basic Cuckoo Search algorithm is described. Subsequently, in Section 4, the detailed procedure of the Modified Cuckoo Search algorithm (MoCS) is presented. The experimental results using the benchmark test functions and six standard real datasets to check the efficiency of the MoCS algorithm are presented in Section 5. Finally, in Section 6, a conclusion is presented.

2. FORMULATION OF THE CLUSTERING PROBLEM

The clustering of data can be achieved on the basis of some similarity criterion. The similarity of data can be determined with the help of distance measuring techniques. There are various distance measures in use. The most popular among them are Manhattan distance, Minkowski distance, and Euclidean distance, etc. In this paper, Euclidean distance (Nayak et al., 2017) is used as a distance measurement technique which can be represented as follows:

\[
D(p_i, p_j) = \left\| p_i - p_j \right\| = \sqrt{\sum_{s=1}^{d} (p_{is} - p_{js})^2}
\]  

(1)

Here, \(p_i\) and \(p_j\) are the two objects, \(d\) is the number of attributes of each object and \(D(p_i, p_j)\) is the Euclidean distance between two objects.

The data clustering problem can be considered as a global optimization problem. This can be represented as follows (Nayak et al., 2017):

\[
\min_{Z, M} f(Z, M) = \sum_{k=1}^{K} \sum_{i=1}^{n} m_{ik} D(y_i, z_k)
\]  

(2)

where, \(M = \{m_{ik} \mid i = 1, 2, \ldots, n, k = 1, 2, \ldots, K\}\) and \(Z = \{z_k \mid k = 1, 2, \ldots, K\}\) such that:

\[
\sum_{k=1}^{K} m_{ik} = 1, \quad i = 1, \ldots, n
\]  

(3)

\[
\sum_{i=1}^{n} m_{ik} \geq 1, \quad k = 1, \ldots, K
\]  

(4)

Here, \(n\) denotes the total number of objects, and \(K\) denotes the number of groups. \(m_{ik} = 1\) denotes the presence of \(i^{th}\) data object in the \(k^{th}\) group, \(m_{ik} = 0\) denotes the absence of \(i^{th}\) data object in the \(k^{th}\) group and \(D(y_i, z_k)\) denotes the Euclidean distance between \(i^{th}\) object \(y_i\) and \(k^{th}\) cluster centroid \(z_k\).

Let \(Y = \{y_1, y_2, \ldots, y_n\}\) represent a dataset which is partitioned into \(K\) groups satisfying the following criteria (Nanda & Panda, 2014):
Here, \( C_i \) represents the \( i^{th} \) group out of \( K \) groups. The members of the \( i^{th} \) group can be found by the following equation (Nayak et al., 2017):

\[
C_i = \{ y_p \mid \| y_p - z_i \| \leq \| y_p - z_j \|, y_p \in Y, i \neq j, j = 1,2,\ldots,K \}
\] (8)

Here, \( z_i \) is the centroid of a cluster \( C_i \) and \( z_j \) is the centroid of the cluster \( C_j \). The object \( y_p \) is a member of the cluster \( C_i \) if its Euclidean distance to \( z_i \) is less than or equal to any other cluster centroid. The centroid of the cluster \( C_i \), i.e., \( z_i \) can be calculated by the following equation:

\[
z_i = \frac{1}{|C_i|} \sum_{y_p \in C_i} y_p
\] (9)

Hence, \( z_i \) is the mean of all members of the cluster \( C_i \) and represents its new centroid.

3. BASIC CUCKOO SEARCH ALGORITHM

CS is a nature-inspired metaheuristic algorithm. It is inspired by the aggressive breeding strategy of cuckoo species laying their eggs in the nests of other host birds. These cuckoo eggs, when discovered by host birds, are either thrown away or destroyed. The cuckoo nests are represented as solutions to any problem. Moreover, when these cuckoo eggs hatch earlier, those chicks can imitate the sound of host chicks to get more food. So, this reduces the chance of cuckoo eggs being discovered by host birds and being abandoned. The CS algorithm is based on three assumptions (Yang & Deb, 2009). First, the cuckoo lays one egg in any randomly chosen nest at a time. Secondly, the best nests with better fitness values are chosen for the next iteration. Thirdly, the number of nests is fixed and the egg laid by a cuckoo is discovered by a host bird with a certain probability. The advantage of the CS algorithm is that it is efficient and takes fewer parameters as compared to other metaheuristic algorithms.

The steps of the CS algorithm are described as follows:

**Step 1: Initialization**

The CS algorithm involves the initialization of three essential parameters, number of cuckoos \( N \), abandon probability of cuckoo \( p_a \) and maximum iteration of cuckoos \( m_{\text{iteration}} \). Along with these parameters, the step size \( \alpha \), levy flight component \( \beta \), iteration \( \text{iter} \), and dimension \( d \) are defined. The CS algorithm also involves a search space for each cuckoo with the upper boundary \( u = \{ y_u^1, y_u^2, \ldots, y_u^d \} \) and lower boundary \( l = \{ y_l^1, y_l^2, \ldots, y_l^d \} \).
Step 2: Random initialization of host nests

The nests for the $N$ host birds at $\text{iter} = 1$ are randomly initialized by the following equation:

$$y_i^d = \text{rand}() \times (u^d - l^d) + l^d \quad (10)$$

Here, $i = 1,2,\ldots,N$.

Step 3: Generation of new solution by levy flight approach using Mantenga’s algorithm

The new generation of solutions is generated by the levy flight approach iteratively. Levy flight is a random walk with step lengths. The advantage of this levy flight approach is that it is efficient in exploring large search spaces for an optimization algorithm and arrives at the solution faster. The levy flight component is also responsible for the exploration process to take place as it involves large step lengths in order to search for potential solutions.

The new solutions for the next generation can be obtained by the following equation (Cheng et al., 2018):

$$y_{i}^{\text{iter+1}} = y_{i}^{\text{iter}} + \alpha \cdot (y_{\text{best}}^{\text{iter}} - y_{i}^{\text{iter}}) \otimes \text{levy}(s, \beta) \quad (11)$$

Here, $y_{i}^{\text{iter+1}}$ refers to solutions for the next generation at iteration $\text{iter} + 1$, $y_{i}^{\text{iter}}$ refers to the current solution at iteration $\text{iter}$, $\alpha$ is the step size which is usually taken as 1, $\otimes$ refers to the entry wise product, $y_{\text{best}}^{\text{iter}}$ is the best solution obtained till the current iteration, $\text{levy}(s, \beta)$ is the random walk of the levy flight approach. In the levy flight approach, the levy step size $s$ can be determined by Mantenga’s algorithm. This can be represented by the following equation as follows (Yang, 2014):

$$s = \frac{u}{|u|^{1/\beta}} \quad (12)$$

Here, $u$ and $v$ are two Gaussian distributions which can be determined as follows:

$$u - N(0, \sigma^2) \quad v - N(0, 1) \quad (13)$$

Here, $u - N(0, \sigma^2)$ means the samples are from Gaussian normal distribution with 0 mean and variance $\sigma^2$. The variance can be determined as follows:

$$\sigma^2 = \left[ \frac{\Gamma(1 + \beta)}{\beta \Gamma((1 + \beta) / 2)} \cdot \frac{\sin(\pi \beta / 2)}{2^{(\beta-1)/2}} \right]^{1/\beta} \quad (14)$$

Here, $\Gamma$ is the gamma function and $\beta$ lies between 1 and 3.

The fitness values of these new solutions are then obtained by any objective function, denoted by $f_j$. A nest denoted as $j$ is then chosen randomly among these nests and its fitness can be represented
as \( f_j \). If the fitness of \( i^{th} \) cuckoo nest is better than the fitness of randomly chosen nest, then nest \( j \) is replaced by \( y_i^{iter+1} \) nest or the \( y_j^{iter+1} \) nest is replaced by nest \( j \) (Naik & Panda, 2016).

**Step 4:** Abandoning fraction of some nests

The fraction of nests, determined by \( p_a \) are abandoned and new ones are built in their places. This process involves the selection of two random solutions for the \( i^{th} \) cuckoo. The two random solutions contribute to the exploitation of the search space. The abandon probability \( p_a \) is used to switch between the exploration and exploitation phase. The generation of new solutions by the local search random walk can be obtained by the following equation (Yang, 2014):

\[
y_i^{iter+1} = y_i^{iter} + \alpha s \odot H(p_a - \varepsilon) \odot (y_j^{iter} - y_k^{iter})
\]

Here, \( y_i^{iter+1} \) is the new solution of nests at iteration \( iter + 1 \), \( y_i^{iter} \) is the previous solution of nests at iteration \( iter \), \( \odot \) represents the entrywise multiplication, \( H(u) \) is a heavy side function, \( \alpha \) is the step size which is usually taken as 1, \( \varepsilon \) is a random number drawn from a uniform distribution and \( s \) is the step size. The \( y_j^{iter} \) and \( y_k^{iter} \) are the two randomly selected solutions at iteration \( iter \).

**Step 5:** Obtaining the best nests

The best nests are updated in each iteration. Finally, the best nests among \( N \) host nests are found.

**Step 6:** Stopping criteria

The generation of new nests continues until the maximum iteration \( m_{iteration} \) has reached.

4. MODIFIED CUCKOO SEARCH ALGORITHM FOR CLUSTERING

In this section, a step by step discussion of the proposed method has been demonstrated.

4.1. Proposed Method Description

MoCS algorithm is used to balance exploration and exploitation properties of the CS algorithm. If we analyze equation (11), this may lead to the following outcome:

- In this equation, \( y_{local} - y_i^{iter} \) contributes towards the exploitation of the search space and \( levy(s,\beta) \) helps in exploration with a generation of step size.
- Exploration in CS is mainly dependent on Levy flight that generates a larger step size for better and larger exploration. However, taking a large fixed step size may skip the optimum result while exploring the search space.

Here, instead of considering a fixed \( \beta \) value, an adaptive \( \beta \) has been considered throughout the process:

\[
\beta = (1 - ((m_{iteration} - t) / m_{iteration})) + 1
\]
Here, \( m_{\text{iteration}} \) represents maximum iteration, and \( t \) represents the current iteration number. From the above equation, it can be observed that the value of \( \beta \) increases as the number of iteration increases. \( \beta \) is used for step calculation. More is the \( \beta \) value more be the step size and greater be the exploration. This proposal intends to search nearby positions with smaller stepsize and gradually the distant positions to be explored with a larger step size. By this process, we not only make it sure that any of the optimum value must not be skipped due to fixed large step size but also the increase in step size in the later period of the search helps not to get trapped in a local optimum by exploring distant position that may have better fitness value. Hence, the increase in stepsize in later stage avoids premature convergence by overcoming the negative impact of continuous exploitation based on \( y_{\text{best}} \). So by this approach, the exploration ability of the model gets increased as this focuses on both nearer as well as distant positions in order to get global optimum. By this process, an effort has been made to balance both exploration and exploitation of the search space to get the optimum result by avoiding the local optima problem.

4.2. Population Representation and Initialization for Clustering

In the MoCS algorithm, the nests represent the solution to the problem. For the clustering problem, the centroids of the clusters need to be determined. So, each solution of the population of cuckoo nests represent the coordinates of the centroids. The population \( P_{\text{iter}} \) with \( N \) solutions and \( D \) dimensions is represented as \( P_{\text{iter}} = \{Y_{1,\text{iter}}, \ldots, Y_{N,\text{iter}}\} \). Each solution \( Y_{i,\text{iter}} \) is represented as \( Y_{i,\text{iter}} = \{y_{1,\text{iter}}, \ldots, y_{d,\text{iter}}\}, \) where \( i = 1, \ldots, N \).

This population is randomly initialized in a search space within the lower and upper boundaries \( Y_{\text{lower}} = \{y_{1,\text{lower}}, \ldots, y_{d,\text{lower}}\} \) and \( Y_{\text{upper}} = \{y_{1,\text{upper}}, \ldots, y_{d,\text{upper}}\} \) respectively.

The population consists of \( N \) solutions. Each of these solutions is assigned the values of the coordinates of all the centroids of \( k \) groups. If each group has \( d \) features, then the dimension \( D = k \times d \). The initial population at iteration \( \text{iter} = 0 \) can be denoted as follows:

\[
P_{\text{iter}} = \begin{bmatrix}
y_{1,\text{iter}}, y_{2,\text{iter}}, \ldots, y_{d,\text{iter}} \\
y_{1,\text{iter}}, y_{d+1,\text{iter}}, \ldots, y_{2d,\text{iter}} \\
\vdots \\
y_{1,\text{iter}}, y_{d+1,\text{iter}}, \ldots, y_{Nd,\text{iter}} \\
y_{N,\text{iter}}, y_{1,\text{iter}}, \ldots, y_{d,\text{iter}} \\
y_{N,\text{iter}}, y_{d+1,\text{iter}}, \ldots, y_{2d,\text{iter}} \\
\vdots \\
y_{N,\text{iter}}, y_{d+1,\text{iter}}, \ldots, y_{Nd,\text{iter}}
\end{bmatrix}
\]  

(17)

To find the limit of every feature, Xiang et al. (2015) considered the range of each feature of the real dataset, which is the same for every cluster. This study considers the upper limit and lower limit of each individual feature of the dataset in each cluster which can be represented as follows:

\[
Y_{\text{upper}} = \left\{y_{1,\text{upper}}, y_{2,\text{upper}}, \ldots, y_{d,\text{upper}}\right\}
\]  

(18)

\[
Y_{\text{lower}} = \left\{y_{1,\text{lower}}, y_{2,\text{lower}}, \ldots, y_{d,\text{lower}}\right\}
\]  

(19)

4.3. Fitness Evaluation

In this paper, the sum of intra-cluster distance is used as a fitness function. The sum of intra-cluster distance is defined as the summation of Euclidean distance between each object and its corresponding centroid. The minimum is the sum of intra-cluster distance, the maximum is the clustering accuracy. This can be represented as:
Fitness = \sum_{i=1}^{k} \sum_{Y_j \in C_i} \|Y_j - Z_i\| \quad (20)

Here, \(Y_i\) is the object, \(k\) is the number of groups or clusters, \(Z_i\) is the centroid of the \(i^{th}\) group and \(\|Y_j - Z_i\|\) is the Euclidean distance between each object to its centroid. This fitness function needs to be minimized for obtaining better clustering results.

4.4. Termination Criteria

The process of generating new nests by levy flight approach, abandoning of some fraction of the population of the nest and creating new nests in their place continues till the maximum iteration is reached. The resulting optimized nest obtained is the solution to the problem of clustering.

4.5. MoCS Algorithm

MoCS algorithm uses equation (16) for balancing the exploration and exploitation of the search space. In this algorithm, population size \(N\) is taken as 50, abandon probability \(P_a\) is taken as 0.25 and maximum iteration \(m_{\text{iteration}} = 600\).

4.5.1. Pseudocode for MoCS Clustering

1. Begin
2. Let \(k\) clusters obtained from the dataset be represented as:

\[
C_1 = \begin{bmatrix} y_{1,1}^1 & \cdots & y_{1,d}^1 \\ \vdots & \ddots & \vdots \\ y_{n_1}^{1,1} & \cdots & y_{n_1,d}^1 \end{bmatrix} \quad C_2 = \begin{bmatrix} y_{1,1}^2 & \cdots & y_{1,d}^2 \\ \vdots & \ddots & \vdots \\ y_{n_2}^{2,1} & \cdots & y_{n_2,d}^2 \end{bmatrix} \quad \cdots \quad C_k = \begin{bmatrix} y_{1,1}^k & \cdots & y_{1,d}^k \\ \vdots & \ddots & \vdots \\ y_{n_k}^{k,1} & \cdots & y_{n_k,d}^k \end{bmatrix}
\]

Here, \(n_1, n_2, \text{ and } n_k\) represents the number of instances in \(C_1, C_2, \text{ and } C_k\) respectively.

3. The maximum and minimum range of each feature of the dataset is found for every cluster:

\[
\text{max}_i \text{ range} = \text{max}(C_i) \quad (22)
\]
\[
\text{min}_i \text{ range} = \text{min}(C_i) \quad (23)
\]

Here \(i\) represents the \(i^{th}\) number cluster. The obtained ranges are in the form of:

\[
\begin{align*}
&\left\{ y_{\text{upper}}^{1,1}, \ldots, y_{\text{upper}}^{1,d} \right\}, \left\{ y_{\text{upper}}^{2,1}, \ldots, y_{\text{upper}}^{2,d} \right\} \ldots \left\{ y_{\text{upper}}^{k,1}, \ldots, y_{\text{upper}}^{k,d} \right\} \\
&\left\{ y_{\text{lower}}^{1,1}, \ldots, y_{\text{lower}}^{1,d} \right\}, \left\{ y_{\text{lower}}^{2,1}, \ldots, y_{\text{lower}}^{2,d} \right\} \ldots \left\{ y_{\text{lower}}^{k,1}, \ldots, y_{\text{lower}}^{k,d} \right\}
\end{align*}
\]
4. The maximum and minimum limits of each cluster are concatenated to obtain the limits of the entire search space:

\[
Y_{upper} = \left\{ y_{upper}^{1,1}, \ldots, y_{upper}^{1,d}, y_{upper}^{2,d+1}, \ldots, y_{upper}^{2,d}, \ldots, y_{upper}^{k,(k-1)d+1}, \ldots, y_{upper}^{k,kd} \right\}
\]

\[
Y_{lower} = \left\{ y_{lower}^{1,1}, \ldots, y_{lower}^{1,d}, y_{lower}^{2,d+1}, \ldots, y_{lower}^{2,d}, \ldots, y_{lower}^{k,(k-1)d+1}, \ldots, y_{lower}^{k,kd} \right\}
\]

5. The initial population of centroids is generated by randomization of search space uniformly within the upper limits and lower limits of the search space.

6. The fitness of the population of centroids is calculated using the fitness function given in equation (20). These fitness values are compared and the minimum fitness value is found. The global best value for the current iteration is updated with this minimum fitness value.

7. While (maximum iterations has not reached):
   
   7.1 The value of the levy flight component $\beta$ is determined by equation (16).
   
   7.2 The new generation of the population of centroids is obtained by levy flight approach using equation (11).
   
   7.3 The fitness of this new generation of centroids is calculated by using equation (20).
   
   7.4 If the fitness of the current generation of centroids is better than the fitness of the previous generation of centroids, then fitness is updated by the fitness of the current generation of centroids.
   
   7.5 The fraction $p_a$ of the population of centroids is abandoned and the new population of centroids is generated using equation (15).
   
   7.6 The fitness of this population is calculated using equation (20). The fitness matrix is updated. The minimum fitness value is assigned to the global best.
   
   7.7 The current iteration $t$ is updated by 1.

8. End

9. The optimized centroids generated are chosen for clustering.

10. The clusters are found using equation (8).

11. The generated cluster number are matched with the actual cluster number of the data object of the actual dataset to find the accuracy of the cluster using equation (28).

12. End

5. EXPERIMENTAL RESULTS AND DISCUSSION

This experiment is carried out in two parts. First, the efficiency of MoCS algorithm for numerical optimization is proved using fifteen benchmark test functions. Secondly, the suitability of MoCS algorithm for clustering is proved by comparing this algorithm with five nature-inspired algorithms such as PSO, ACO, ABC, PSOAG, and CS algorithm. This experiment was carried out in MATLAB 2018b in Windows 10 operating system on Intel Core 1.60GHz processor with 8GB RAM.

5.1. Experiment on Benchmark Test Functions

For testing the efficiency of MoCS algorithm, we consider fifteen benchmark test functions. The MoCS algorithm is compared with the CS algorithm. Out of 15 test functions, the first 10 functions are unimodal functions which are easier to optimize and have global optimum value as 0 and the next
5 test functions are multimodal functions that are difficult to optimize. The test functions are executed for a maximum of 10,000 iterations. Each algorithm is executed for a maximum of 30 times independently. In both the algorithms, the population size is set to 50. Table 1 presents a summary of ten benchmark test functions with 15 dimensions. Table 2 represents a summary of five test functions with a fixed dimension. In both of the cases \( p_a \) and \( \alpha \) are set to 0.25 and 1 respectively (Naik & Panda, 2016). However, \( \beta \) is fixed and set to 1.5 for CS algorithm, but this value changes dynamically for the MoCS algorithm.

### 5.2 Performance Evaluation of Test Functions

This section presents a detailed comparison between CS and the proposed MoCS algorithm based on fifteen benchmark functions. Table 3 represents the obtained results of these fifteen test functions for CS and MoCS algorithm executed for 10,000 iterations. Each algorithm is then executed for 30

| Function name | Mathematical Description | Search Range | Optimal value |
|---------------|--------------------------|--------------|---------------|
| Sphere function | \( f(Y) = \sum_{i=1}^{d} y_i^2 \) | \([-100, 100]^d\) | \( f(0) = 0 \) |
| Schwefel’s 2.21 function | \( f(Y) = \max \{ |y_i|, 1 \leq i \leq d \} \) | \([-100, 100]^d\) | \( f(0) = 0 \) |
| Noise function | \( f(Y) = \sum_{i=1}^{d} iy_i^4 + \text{random}(0,1) \) | \([-1.28, 1.28]^d\) | \( f(0) = 0 \) |
| Step function | \( f(Y) = \sum_{i=1}^{d} (|y_i + 0.5|)^2 \) | \([-100, 100]^d\) | \( f(0) = 0 \) |
| Schwefel’s 1.2 function | \( f(Y) = \sum_{i=1}^{d} (\sum_{j=1}^{i} y_j)^2 \) | \([-100, 100]^d\) | \( f(0) = 0 \) |
| Schwefel’s 2.22 function | \( f(Y) = \sum_{i=1}^{d} |y_i| + \prod_{i=1}^{d} |y_i| \) | \([-10, 10]^d\) | \( f(0) = 0 \) |
| Rosenbrock function | \( f(Y) = \sum_{i=1}^{d-1} \left[ 100(y_{i+1}^2 - y_i^2)^2 + (y_i - 1)^2 \right] \) | \([-30, 30]^d\) | \( f(0) = 0 \) |
| Rastrigin function | \( f(Y) = \sum_{i=1}^{d} [y_i^2 - 10 \cos(2\pi y_i) + 10] \) | \([-5.12, 5.12]^d\) | \( f(0) = 0 \) |
| Griewank function | \( f(Y) = 1 / 4000 \sum_{i=1}^{d} y_i^2 - \prod_{i=1}^{d} \cos(y_i / \sqrt{i}) + 1 \) | \([-600, 600]^d\) | \( f(0) = 0 \) |
| Ackley function | \( f(Y) = -20 \exp(-0.2\sqrt{1/n} \sum_{i=1}^{d} y_i^2) - \exp(1/n \sum_{i=1}^{d} \cos(2\pi y_i)) + 20 + e \) | \([-32, 32]^d\) | \( f(0) = 0 \) |
times independently to obtain the best, average and standard deviation values for these functions. These values are used as measures of evaluation. Figure 1-15 presents the convergence performance graph of these test functions for both CS and MoCS algorithm.

In addition to mean and standard deviation, using a t-test we try to prove that MoCS performs better than traditional CS. Here, the test problem may be stated as the mean value produced by MoCS for different datasets are smaller than the traditional CS algorithm. Considering this problem, 

\[ H_a : \mu < \mu_{H_0} \] 

and 

\[ H_0 : \mu \geq \mu_{H_0} \]

is our left tailed alternative hypothesis and null hypothesis respectively. Here, \( \mu \) (population mean) represents the observed mean by MoCS and \( \mu_{H_0} \) (Hypothesized mean) is the mean resulted from the traditional CS algorithm. Hence, we are supposed to prove the alternative hypothesis and null hypothesis need to be disproved. In addition to this, the confidence level (C) and level of significance (\( \alpha \)) have been considered as 95% and 5% respectively. To find the acceptance of the null hypothesis, the t-test has been performed on 30 independent results produced by both MoCS and CS algorithm for each dataset. For result comparison, we have considered two outputs from this test such as \( h \) and \( p-value \). \( h \) is either 0 or 1 represents the acceptance and rejection of the null hypothesis respectively. Here, \( p-value \) is a scalar value ranges within [0,1]. Smaller be the \( p-value \), more be the chance of null hypothesis rejection. Hence, this may lead to the conclusion that the lesser value of \( p \) signifies the superiority of the proposed MoCS over the CS algorithm. The detailed result of the t-test may be referred from Table 4.

From Table 3, it is observed that MoCS algorithm outperforms the CS algorithm for \( f_1, f_2, f_3, f_5, f_6, f_7, f_8, f_{10}, f_{11} \) and \( f_{12} \) function. For \( f_4 \) and \( f_{14} \) function, both CS and MoCS algorithms obtain the same results. It can also be observed that although both algorithms obtain the same result in case of the best and average values for \( f_{13} \) and \( f_{15} \) function, but CS algorithm outperforms the MoCS algorithm in case of standard deviation values for both of these functions. Analyzing the obtained convergence performance for these functions in figure 1-15, it can be observed that the MoCS algorithm converges faster than the CS algorithm for each of these functions.

### Table 2. Summary of benchmark test functions with fixed dimension

| Function                          | Mathematical Representation                                                                 | Search Range                     | Optimal Value                  |
|----------------------------------|---------------------------------------------------------------------------------------------|----------------------------------|--------------------------------|
| Shekel’s Foxholes Function       | \( f_{11}(Y) = (1 / 500) + \sum_{j=1}^{25} (1 / (j + \sum_{i=1}^{2} (y_j - a_{ij})^8)) \)  | \((-65.5, 65.5)^2\)            | \( f_{11}(-32, 32) \approx 1 \) |
| Kowalik function                 | \( f_{12}(Y) = \sum_{i=1}^{11} [a_i - (y_i (b_i + y_j) / (b_i + y_i + y_j))] \)          | \((-5, 5)^4\)                   | \( f_{12}(0.1928, 0.1908, 0.1231, 0.1358) \approx 0.0003075 \) |
| Six-Hump Camel-Back function     | \( f_{13}(Y) = 4y_1^2 - 2.1y_1^4 + (1 / 3)y_1^6 + y_2y_3 - 4y_2^4 + 4y_3^4 \)          | \((-5, 5)^2\)                   | \( f_{13}(0.08983, -0.7126) = -1.0316285 \) |
| Hartman function                 | \( f_{14}(Y) = \sum_{i=1}^{4} c_i \exp(\sum_{j=1}^{3} a_{ij} (y_j - p_{ij})^2) \)         | \((0, 1)^{4}\)                  | \( f_{14}(0.114, 0.556, 0.852) = -3.86 \) |
| Shekel function                  | \( f_{15}(Y) = \sum_{i=1}^{5} [(Y - a_i)(Y - a_i)^r + c_i]^{-1} \)                          | \((0, 10)^{4}\)                 | \( f_{15}(\rightarrow Y) = -10.1532 \) |
| Function | Indexes | CS | MoCS |
|----------|---------|----|------|
| $f_1$    | Best    | 5.1187e-018(2) | 9.2590e-021(1) |
|          | Avg     | 4.2186e-016(2) | 1.7334e-019(1) |
|          | Std     | 8.7488e-016(2) | 2.3028e-019(1) |
| $f_2$    | Best    | 0.0424(2) | 0.0165(1) |
|          | Avg     | 0.0740(2) | 0.0300(1) |
|          | Std     | 0.0218(2) | 0.0098(1) |
| $f_3$    | Best    | 0.0300(2) | 0.0172(1) |
|          | Avg     | 0.0479(2) | 0.0252(1) |
|          | Std     | 0.0067(2) | 0.0045(1) |
| $f_4$    | Best    | 0(1) | 0(1) |
|          | Avg     | 0(1) | 0(1) |
|          | Std     | 0(1) | 0(1) |
| $f_5$    | Best    | 4.1648e-006(2) | 1.2789e-005(1) |
|          | Avg     | 1.3117e-004(2) | 1.1659e-004(1) |
|          | Std     | 4.0383e-004(2) | 2.5095e-005(1) |
| $f_6$    | Best    | 2.4158e-011(2) | 1.7588e-013(1) |
|          | Avg     | 3.3690e-010(2) | 7.5927e-013(1) |
|          | Std     | 2.2265e-010(2) | 7.0232e-013(1) |
| $f_7$    | Best    | 0.0322(2) | 0.1134(1) |
|          | Avg     | 3.1929(2) | 2.3420(1) |
|          | Std     | 2.4680(2) | 2.2158(1) |
| $f_8$    | Best    | 6.4662(2) | 4.9688(1) |
|          | Avg     | 9.1540(2) | 6.3514(1) |
|          | Std     | 1.3740(2) | 0.7702(1) |
| $f_9$    | Best    | 1.4717e-012(2) | 3.6637e-015(1) |
|          | Avg     | 0.0085(2) | 0.0034(1) |
|          | Std     | 0.0077(2) | 0.0046(1) |
| $f_{10}$ | Best    | 3.7251(2) | 3.3252e-004(1) |
|          | Avg     | 9.6679(2) | 0.9576(1) |
|          | Std     | 4.3522(2) | 0.6823(1) |
| $f_{11}$ | Best    | 0.99980(1) | 0.99980(1) |
|          | Avg     | 0.99980(1) | 0.99980(1) |
|          | Std     | 1.4474e-004(2) | 1.8895e-016(1) |
| $f_{12}$ | Best    | 3.3628e-004(1) | 3.7732e-004(2) |
|          | Avg     | 6.5815e-004(2) | 6.2440e-004(1) |
|          | Std     | 1.3228e-004(2) | 1.2698e-004(1) |
| $f_{13}$ | Best    | -1.0316(1) | -1.0316(1) |
|          | Avg     | -1.0316(1) | -1.0316(1) |
|          | Std     | 6.6486e-016(1) | 6.7752e-016(2) |
| $f_{14}$ | Best    | -3.8628(1) | -3.8628(1) |
|          | Avg     | -3.8628(1) | -3.8628(1) |
|          | Std     | 2.7101e-015(1) | 2.7101e-015(1) |
| $f_{15}$ | Best    | -10.1532(1) | -10.1532(1) |
|          | Avg     | -10.1532(1) | -10.1532(1) |
|          | Std     | 6.0194e-015(1) | 6.9584e-015(2) |

Avg (Rank) 5.0000 3.2000
To analyze the results in a better way, we have adopted a rank-based system. The ranks are enclosed in brackets beside the obtained values. The minimum value is ranked as 1 and the maximum value is ranked as 2 for each case. The sum of these ranks is taken and divided by the number of functions to obtain the final ranking result. The algorithm which obtains the lowest average rank is considered as the best algorithm for numerical optimization. Considering the final ranking results, it is observed that MoCS achieves an average rank of 3.2000, whereas the CS algorithm achieves a rank of 5.0000. After considering the entire set of benchmark functions, it can easily be realized that MoCS outperforms CS in most of the cases. Also, analyzing the convergence performance of both the algorithms from Figure 1-15, the faster and better convergence of MoCS can be acknowledged. So, MoCS can be considered as an efficient algorithm for numerical optimization as compared to the CS algorithm.

In addition to above comparison, in order to prove our claim, we need to disprove the null hypothesis and accept the alternative hypothesis. The detailed result may be analysed by referring Table 4. If we consider the \( h \) value, MoCS completely outperformed the CS algorithm in 10 out of 13 datasets. However, in the case of Schwefel’s 1.2, Shekel’s Foxholes, and Kowalik functions the difference in mean and standard deviation is not that significant to reject the null hypothesis. We got NaN for two of the functions as the result of mean and standard deviation is equal for both the cases. Considering the \( p \) -value, this may be claimed that MoCS completely dominates CS with much more than 95% confidence level. Moreover, for \( f_1, f_2, f_3, f_6, f_8, f_9, f_{10} \) and \( f_{15} \) this may easily be analyzed that MoCS outperformed CS with more than 99% confidence level.

5.3. Experiment on MoCS for Clustering Datasets

This section presents the comparison of the MoCS algorithm for clustering with five other swarm intelligence algorithms such as ABC, ACO, PSO, PSOAG and CS using real datasets. The MoCS algorithm is compared in terms of the sum of intra-cluster distance values and the accuracy of clustering.

5.3.1. Datasets

In this paper, six real datasets from the UCI Machine learning repository have been used for evaluation purposes. They are Iris, Wine, Contraceptive Method Choice (CMC), Cancer, Thyroid and Vowel (Bache & Lichman, 2013). Table 5 lists out these datasets with their number of instances, number of attributes, number of groups and number of instances belonging to each individual group. The brief description of these datasets is presented as follows:

- **Iris Flower dataset:** This dataset consists of 150 instances and 4 attributes (sepal_length, sepal_width, petal_length, petal_width). It contains three species of flowers each consisting of 50 instances (Iris setosa (50 instances), Iris virginica (50 instances), Iris versicolor (50 instances)). \([n=150, d=4, k=3]\)
- **Wine Quality dataset:** This dataset consists of 178 instances and 13 attributes (alcohol, malic acid, ash content, alkalinity of ash, concentration of magnesium, total phenols, flavanoids, non-flavonoid phenols, proanthocyanins, intensity of color, hue, OD280/OD315 diluted wines, and pralines). These wines are categorized based on three cultivators- 59 instances of wine from first cultivator, 71 instances of wine from second cultivator, and 48 instances of wine from third cultivator. \([n=178, d=13, k=3]\)
- **Contraceptive Method Choice (CMC) dataset:** This dataset consists of 1473 instances and 9 attributes (wife’s age, wife’s education, husband’s education, number of children born, religion of wife, working condition of wife, husband’s occupation, standard of living index, and media exposure). This is categorized into three groups (no use-629 instances, long term use-334 instances, short term use-510 instances). \([n=1473, d=9, k=3]\)
Wisconsin Breast Cancer (WBC) dataset: This dataset has 683 instances and 9 attributes (clump thickness, uniformity of cell size, uniformity of cell shape, marginal adhesion, single epithelial cell size, bare nuclei, bland chromatin, normal nucleoli, and mitoses). These instances are divided into two groups—whether the cancer is in the benign stage (444 instances) or malignant stage (239 instances). \([n=683, d=9, k=2]\)

Thyroid dataset: This dataset has 215 instances and 5 attributes (T3resin, Thyroxin, Triiodothyronine, Thyroid stimulating, and TSH Value). These instances are divided into three categories—normal (150 instances), hyperthyroidism (35 instances) and hypothyroidism (30 instances). \([n=215, d=5, k=3]\)

| Function | \(h\) | \(p - value\) |
|----------|-------|----------------|
| \(f_1\)  | 1     | 1.1935e-05     |
| \(f_2\)  | 1     | 5.1557e-08     |
| \(f_3\)  | 1     | 8.1092e-12     |
| \(f_4\)  | NaN   | NaN            |
| \(f_5\)  | 0     | 0.9772         |
| \(f_6\)  | 1     | 2.6483e-08     |
| \(f_7\)  | 1     | 0.0138         |
| \(f_8\)  | 1     | 2.3649e-08     |
| \(f_9\)  | 1     | 0.0030         |
| \(f_{10}\)| 1     | 6.9892e-16     |
| \(f_{11}\)| 0     | 0.1623         |
| \(f_{12}\)| 0     | 0.9004         |
| \(f_{13}\)| 1     | 0.0415         |
| \(f_{14}\)| NaN   | NaN            |
| \(f_{15}\)| 1     | 0.0050         |

Table 4. Performance results of test functions based on t-test
Figure 1. Convergence graph of function $f_1$.

Figure 2. Convergence graph of function $f_2$. 

Figure 3. Convergence graph of Schwefel 2.21 function.
Figure 3. Convergence graph of function $f_3$

![Convergence graph of Noise function](image)

Figure 4. Convergence graph of function $f_4$

![Convergence graph of Step function](image)
Figure 5. Convergence graph of function \( f_5 \).

![Convergence graph of Schwefel 1.2 function](image1)

Figure 6. Convergence graph of function \( f_6 \).

![Convergence graph of Schwefel 2.22 function](image2)
Figure 7. Convergence graph of function \( f_7 \).

![Convergence graph of Rosenbrock function](image)

Figure 8. Convergence graph of function \( f_8 \).

![Convergence graph of Rastrigin function](image)
Figure 9. Convergence graph of function $f_9$

![Convergence graph of Griewank function](image1)

Figure 10. Convergence graph of function $f_{10}$

![Convergence graph of Ackley function](image2)
Figure 11. Convergence graph of function $f_{11}$

![Convergence graph of Shekel foxholes function](image1)

Figure 12. Convergence graph of function $f_{12}$

![Convergence graph of Kowalik function](image2)
Figure 13. Convergence graph of function $f_{13}$

Figure 14. Convergence graph of function $f_{14}$
Indian Telegu Vowel dataset: This dataset has 871 instances and 3 attributes. These instances are splitted into 6 groups (d-72 instances, a-89 instances, i-172 instances, u-151 instances, e-207 instances, o-180 instances). [n=871, d=3, k=6]

5.3.2. Evaluation Measures

In order to analyze the results of the proposed nature-inspired algorithm, five other nature-inspired swarm intelligence methods are taken into consideration- ACO, ABC, PSO, PSOAG, and CS. These are evaluated on the basis of two criteria:

- **Sum of intra-cluster distance**: The distance between an object and its corresponding centroid is called intra-cluster distance. This distance needs to be minimized and is represented by equation 20.
• **Accuracy of cluster**: Accuracy of cluster represents the number of data objects that have been placed in their respective clusters accurately. This is represented by the below equation:

\[
\text{Accuracy}_{\text{cluster}} = \left( \frac{\sum_{i=1}^{n} \text{if}(S_i = T_i) \text{ then } 1 \text{ or } 0}{n} \right) \times 100
\]  

(28)

Here, \(S_i\) and \(T_i\) represent the dataset and \(i^{th}\) point represent the member before and after clustering respectively.

5.3.3. **Performance Evaluation**

In this section, a detailed comparison between the proposed method and other considered swarm optimization algorithm is carried out. For a fair comparison, a similar environment needs to be taken care of. Hence, the swarm intelligence algorithms those have been considered for comparison are executed for a maximum of 600 iterations. Each of these algorithms has been executed 30 times independently to obtain the mean and standard deviation of the clustering results. The comparisons have been considered based on two major factors, i.e. minimum intra-cluster distance and average accuracy. Moreover, best-obtained centroids for each dataset have been provided in this work to validate the result.

Table 6 presents a comparison study of the proposed algorithm with different swarm intelligence algorithms based on the sum of intra-cluster distance. This is to be noted that the results of ACO, ABC, PSO, and PSOAG algorithm are referred from the paper (Nayak et al., 2017). From Table 6, it is observed that MoCS proves itself superior compared to other algorithms in the case of Iris, Cancer and Thyroid dataset. The PSOAG algorithm performs better than other algorithms for the Wine and Vowel dataset. CS algorithm performs better in case of average value, whereas MoCS performs better in case of standard deviation value for the CMC dataset.

A rank-based system is also adapted here to analyze the results and for a fair comparison. Here, each method is assigned with a rank based on the result for a particular dataset. In Table 6, the minimum sum

| Datasets   | Indexes | ACO         | ABC         | PSO         | PSOAG       | CS          | MoCS        |
|------------|---------|-------------|-------------|-------------|-------------|-------------|-------------|
| Iris       | Mean    | 100.67(4)   | 101.00(5)   | 104.45(6)   | 96.97(3)    | 96.8309(2)  | 96.7786(1)  |
|            | Std     | 1.58(5)     | 1.43(4)     | 4.77(6)     | 0.35(3)     | 0.1042(2)   | 0.0549(1)   |
| Wine       | Mean    | 16300.71(2) | 16506.75(6) | 16303.16(3) | 106296.30(1)| 16307.00(4)| 16308.00(5)|
|            | Std     | 10.86(5)    | 131.42(6)   | 4.82(4)     | 1.69(1)     | 2.4602(2)   | 2.6260(3)   |
| CMC        | Mean    | 6151.35(6)  | 5649.94(4)  | 5750.07(5)  | 5559.98(3)  | 5556.1(1)   | 5556.7(2)   |
|            | Std     | 63.22(6)    | 54.01(4)    | 59.56(5)    | 31.97(3)    | 7.9416(2)   | 5.6035(1)   |
| Cancer     | Mean    | 3376.20(5)  | 3102.63(4)  | 4024.79(6)  | 2984.24(3)  | 2969.40(2)  | 2968.60(1)  |
|            | Std     | 42.60(4)    | 68.00(5)    | 270.67(6)   | 17.63(3)    | 2.2387(2)   | 1.6238(1)   |
| Thyroid    | Mean    | 1950.37(4)  | 2111.39(5)  | 2369.90(6)  | 1902.77(3)  | 1870.30(2)  | 1870.10(1)  |
|            | Std     | 15.16(4)    | 94.36(6)    | 85.11(5)    | 16.69(3)    | 2.3107(2)   | 1.6533(1)   |
| Vowel      | Mean    | 170849.30(6)| 160347.29(3)| 154017.60(2)| 149734.40(1)| 161113.00(5)| 160490.00(4)|
|            | Std     | 2055.93(2)  | 3275.35(3)  | 3722.14(6)  | 4.667       | 3537.80(4)  | 3703.50(5)  |
| Avg (Rank) |         | 8.834       | 9.167       | 10.000      | 4.667       | 5.000       | 4.334       |
of intra-cluster distance is ranked the lowest and the maximum sum of intra-cluster distance is ranked the highest. For example, for Iris dataset MoCS produces the best result and PSO produces the worst. Hence, MoCS has been assigned with rank one and PSO has been awarded with rank six. The same idea has been followed in the case of both average intra-cluster distance and the standard deviation. Then these ranks are averaged upon all the datasets and the resultant rankings have been considered based on the average result. Hence, lesser is the value, better is the ranking. According to the average ranking process, MoCS algorithm obtains the lowest average rank of 4.334, followed by PSOAG with an average rank of 4.667, CS with a value of 5.000, ACO with an average rank of 8.834, ABC with a rank of 9.167 and PSO obtains the highest average rank of 10. From these results, the potential of the proposed MoCS has been proven in terms of intra-cluster distance. In addition to that, MoCS can be considered as more robust as compared to others in terms of minimum result fluctuation.

Table 7 presents a comparison study of the proposed algorithm with different swarm intelligence algorithms based on the accuracy of the clustering. From Table 7, it is observed that MoCS proves itself superior compared to other swarm intelligence algorithms in the case of two out of six datasets. The PSOAG algorithm performs better than other algorithms for Iris and Thyroid dataset. In the case of Cancer dataset, both CS and MoCS outperform other algorithms by obtaining better mean and standard deviation results. CS algorithm performs better than other compared algorithms for Vowel dataset. It can also be observed that although MoCS algorithm lags behind PSOAG algorithm for Wine dataset in terms of intra-cluster distance in Table 6, it performs well in terms of accuracy of cluster. Considering the standard deviation results, MoCS algorithm obtains a better standard deviation, close to 0 for almost every dataset.

This is obvious that the method which obtains better accuracy that may lead to a better clustering results. Considering the same rank-based system, the maximum accuracy value is ranked the lowest and minimum accuracy value is ranked the highest. According to the average ranking, MoCS algorithm obtains the lowest rank of 3.5, followed by CS with a rank of 3.834, PSOAG with a rank of 5.834, PSO with a rank of 7.5, ABC with a rank of 7.667 and ACO obtains the highest rank of 10.167. From the above analysis, it can be concluded that MoCS algorithm obtains the lowest rank in terms of the sum of intra-cluster distance and clustering accuracy, so it can be considered as an efficient and suitable method for clustering.

Table 8-13 presents the best-obtained cluster centroids for all the six datasets. These centroids may be used to validate the results of sum of intra-cluster distance and clustering accuracy. Figure 16-21 presents the convergence performance graph of these six datasets in 600 iterations. Considering the above tables and figures, the efficacy of the proposed algorithm can be analyzed.

| Datasets | Indexes | ACO | ABC | PSO | PSOAG | CS | MoCS |
|----------|---------|-----|-----|-----|-------|----|------|
| Iris     | Mean    | 72.17(6) | 90.63(2) | 89.73(4) | 91.03(1) | 86(5) | 90(3) |
|          | Std     | 3.55(5) | 1.63(3) | 2.28(4) | 1.27(2) | 0(1) | 0(1) |
| Wine     | Mean    | 61.18(6) | 70.90(4) | 71.21(2) | 70.98(3) | 70.7865(5) | 72.4719(1) |
|          | Std     | 3.13(6) | 0.92(5) | 0.36(4) | 0.33(3) | 1.4454e-014(1) | 2.8908e-014(2) |
| CMC      | Mean    | 36.96(6) | 40.10(3) | 39.80(5) | 39.87(4) | 45.8927(2) | 46.2332(1) |
|          | Std     | 0.77(6) | 0.70(5) | 0.53(4) | 0.30(3) | 3.6134e-014(2) | 0(1) |
| Cancer   | Mean    | 78.23(5) | 95.51(3) | 94.36(4) | 96.31(2) | 96.4861(1) | 96.4861(1) |
|          | Std     | 1.13(4) | 0.55(3) | 1.61(5) | 0.20(2) | 4.3361e-014(1) | 4.3361e-014(1) |
| Thyroid  | Mean    | 51.93(6) | 60.67(5) | 62.93(4) | 74.37(1) | 73.9535(2) | 72.0930(3) |
|          | Std     | 2.37(2) | 10.55(4) | 3.63(3) | 10.95(5) | 0(1) | 0(1) |
| Vowel    | Mean    | 36.50(6) | 53.59(3) | 54.05(2) | 51.75(4) | 59.8163(1) | 45.9242(5) |
|          | Std     | 1.78(3) | 4.75(6) | 4.01(4) | 4.25(5) | 3.6134e-014(2) | 7.2269e-015(1) |
| Avg (Rank) |        | 10.167 | 7.667 | 7.5000 | 5.834 | 3.834 | 3.500 |
Table 8. Best obtained centroids for Iris dataset

| Dataset | Centroid1 | Centroid2 | Centroid3 | Features |
|---------|-----------|-----------|-----------|----------|
| Iris    | 7.8690    | 4.3000    | 5.9383    | $f_1$    |
|         | 2.5583    | 3.1708    | 3.3731    | $f_2$    |
|         | 6.9000    | 1.4833    | 2.4364    | $f_3$    |
|         | 2.5000    | 0.1000    | 1.4117    | $f_4$    |

Table 9. Best obtained centroids for Wine dataset

| Dataset | Centroid1 | Centroid2 | Centroid3 | Features |
|---------|-----------|-----------|-----------|----------|
| Wine    | 13.1000   | 12.4000   | 12.4000   | $f_1$    |
|         | 0.7000    | 1.7000    | 1.8000    | $f_2$    |
|         | 1.4000    | 2.8000    | 1.4000    | $f_3$    |
|         | 17.4000   | 20.1000   | 19.8000   | $f_4$    |
|         | 102.1000  | 99.2000   | 93.5000   | $f_5$    |
|         | 3.3000    | 2.5000    | 1.1000    | $f_6$    |
|         | 2.8000    | 0.9000    | 1.8000    | $f_7$    |
|         | 0.7000    | 0.7000    | 0.1000    | $f_8$    |
|         | 0.4000    | 1.1000    | 1.2000    | $f_9$    |
|         | 6.5000    | 5.4000    | 4.4000    | $f_{10}$ |
|         | 1.7000    | 0.5000    | 1.4000    | $f_{11}$ |
|         | 1.5000    | 2.0000    | 2.1000    | $f_{12}$ |
|         | 1113.8000 | 691.1000  | 500.6000  | $f_{13}$ |
Table 10. Best obtained centroids for CMC dataset

| Dataset | Centroid1 | Centroid2 | Centroid3 | Features |
|---------|-----------|-----------|-----------|----------|
| CMC     | 39.2417   | 33.3886   | 28.7153   | \( f_1 \) |
|         | 2.8834    | 3.1402    | 2.9786    | \( f_2 \) |
|         | 3.5364    | 3.5678    | 3.5120    | \( f_3 \) |
|         | 4.0777    | 3.9383    | 2.3740    | \( f_4 \) |
|         | 0.8754    | 0.9866    | 1.0000    | \( f_5 \) |
|         | 0.7785    | 0.7362    | 0.7002    | \( f_6 \) |
|         | 1.9948    | 2.0424    | 2.2894    | \( f_7 \) |
|         | 3.4354    | 3.3054    | 3.0666    | \( f_8 \) |
|         | 0.0387    | 0.0005    | 0.0122    | \( f_9 \) |

Table 11. Best obtained centroids for Cancer dataset

| Dataset | Centroid1 | Centroid2 | Features |
|---------|-----------|-----------|----------|
| Cancer  | 2.1576    | 7.8795    | \( f_1 \) |
|         | 1.0000    | 7.6621    | \( f_2 \) |
|         | 1.0000    | 7.6202    | \( f_3 \) |
|         | 1.0000    | 6.5648    | \( f_4 \) |
|         | 1.9638    | 5.7986    | \( f_5 \) |
|         | 1.0000    | 9.2539    | \( f_6 \) |
|         | 1.9770    | 6.6824    | \( f_7 \) |
|         | 1.0000    | 6.8940    | \( f_8 \) |
|         | 1.0000    | 2.2683    | \( f_9 \) |
Table 12. Best obtained centroids for Thyroid dataset

| Dataset | Centroid1  | Centroid2  | Centroid3  | Features |
|---------|------------|------------|------------|----------|
| Thyroid | 118.6454   | 85.6684    | 104.3695   | $f_1$    |
|         | 9.6002     | 11.4933    | 12.9289    | $f_2$    |
|         | 2.4574     | 3.5886     | 3.1825     | $f_3$    |
|         | 1.4336     | 0.7257     | 0.1000     | $f_4$    |
|         | 3.9843     | 6.9059     | -0.7000    | $f_5$    |

Table 13. Best obtained centroids for Vowel dataset

| Dataset | Centroid1  | Centroid2  | Centroid3  | Centroid4 | Centroid5 | Centroid6 | Features |
|---------|------------|------------|------------|-----------|-----------|-----------|----------|
| Vowel   | 0.5274     | 0.4545     | 0.4628     | 0.2500    | 0.6687    | 0.7586    | $f_1$    |
|         | 0.5450     | 0.1900     | 1.1182     | 2.5500    | 1.3932    | 1.7347    | $f_2$    |
|         | 2.4582     | 2.6464     | 2.3346     | 3.2000    | 2.3039    | 0.9984    | $f_3$    |

Figure 16. Performance graph of Iris dataset
Figure 17. Performance graph of Wine dataset

Figure 18. Performance graph of CMC dataset
Figure 19. Performance graph of Cancer dataset

![Convergence graph for Cancer dataset](image)

Figure 20. Performance graph of Thyroid dataset

![Convergence graph for Thyroid dataset](image)
6. CONCLUSION

An algorithm for data clustering called Modified Cuckoo Search (MoCS) is proposed in this paper. Here, an attempt has been made to balance exploration and exploitation of the search space in the CS algorithm. To test the efficacy of the proposed model, the performance of MoCS algorithm is tested using 10 benchmark test functions with 15 dimension and 5 test functions with fixed dimension. The comparison results revealed that the MoCS algorithm is an efficient algorithm for numerical optimization as compared to CS algorithm. Further, the comparison of MoCS algorithm with five other nature-inspired algorithms on the basis of sum of intra-cluster distance and clustering accuracy is presented. All these comparison results conclude that MoCS algorithm can be considered as a suitable algorithm for data clustering. However, this may be tested on high dimensional data and may be implemented for different applications associated with those to evaluate its efficiency. Most importantly, we have focused on intra-cluster distance for clustering, but neglected inter-cluster distance which is one of the important aspects of clustering. Hence, this may be taken as a future challenge to focus on building a more effective fitness function that includes both aspects.
REFERENCES

Ahalya, G., & Pandey, H. M. (2015). Data clustering approaches survey and analysis. In International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE) (pp. 532-537). doi:10.1109/ABLAZE.2015.7154919

Armano, G., & Farmani, M. R. (2016). Multiobjective clustering analysis using particle swarm optimization. Expert Systems with Applications, 55, 184–193. doi:10.1016/j.eswa.2016.02.009

Bache, K., & Lichman, M. (2013). UCI Machine Learning Repository. University of California, School of Information and Computer Science, Irvine, CA, USA. Retrieved July 8, 2019, from https://archive.ics.uci.edu/ml

Cheng, J., Wang, L., Jiang, Q., Cao, Z., & Xiong, Y. (2018). Cuckoo search algorithm with dynamic feedback information. Future Generation Computer Systems, North-Holland, 89, 317–334. doi:10.1016/j.future.2018.06.056

Cura, T. (2012). A particle swarm optimization approach to clustering. Expert Systems with Applications, 39(1), 1582–1588. doi:10.1016/j.eswa.2011.07.123

Dorigo, M. (1992). Optimization, Learning, and Natural Algorithms (PhD thesis). Politecnico di Milano, Italy.

Jiang, B., Wang, N., & Wang, L. (2013). Particle swarm optimization with age-group topology for multimodal functions and data clustering. Communications in Nonlinear Science and Numerical Simulation, 18(11), 3134–3145. doi:10.1016/j.cnsns.2013.03.011

Karaboga, D. (2005). An idea based on honey bee swarm for numerical optimization. Technical Report-tr06. Erciyes University, 200, 1–20.

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. In Proceedings of ICNN’95 - International Conference on Neural Networks (pp. 1942–1948). doi:10.1109/ICNN.1995.488968

Krishnanand, K. N., & Ghose, D. (2005). Detection of multiple source locations using a glowworm metaphor with applications to collective robotics. In Proceedings 2005 IEEE Swarm Intelligence Symposium, 2005.SIS 2005. IEEE. doi:10.1109/SIS.2005.1501606

Leung, Y., Zhang, J.-S., & Xu, Z.-B. (2000). Clustering by Scale-Space Filtering. IEEE Transactions on Pattern Analysis and Machine Intelligence, 22(12), 1396–1410. doi:10.1109/34.895974

Lloyd, S. (1982). Least Square Quantization with PCM. IEEE Transactions on Information Theory, 28(2), 129–13. doi:10.1109/TIT.1982.1056489

Long, W., Liang, X., Huang, Y., & Chen, Y. (2014). An effective hybrid cuckoo search algorithm for constrained global optimization. Neural Computing and Applications, 25(3-4), 911–926. doi:10.1007/s00521-014-1577-1

Naik, M. K., & Panda, R. (2016). A novel adaptive cuckoo search algorithm for intrinsic discriminant analysis based face recognition. Applied Soft Computing, 38, 661–675. doi:10.1016/j.asoc.2015.10.039

Nanda, S. J., & Panda, G. (2014). A survey on nature inspired metaheuristic algorithms for partitional clustering. Swarm and Evolutionary Computation, Elsevier, 16, 1–18. doi:10.1016/j.swevo.2013.11.003

Nayak, S. K., Rout, P. K., & Jagadev, A. K. (2017). A cross mutation-based differential evolution for data clustering. International Journal of Data Mining, Modelling and Management, 9(1), 17–38. doi:10.1504/IJDMMM.2017.082571

Nesmachnow, S. (2014). An overview of metaheuristics: Accurate and efficient methods for optimisation. International Journal of Metaheuristics, 3(4), 320. doi:10.1504/IJMHEUR.2014.068914

Saida, I. B., Nadjet, K., & Omar, B. (2014). A New Algorithm for Data Clustering Based on Cuckoo Search Optimization. Springer. doi:10.1007/978-3-319-01796-9_6

Shelokar, P., Jayaraman, V., & Kulkarni, B. (2004). An ant colony approach for clustering. Analytica Chimica Acta, Elsevier, 509(2), 187–195. doi:10.1016/j.aca.2003.12.032

Xiang, W. L., Zhu, N., Ma, S. F., Meng, X. L., & An, M. Q. (2015). A dynamic shuffled differential evolution algorithm for data clustering. Neurocomputing, Elsevier, 158, 144–154. doi:10.1016/j.neucom.2015.01.058
Xu, D., & Tian, Y. (2015). A Comprehensive Survey of Clustering Algorithms. *Annals of Data Science. Springer Berlin Heidelberg*, 2(2), 165–193. doi:10.1007/s40745-015-0040-1

Yang, X. S. (2008). *Nature Inspired Metaheuristic Algorithms*. Luniver Press.

Yang, X.-S. (2014). *Nature-inspired optimization algorithms*. Elsevier.

Yang, X.-S., & Deb, S. (2009). Cuckoo Search via Lévy flights. In 2009 *World Congress on Nature & Biologically Inspired Computing (NaBIC)*, (pp. 210–214). doi:10.1109/NABIC.2009.5393690

Zhang, C., Ouyang, D., & Ning, J. (2010). An artificial bee colony approach for clustering. *Expert Systems with Applications, Pergamon, 37*(7), 4761–4767. doi:10.1016/j.eswa.2009.11.003

Preeti Pragyan Mohanty has completed Masters in Computer Science and Engineering from Siksha ‘O’ Anusandhan (Deemed to be University) in 2019. She completed her BTech in Computer Science and Engineering from Institute of Technical Education and Research, Siksha ‘O’ Anusandhan (Deemed to be University), Bhubaneswar in 2018. Her area of interests include data mining and evolutionary computing.

Subrat Kumar Nayak is currently working as an Associate Professor in the Department of Computer Science and Engineering, Institute of Technical Education and Research, Siksha ‘O’ Anusandhan (Deemed to be University), Odisha, India. He received his BTech in Computer Science and Engineering from the Biju Pattanaik University of Technology, Odisha, India, in 2008. He received his M.Tech degree in 2012 and awarded a Ph.D. degree in the year 2017 from Siksha ‘O’ Anusandhan (Deemed to be University). His field of interest includes evolutionary computation, multi-objective optimization, and application of soft computing techniques in data mining.