Towards Enhancement of Performance of K-Means Clustering Using Nature-Inspired Optimization Algorithms

1. Introduction

Based on a partitioning strategy, K-means clustering algorithm [1] assigns membership to data points by measuring the distance between each pair of data point and centroid of a designated cluster. The membership assignment will be progressively refined until the best possible assignment is yielded—that is when the total intradistances of the data points within a cluster are minimized and the total interdistances of the data points across different clusters are maximized. The final quality of the clustering results, however, depends largely on the values of the initial centroids at the beginning of the partitioning process. These initial centroid values are randomly generated each time the clustering kick-starts which are different from time to time. By such random chance, K-means can probably plunge into local optima whereby the final quality of the clusters falls short from the globally best. An example that is depicted in Figure 1 demonstrates some possible outcomes of K-means. The top two snapshots represent good clustering results where the cluster distributions are even; the bottom two snapshots show otherwise, the clustering results in uneven distribution. All these depend on the starting positions of the centroids which are randomly generated.

Technically it is possible though not feasible in achieving a globally optimum clustering result, via a brute-force approach in trying out exhaustively all partitioning possibilities. As the number of clusters and the number of data points increase, the combinatorial number of possible grouping arrangements escalates, leading to computationally prohibitive. Therefore, heuristic approach is desired for seeking global optima stochastically, improving the quality of the final clustering results iteration by iteration. Metaheuristics which enable incremental optimization by design are ideal candidates for such computation.
A substantial collection of nature-inspired optimization methods aka metaheuristics have emerged recently with designs mimicking swarm behavior exhibited by living creatures. Each of the search agents represents a particular combination of centroid positions; they move and search for optimality in their own ways, they sometimes communicate with each other and are collectively guided towards a global optimization goal. To date, the proposed nature-inspired optimization algorithms have gained much attention among data mining researchers. The computational merits have been verified mathematically and their feasibilities have been applied in various practical applications. However, validating the efficacy of hybrids combining such nature-inspired algorithms with classical data mining algorithms is still at an infant stage [2].

By the merits of design, nature-inspired optimization algorithms are believed to be able to overcome the shortcomings of K-means clustering algorithms on the issue of getting stuck in local optima. The objective of this study is to validate the efficacy of the hybrids and to quantitatively measure the quality of clustering results produced by each of the hybrids. In our experiments, we used four popular nature-inspired optimization methods to combine with K-means clustering algorithm. The integration is paramount, because it enables enhancement over data mining algorithms which have many wide applications.

A preliminary experiment reported in [3] shows their feasibility as a successful pioneer exemplar. This paper reports further experiment including a case study of image segmentation using these hybrid algorithms. In the past, some researchers started to integrate nature-inspired optimization methods into K-means algorithms [4]; their research efforts are limited to almost the same form of swarming maneuvers—that is, there is constantly a leader in the swarm which the fellow agents follow. Some examples are nature-inspired optimization algorithms such as the Artificial Bee Colony [5], Firefly [6], and Particle Swarm Optimization [7]. For the sake of intellectual curiosity, in our experiments as well as whose models described in this paper, two nature-inspired algorithms which take on a slightly different course of swarming are included. They are the Bat Algorithm [8] which swarm with varying speeds and the Cuckoo Algorithm [9] which do not swarm but iterate with fitness selection improvement. They represent another two main groups of algorithms and their variants, which are adaptive to the environment by their sensing abilities, and utilize a special method to improve the solution by evolving the old solutions from one generation into better ones in new generations. Specifically, the performance indicators in terms of speed and time consumption for clustering by these two bioinspired algorithms integrated with K-means are observed. The technical details of the aforementioned nature-inspired algorithms and the K-means clustering are not duplicated here. Readers are referred to the respective references for the background information of the algorithms involved in this paper.

2. Enhancing K-Means Clustering by Nature-Inspired Optimization Algorithms

A major reason in obtaining quality K-means clustering results is having the right combination of centroid positions.
The resultant centroids ideally should be dispersed in such a way that the clusters formed upon them yield the maximum quality, which we call a global optimum. It is characterized by having the properties of maximum intrasimilarities and minimum intersimilarities in clustering. K-means is known to produce clustered data with nonoverlapping convex clusters, and they always converge quickly. One drawback is that clusters in K-means do not always converge into the global optimum. Just like any other partition-based clustering algorithm, initial partition is made by some randomly generated centroids which do not guarantee the subsequent convergence will lead to the best possible clustering result. K-means, for example, is reputed to have the final clusters stuck at local optima which prevent further exploration for the best results [10]. In reality, to achieve the best clustering results, it would require running many rounds with each round taken different random initiations of centroid values. Hence, in this paper, we have four hybrids, which resulted from combining four bioinspired optimization algorithms into K-means. With the capital letter C denoting “clustering,” the four hybrid algorithms are called C-ACO, C-Bat, C-Cuckoo, and C-Firefly, respectively. The original mathematical models for the four bioinspired optimization algorithms can be found in [6, 8, 9, II], respectively.

Table 1 consists of the parameters for the C-Firefly algorithm. $X$ is a composite matrix of size $[N,(K \otimes D)]$, where $x \in X$ since $X$ has a maximum of $K$ centroids and each centroid is represented by a maximum of $D$ dimensions by the attributes.

The ranges of the parameters are as follows: $i = 1, \ldots, N$, $j = 1, \ldots, S$, and $k = 1, \ldots, K$. The double line notation in (4) means it is a function of Euclidean distance. The interpretation of (4) is that the $i$th search agent that is now handling the $k$th cluster takes a value by measuring the minimized distance between the $i$th search agent and centroid of the $k$th center. The equation is an objective function in which the smaller the value the better. As long as the value of cmat is minimized by this metaheuristic optimization approach, every data point within a cluster would be drawn as close as possible to the centroid. The metaheuristic will guide the search agents to find the appropriate centroids for the clusters.

Nature-inspired optimization algorithms require certain functional parameters to be initiated with values to run. The function parameters are defined as follows. They allow the users to set with user-specific values for customizing the operations of the algorithms. Some of the parameters are common across different bioinspired optimization algorithms. In this paper, we have four hybrids, which resulted from combining four bioinspired optimization algorithms into K-means. With the capital letter C denoting “clustering,” the four hybrid algorithms are called C-ACO, C-Bat, C-Cuckoo, and C-Firefly, respectively. The original mathematical models for the four bioinspired optimization algorithms can be found in [6, 8, 9, II], respectively.

Table 1 consists of the parameters for the C-Firefly algorithm. $X$ is a composite matrix of size $[N,(K \otimes D)]$, where $x \in X$ since $X$ has a maximum of $K$ centroids and each centroid is represented by a maximum of $D$ dimensions by the attributes.

The ranges of the parameters are as follows: $i = 1, \ldots, N$, $j = 1, \ldots, S$, and $k = 1, \ldots, K$. The double line notation in (4) means it is a function of Euclidean distance. The interpretation of (4) is that the $i$th search agent that is now handling the $k$th cluster takes a value by measuring the minimized distance between the $i$th search agent and centroid of the $k$th center. The equation is an objective function in which the smaller the value the better. As long as the value of cmat is minimized by this metaheuristic optimization approach, every data point within a cluster would be drawn as close as possible to the centroid. The metaheuristic will guide the search agents to find the appropriate centroids for the clusters.
and C-Bat have recently been reported in [1]. Readers may refer to [1] for the detailed integration of a K-Means clustering algorithm with the firefly and bat algorithms.

2.1. Cuckoo Clustering Algorithm (C-Cuckoo). In the original cuckoo algorithm, Yang and Deb used an analogy whereby each egg in a nest represents a solution, and a cuckoo egg represents a new solution. The goal is to use the new and better solution to replace a relatively poor solution by chasing off an old egg with a new cuckoo egg. We adopt the same analogy in constructing our C-Cuckoo algorithm. The solution $x$ represents the host nest. In the clustering algorithm, the solution is composed of a set of real numbers representing the cluster center. As defined earlier, $x$ takes the form of a $(N,K \otimes D)$ matrix where $N$ is the population, $K$ is the number of clusters, and $D$ is the number of attributes associated with each data point. The second index of the matrix represents the center of all $K$ clusters, and the whole $x$ represents the current locations of all the cuckoos. We now give a simple example. If there is a given data set and two dimensions and we need to create two clusters ($K = 2, D = 2$), then the value of $x$ is four. The $ith$ $x$ in the middle of the clustering process may look something like this:

Cluster 1: 1 3
Cluster 2: 8 9

In the initialization phase, the population of $n$ host nests $x_i$, where $i = 1, 2, ..., n$, is generated. The cluster centers are represented by the means of the attributes. Each cuckoo has the same parameters (Table 2): Tol (tolerance) and $pa$ (alien egg discovery rate). In this phase, the most important action is that a cluster ID is randomly assigned to each cuckoo as the initial clustering result.

Because the cuckoo has characteristics typical of Levy flight, when it comes to generating a solution $x^{(i+1)}$ for cuckoo $i$, we use the equation $x^{(i+1)} = x^{(i)} + \delta \odot \text{Levy}(\lambda)$, where $\delta$ is the step size scalar used to control the resolution of the step length. In our algorithm, we use $\delta = 1$, which satisfies most cases. The above formula means the cuckoo takes a random flight, which is based on Mantegna’s algorithm [12]. The algorithm takes the following steps. First, the number of centroids $k$ is initialized, as are the other variables. By going through a random walk, the nest, which is regarded as the central point of a cluster, is updated. The step length $s$ is calculated by $s = u/|v|^{1/\beta}$ and $\beta \in [1, 2]$, where $u$ and $v$ are drawn from normal distributions. That is, $u \sim N(0, \sigma_u^2)$ and $v \sim N(0, \sigma_v^2)$, where $\sigma_u = \left[ \Gamma(1 + \beta) \sin(\pi \beta/2)/\Gamma((1 + \beta)/2) \sigma_x^{(\beta−1)/2} \right]^{1/\beta}$. This distribution follows the Levy distribution.

The goal of this clustering algorithm is to search for the best center to minimize the distance between the center of the cluster and its points. Our objective function is, thus, the same as (3) and its result is the degree of fitness. After calculating the degree of fitness, we use (4) to assign each point to a suitable cluster. A better degree of fitness represents a good quality cuckoo solution.

The best solution derived from the above equations is then nominated, and the new solution replaces the old. If no new solution is better than the old one, the current solution is retained as the best. The host bird cleans out the nest according to a given probability. The next iteration of the computation process then occurs to look for the next best solution. In each iteration, the best and hence the optimal solution to date is set as the clustering centroid.
The centroid is represented as a paired tuple, $\text{cen}(i,:)$, where $i$ is the central point of the cluster. The tuple has the format of $(k,d)$, where $k$ is the $k$th cluster and $d$ is the coordinates in $(x,y,z,...)$ or higher dimensions. For example, the locations of three clusters can be represented as $[1,(3,4,5),2,(8,9,3),3,(5,6,4)]$. Each cuckoo represents a cluster with coordinates $(k,d)$, and $d$ is the central coordinate of the cuckoo. Each cuckoo is initially assigned a random $d$ value. Subsequently, $d$ is updated by iterative optimization. The progressive search for the best solution helps to avoid local optima in a manner similar to chromosome mutations in genetic algorithms. When the locations of the cuckoos are set in each round, the distances between the data points and the centers are measured by their Euclidean distance. The data points are reassigned to their nearest cluster by

$$\text{MIN} \{x - \text{cen}\}.$$  

The clusters are then reformed on the basis of the newly assigned data points. At the beginning, the averages of the data points are used as starting centroids by

$$\text{cen} = \frac{1}{N} \sum_{x_i \in \text{cluster}_i} x_i, \quad i = 1,\ldots,K.$$  

In this way, the algorithm achieves better partitioning of clusters at the start to avoid the center points being too near or too far from each other, as would occur if they were assigned purely by random chance. As the algorithm runs, the clustering distribution is refined and changes to quality centroids are avoided by averaging the data. This is why (6) is only needed at the beginning to initialize the starting centroids. According to survival of the fittest, the partitioning process reaches a final optimum. The logic of the C-Cuckoo algorithm is shown as a flow chart in Figure 2.

2.2. Bat Clustering Algorithm (C-Bat). Each bat has its own unique ID, position, and fitness attributes. However, in the bat algorithm, each bat is assigned the same loudness, pulse frequency, and wavelength. The position of a bat is represented by a solution $x$. Its location is determined by the values of $D$ dimensions. As for the C-Cuckoo algorithm, the solution uses a $(N,K \otimes D)$ matrix, the second term identifying the location of the bat.

The initiation step is similar to that employed for the C-Cuckoo algorithm. However, the bats have an additional feature: each bat has a velocity $v_i$, which is similar to particle swarm optimization. The bat's position is partly determined by its velocity. At first, the bats are randomly distributed. After initialization, the bats move to a better place according to (8). A random number is then produced: if it is larger than the current bat rate, the algorithm selects a solution from those calculated and generates a local solution. The centroids are the averages of the nearby data points. The distances are then minimized according to the direction of the optimization goal. The objective functions are identical to (5) and (6) above. The convergence process then starts to iterate based on the following formula:

$$\beta = Q \times \text{rand}() ,$$  

where $\text{rand}()$ is a random value generator and the random values are distributed over $[0,1]$. Consider

$$f_i = f_{\text{min}} + (f_{\text{max}} - f_{\text{min}}) \times \beta ,$$

$$v_i = v_i^{t-1} + (x_i^{t-1} - x^*) \times f_i ,$$

$$x_i^{t} = x_i^{t-1} + v_i^{t}.$$  

The positions of the bats are then updated. $f$ represents the frequency of echolocation. When the frequency equals the sum of the minimum frequency and the difference between the maximum and minimum frequencies, the speed of the bat is updated. The new speed is set to the previous
Initialize the bat population \( x \) and \( v \), define frequency and loudness

Set the initial centroids to be mean of each attribute

Assign clusters for each bat \( x \) randomly

Calculate the fitness and find the best solution

Last iteration? Yes

Generate new solution by adjusting frequency and updating velocity, solution and centroid

No

\( \text{Rand}(\cdot) > r_i \)

Yes

Choose a nest \( j \) randomly

No

\( \text{Rand}(\cdot) > A_i \text{ and } f(x_i) < f(x_j) \)

Yes

Accept the new solutions increase \( r_i \) and reduce \( A_i \)

End

**Figure 3: Workflow of C-Bat algorithm.**

speed plus the product of the previous frequency and the difference between the current position and the previous position. A variable called the pulse rate is also used in the algorithm. When the pulse rate is exceeded, the following formula is updated:

\[
    x_{i+1} = x_i + 0.01 \times \text{rand}(\cdot).
\]  

Equation (9) serves as the updating function, where \( x_b \) is taken as the best solution. It is also used to represent the best position for the bat to move towards. If the loudness value is not high enough and the new solution is better than the old one, the better one becomes the solution. A fitness function the same as that employed for the C-Cuckoo algorithm is then applied by checking whether echolocation is loud enough. The logic of the C-Bat algorithm is shown as a flow chart in Figure 3.

**Table 4: Testing dataset information.**

| Dataset        | Instances | Attributes | Clusters |
|---------------|-----------|------------|----------|
| Iris          | 150       | 4          | 3        |
| Wine          | 178       | 13         | 3        |
| Haberman's survival | 306   | 3          | 2        |
| Libras        | 360       | 91         | 15       |
| Synthetic     | 600       | 60         | 6        |

**3. Experiments**

There are two sets of experiments; one is focused in evaluating the performance of algorithms using a series of multivariate real-life datasets. The other is to test out the efficacy of the algorithms in image segmentation. The purpose is to validate the new algorithms with respect to their clustering quality. The first test is about how the new algorithms work with general-purpose datasets with different number of attributes and instances. Their performances in particular are evaluated in details. The latter test is to observe how well these algorithms will work in the domain of machine vision. The setups of the experiments and their results are discussed as follow.

**3.1. Experiment Set-Up.** The new nature-inspired clustering algorithms (C-ACO, C-Bat, C-Cuckoo, and C-Firefly) proposed here are experimented over six datasets which are available for download from UCI data repository (http://archive.ics.uci.edu/ml). The clustering results of the new hybrid clustering algorithms are compared with those of original K-means which serves as a benchmarking reference. The computer simulation environment is implemented in MATLAB software and the algorithms are coded in MATLAB programs. The hardware platform is a MacBook Pro computer configured with a 2.3 GHz CPU and 4 GB RAM. In each trail run, each of the testing datasets is run repeated ten times for measuring up the average CPU time consumption, as well as obtaining the average values of the top objective fitness. The datasets used are collected from real-life applications, namely, Iris, Wine, Haberman's survival, Libras, and Synthetic. Synthetic is a dataset of control chart time-series which are artificially generated clustering data point values in random. The information regarding the testing datasets is shown in Table 4.
CPU time consumption per iteration during the clustering operation, and the number of loops taken for all the clusters to get converged. The criterion for convergence which decides when the looping of evolution stops is the fraction of the minimum distance between the initial cluster centers that takes on a numeric value between [0, 1]. For example, if the criterion is initialized with 0.01, the looping halts when a complete iteration does not move any of the cluster centers by a distance of more than 1% of the smallest distance between any of the initial cluster centers.

The quality of the final outcome of clustering is measured by the integrity of each of the clusters, which in turn is represented by the final fitness value of the objective function. The resultant fitness value of the objective function is driven by how much each variable contributes towards the final goal which is being optimized in the process. From the perspective of clustering the goal is finding a suitable set of centroids as guided by the metaheuristic of the nature-inspired algorithm. The metaheuristic will always insist that the relocation of centroids in each step is progressive aiming at exploring for the optimum grouping. The end result of the ideal group should lead to having the data points within each cluster closest to their centroid. Iteratively, the search for the optimum grouping proceeds. During the search the k centroids relocate in the search space step-by-step according to the swarming pattern of the nature-inspired optimization algorithm until no more improvement is observed. It stops when there is no further relocation that will offer a better result. To be precise, no other new relocation of centroids seems to provide better integrity of the clusters. The algorithm is geared at minimizing the intrasimilarity of each cluster by an objective function. In this case, it is a squared error function so any slight difference will be enlarged by the square function. Equation (10) defines such objective function. Consider

\[
V = \sum_{j=1}^{K} \sum_{i=1}^{N} \|x_{ij} - cen_j\|^2.
\] (10)

In the experiments, each dataset is run ten times to test and obtain the average CPU time and is also run ten times to test the objective function values/best fitness value. The parameters are set as reported in Tables 5 and 6.

### 3.2. Testing the New Clustering Algorithms with General-Purpose Multivariate Datasets

The five diagrams in Figure 4 below present snapshots of the experimental run for the Iris dataset. The original data points are shown in the topmost plot Figure 4(a), and the data points in different colors obtained by the new clustering algorithms are shown in Figures 4(b), 4(c), 4(d), and 4(e) by C-Firefly, C-Cuckoo, C-ACO, and C-Bat, respectively.

The quantitative experimental results are shown in Tables 7, 8, 9, 10, and 11. To make observation easier, the best result across the four algorithms under test (in each column) is highlighted with a double asterisks. It is apparent to observe that the C-Cuckoo and C-Bat algorithms achieve a lot better of objective fitness value than do the C-ACO and C-Firefly algorithms. Overall, the four nature-inspired clustering algorithms execute in less time and succeed in achieving higher accuracy in clustering than the plain K-means. High accuracy is referred to high cluster integrity where the data points in a cluster are close to their centroid. This observation tallies with our proposition that K-means enhanced by nature-inspired optimization algorithms speeds up the searching time for the good centroids for good clustering integrity. This enhancement is important because for all the partitioning-based clustering methods potentially they can be enhanced by nature-inspired algorithms in the similar way—the end result is expected in search process acceleration and local optima avoidance.

### Table 5: Parameters set for C-Firefly and C-Cuckoo.

| Parameter | C-Firefly | C-Cuckoo |
|-----------|-----------|-----------|
| \(\alpha\) (randomness): \(0.2\) | \(Pa\) (discovery rate): \(0.25\) | \(Pa\) (discovery rate): \(0.25\) |
| \(\gamma\) (absorption): \(1.0\) | \(Tol\) (tolerance): \(1.0\times10^{-5}\) | \(Tol\) (tolerance): \(1.0\times10^{-5}\) |
| Number of clusters: \(4\) | \(\beta\) (Levy): \(\frac{3}{2}\) | \(\beta\) (Levy): \(\frac{3}{2}\) |
| Max number of iterations: \(20\) | \(\varepsilon\) (Levy): \(\sigma_\epsilon = \left[ \frac{\Gamma(1 + \beta) \sin(\pi \beta/2)}{\Gamma((1 + \beta)/2) \beta^{(\beta-1)/2}} \right]^{1/\beta}\) | \(\sigma_\epsilon = \left[ \frac{\Gamma(1 + \beta) \sin(\pi \beta/2)}{\Gamma((1 + \beta)/2) \beta^{(\beta-1)/2}} \right]^{1/\beta}\) |
| Population: \(400\) | \(\beta = 0\) | \(\beta = 0\) |
| \(\beta_0 = 0\) | Max number of iterations: \(20\) | Max number of iterations: \(20\) |
| Population: \(400\) | | |

### Table 6: Parameters set for C-Bat and C-ACO.

| Parameter | C-Bat | C-ACO |
|-----------|-------|-------|
| \(A\) (loudness): \(0.5\) | Population: \(400\) | Population: \(400\) |
| \(R\) (pulse rate): \(0.5\) | \(q\) (threshold): \(0.9\) | \(q\) (threshold): \(0.9\) |
| Number of clusters: \(4\) | Number of clusters: \(4\) | Number of clusters: \(4\) |
| Max number of iterations: \(20\) | \(\rho\) (evaporation rate): \(0.1\) | \(\rho\) (evaporation rate): \(0.1\) |
| \(Q_{min}\) (frequency min): \(0\) | \(Q_{max}\) (frequency max): \(0.2\) | \(Q_{max}\) (frequency max): \(0.2\) |
| Population: \(400\) | Max number of iterations: \(20\) | Max number of iterations: \(20\) |

### Table 7: Testing objective function fitness and CPU time consumption on IRIS.

| Objective function value | Best | Worst | Average |
|--------------------------|------|-------|---------|
| K-means | 78.9451 | 152.3687 | 127.8941 |
| C-ACO | 139.3081 | 150.9815 | 144.79 |
| C-Firefly | 78.9408** | 109.4036 | 89.88241 |
| C-Cuckoo | 78.9408** | 78.9408** | 78.9408** |
| C-Bat | 78.9408** | 81.2655 | 79.46626 |

| CPU time (second) | Best | Worst | Average |
|-------------------|------|-------|---------|
| K-means | 51.29144 | 51.673431 | 51.444755 |
| C-ACO | 440.7980 | 443.9105 | 443.0668 |
| C-Firefly | 142.89517 | 177.74384 | 167.242438 |
| C-Cuckoo | 15.499098 | 15.798136 | 15.7319299 |
| C-Bat | 8.6212** | 8.8354** | 8.75495** |
Our detailed performance evaluation tests include computing the final objective function fitness values and the CPU time consumption for clustering the data in the general-purpose UCI datasets in Table 4. The objective function fitness value is computed using (10) which represents the overall cluster integrity, and CPU time consumption is timed as how long it is necessary for the clustering algorithm to converge from beginning to end. Given the datasets are of fixed volume, CPU time consumption is related directly to the speed of the clustering operation.

Tables 7 to 11 clearly show that the C-Cuckoo and C-Bat algorithms both yield better objective values than the C-ACO and C-Firefly algorithms. The study reported in [1] has already shown that the C-ACO and C-Firefly algorithms perform more quickly and accurately than a traditional K-means specification. Our evaluation results provide further evidence confirming this phenomenon: nature-inspired algorithms do indeed accelerate the process of finding globally optimum centroids in clustering, and partitioning clustering methods can be combined with nature-inspired algorithms to speed up the clustering process and avoid local optima. Furthermore, our results show that two new hybrid clustering algorithms, the C-Cuckoo and C-Bat specifications, are more efficient and accurate than the others we test.

The next experiment is undertaken to measure the average computation time required per iteration in the clustering process. Only the Iris dataset is used here, as it is one of the datasets in the UCI repository most commonly used for testing time spent per iteration for clustering with nature-inspired algorithms.

From Figures 5 and 6, it can be seen that all four algorithms scale quite well—as the number of iterations increases, the computation time taken remains flat. In particular, C-ACO is very fast. It takes only a fraction of a second to execute each iteration of the clustering process. C-Firefly takes about 8 to 10 seconds. C-Cuckoo and C-Bat are relatively fast, taking less than a second per iteration for code execution. The CPU times taken for each algorithm are reported in Table 12. The
Table 8: Testing objective function fitness and CPU time consumption on wine.

| Objective function value | Best          | Worst          | Average         |
|--------------------------|---------------|----------------|-----------------|
| K-means                  | 2.3707e+006   | 2.3707e+006    | 2.3707e+006     |
| C-ACO                    | 2.3707e+006   | 2.3707e+006    | 2.3707e+006     |
| C-Firefly                | 2.3707e+006   | 2.3707e+006    | 2.3707e+006     |
| C-Cuckoo                 | 2.3707e+006   | 2.3707e+006    | 2.3707e+006     |
| C-Bat                    | 2.3707e+006   | 2.3707e+006    | 2.3707e+006     |

CPU time (second)

| Best          | Worst          | Average         |
|---------------|----------------|-----------------|
| K-means       | 73.793875      | 77.226464       | 75.7386         |
| C-ACO         | 623.3123       | 695.2312        | 653.2154        |
| C-Firefly     | 260.725002     | 278.206271      | 267.823618      |
| C-Cuckoo      | 18.582329      | 19.137426       | 18.751136       |
| C-Bat         | 10.3032**      | 10.8225**       | 10.42148**      |

Table 9: Testing objective function fitness and CPU time consumption on Libras.

| Objective function value | Best          | Worst          | Average         |
|--------------------------|---------------|----------------|-----------------|
| K-means                  | 822.8381      | 899.2441       | 842.3452        |
| C-ACO                    | 1.1361e+003   | 1.6345e+003    | 1.3981e+003     |
| C-Firefly                | 743.3432      | 892.0506       | 777.1993        |
| C-Cuckoo                 | 707.5916**    | 819.1392**     | 763.1102**      |
| C-Bat                    | 745.8008      | 918.2488       | 841.7931        |

CPU time (second)

| Best          | Worst          | Average         |
|---------------|----------------|-----------------|
| K-means       | 1332.059811    | 1392.3113       | 1362.0344       |
| C-ACO         | 1.0142e+004    | 1.3245e+004    | 1.1195e+004     |
| C-Firefly     | 260.725002     | 278.206271      | 267.823618      |
| C-Cuckoo      | 168.502947     | 169.849548      | 169.17942       |
| C-Bat         | 10.3032**      | 10.8225**       | 10.42148**      |

Table 10: Testing objective function fitness and CPU time consumption on Haberman.

| Objective function value | Best          | Worst          | Average         |
|--------------------------|---------------|----------------|-----------------|
| K-means                  | 30507.0207    | 31321.2134     | 30912.2141      |
| C-ACO                    | 2888.4833**   | 3051.1412**    | 2933.1641**     |
| C-Firefly                | 30507.2735    | 31567.2721     | 30822.3426      |
| C-Cuckoo                 | 30507.0207    | 30574.2412     | 30591.1234      |
| C-Bat                    | 30507.8928    | 31455.1241     | 30712.8321      |

CPU time (second)

| Best          | Worst          | Average         |
|---------------|----------------|-----------------|
| K-means       | 170.0535       | 174.123152      | 172.743         |
| C-ACO         | 825.2858       | 856.3724        | 831.223         |
| C-Firefly     | 1257.6772      | 1289.3124       | 1266.4123       |
| C-Cuckoo      | 25.1611        | 26.2312         | 25.4431         |
| C-Bat         | 14.5597**      | 14.9087**       | 14.6722**       |

Table 11: Testing objective function fitness and CPU time consumption on Synthetic.

| Objective function value | Best          | Worst          | Average         |
|--------------------------|---------------|----------------|-----------------|
| K-means                  | 9.8221e+005   | 1.0451e+006    | 9.9632e+005     |
| C-ACO                    | 43847.7264**  | 5.4264.3234**  | 47321.9682**    |
| C-Firefly                | 9.4509e+005   | 9.8221e+005    | 9.6622e+005     |
| C-Cuckoo                 | 9.4499e+005   | 9.7800e+005    | 9.5721e+005     |
| C-Bat                    | 9.5232e+005   | 9.8525e+005    | 9.6150e+005     |

CPU time (second)

| Best          | Worst          | Average         |
|---------------|----------------|-----------------|
| K-means       | 1579.2311      | 1663.2234       | 1617.305490     |
| C-ACO         | 4354.9328      | 4725.4525       | 4556.2073       |
| C-Firefly     | 51092.5733     | 51404.6632      | 51232.5532      |
| C-Cuckoo      | 138.204069     | 151.547294      | 141.900688      |
| C-Bat         | 77.8049**      | 80.8928**       | 79.63274**      |

Figure 5: Computation time (in secs.) for C-Firefly, C-Cuckoo, and C-Bat algorithms.

The following graphs in Figures 7 and 8 show the number of iterations required for the clustering algorithms to converge according to the given threshold criterion. As shown in Figure 7, the C-Firefly, C-Cuckoo, and C-Bat algorithms take about two or three iterations to achieve convergence, which is extremely fast. In contrast, C-ACO in Figure 8 takes many rounds to converge, 4681 iterations to be exact. For the other three algorithms, the best objective function value is reached at 78.94. C-ACO goes no lower than 101 and remains there even if the number of iterations increases to a large value.

Therefore, we may conclude that the C-Firefly, C-Cuckoo, and C-Bat algorithms are suitable for static data. C-Firefly
compares data $O(n^2)$ times in each iteration, so it takes a lot of time to converge. The traditional K-means algorithm converges easily to a local optimum, so the result of the objective function is worse than for the others. C-Bat has the ability to adjust itself in every iteration, and because it only changes location once, at the end of an iteration, it is very fast. Because C-Cuckoo retains better solutions and discards worse solutions, working like a PSO-clustering algorithm, it also performs well in providing objective function values.

Although C-Bat, C-Cuckoo, and C-Firefly may need more time for each iteration, they are good optimization algorithms. They can find the optimal solution relatively quickly overall (because they converge very fast). However, the ants acting as the searching agents in the C-ACO algorithm make only a small move in each iteration. Many comparisons are thus required to find the best solution. In sum, C-ACO may be suitable for applications in which incremental optimization is desired and very little time is needed for each step, but it may take many steps to reach the optimal goal.

The next set of experiments tests the quality of clustering in terms of accuracy (measured as 100% minus the percentage of instances overlapping in wrong clusters) and standard deviation. Standard deviation is related to how much variation from the average (mean) is caused by clustered data. The misclustered data derived in the experiments can be seen in Figure 3. Most of the results are satisfactory. The standard
deviations indicate that the data points tend to be very close to the mean. The mathematical definition is simply
\[
\sigma = \sqrt{\frac{1}{N} \left[ (x_1 - u)^2 + (x_2 - u)^2 + \cdots + (x_N - u)^2 \right]},
\]
where \( u = (1/N)(x_1 + \cdots + x_N) \). Again, the most widely employed dataset, the iris dataset, is used for this set of experiments.

Table 13 shows that the results obtained using the C-Bat algorithm are the best in the iris dataset, whereas those derived with the C-Cuckoo algorithm are the best in the wine dataset. In the Haberman data, all five algorithms are almost equally accurate, though the C-ACO algorithm is slightly more precise.

Table 14 shows that the C-Firefly algorithm has the minimum deviation within clusters, whereas the original K-means algorithm deviates to the greatest extent.

### Table 13: Accuracy of clustering in the three datasets using different clustering algorithms.

| Name      | Iris   | Wine   | Haberman |
|-----------|--------|--------|----------|
| K-means   | 0.8666 | 0.7225 | 0.522    |
| C-Firefly | 0.7866 | 0.7225 | 0.529    |
| C-ACO     | 0.68   | 0.5505 | 0.562**  |
| C-Cuckoo  | 0.89   | 0.7225 | 0.526    |
| C-Bat     | 0.92** | 0.6966 | 0.473    |

### Table 14: Standard deviation measures for different algorithms in different datasets.

| The standard deviation of each algorithm | Iris        | Wine        | Haberman   |
|----------------------------------------|-------------|-------------|------------|
| Algorithm | Cluster1 | Cluster2 | Cluster3 | Algorithm | Cluster1 | Cluster2 | Cluster3 | Algorithm | Cluster1 | Cluster2 | Cluster3 |
|-----------|----------|----------|----------|-----------|----------|----------|----------|-----------|----------|----------|----------|
| K-means   | 1.8476** | 1.7277   | 1.9847   | K-means   | 319.8828 | 194.4313 | 123.3868 |
| C-ACO     | 1.9185   | 1.809    | 1.9742   | C-ACO     | 296.271** | 155.1661** | 154.7441 |
| C-Firefly | 1.8476** | 1.6269** | 1.9136** | C-Firefly | 319.8828 | 194.4315 | 123.3836** |
| C-Cuckoo  | 1.8476   | 1.7345   | 1.982    | C-Cuckoo  | 319.8828 | 194.4315 | 123.3836** |
| C-Bat     | 1.8476** | 1.7311   | 1.9741   | C-Bat     | 318.5658 | 200.5721 | 130.7622 |

### Table 15: Performance of clustering results of image segmentation using different clustering algorithms.

| Algorithms | Intercluster distance | Intracluster distance | CPU time (s) |
|------------|-----------------------|-----------------------|--------------|
| K-means    | 6.5932e+4             | 1.1154e+8             | 24.5465***   |
| C-ACO      | 6.7741e+4             | 1.0561e+8             | 351.9310     |
| C-Bat      | 6.5932e+4             | 1.0561e+8             | 39.3933      |
| C-Cuckoo   | 6.7412e+4             | 1.0561e+8             | 39.7490      |
| C-Firefly  | 3.5565e+4             | 1.0407e+8             | 150.1532     |

3.3. Testing the New Clustering Algorithms in Image Segmentation. In this set of experiment, the new hybrid clustering algorithms are put under test of image segmentation task. Pixel color oriented image segmentation is the core of image analysis which finds its applications in many areas of image interpretation, pattern identification/recognition, and robotic vision. Some popular applications [13] include but are not limited to geographical information remote sensing, medical microscopy, content-based audio/visual media retrieval, factory automation, and unmanned vehicle navigation, just to name a few.

In practical scientific and industrial applications, the quality and accuracy of image segmentation are very important which depend on the underlying data clustering algorithms. A common choice of unsupervised clustering algorithm is K-means in image segmentation based on color. The regions of the image depending on the color features are grouped into a certain set of segments by measuring the intercluster distance and intracluster distance between each image pixel and the centroid within the cluster. The clustering process is exactly the same as that used in the previous experiment on UCI datasets, except that the images under test in this experiment are larger in amount. An 8 MB high-resolution photo like those used in the experiment...
here has typically \( 5184 \times 3456 \) pixels. In addition to spatial information, \( x \)- and \( y \)-axes of the pixel position in the image, each pixel is triplet of red, green, and blue information, ranging from 0 in darkness to 255 being the strongest in intensity. It is well-known that every pixel of an image is made up by mixing the 256 independent intensity levels of red, green, and blue light. Each data point of the 17,915,904 pixels is a five-dimensional matrix comprised of the pixel location information and RGB information, \([x, y, R, G, B]\) where \( 0 \leq x \leq 5184 \), \( 0 \leq y \leq 3456 \), and \( 0 \leq R, G, B \leq 255 \). The hybrid clustering algorithms are extended from K-means in the same way as described in Section 2. The required image data can be technically extracted by using MATLAB functions `imread(filename)` that creates a three-dimensional matrix and `impixel()` that returns the values of the RGB triplet for the pixel.

The experiment is run over four images whose pixels are to be clustered using different preset numbers of \( k = 3 \) and \( k = 4 \). The four images are shots of sceneries, namely, Tower Bridge (TB), Cambridge University (CU), Le Mont-Saint-Michel (MSM), and Château de Chenonceau (CDC). They have similar image composition and identical size. Some particular features that are subtly contained in the images are used for testing the efficacy of the clustering algorithms like those as follows.

(i) TB, thickness of clouds in the sky, the shaded part of the tower bridge.
(ii) CU, depths of perspectives along the cupula and college building, details on the lawn.
(iii) MSM, details of the fortress wall and small windows on the chapel.
(iv) CDC, reflection of the Château over the water.

The performance is again measured by intersimilarity and intrasimilarity across and within the same cluster, as well as time taken in seconds in processing a whole image. The performance results are tabulated in Table 15. The winning performance result by one of the four hybrid clustering algorithms or K-means is marked with a triple asterisks as distinction. The original images under test and the segmented images by the various clustering algorithms are shown in

**Figure 9:** Results of image segmentation by using different nature-inspired clustering algorithms, on a photo called “Tower Bridge.”
Figures 9, 10, 11, and 12, respectively for TB, CU, MSM, and CDC.

The performance is again measured by intersimilarity and intrasimilarity across and within the same cluster, as well as time taken in seconds in processing a whole image. The performance results are tabulated in Table 15. The winning performance result by one of the four hybrid clustering algorithms or K-means is marked with a triple asterisks as distinction.

In all the tests, K-means seems to take the shortest time, probably because it stops early in local optima. This is evident by the fact that none of the results by K-means score the best in either intercluster distance or intracluster distance. In the experiment of TB, C-ACO scores the widest intercluster length, and C-Firefly has the tightest intracluster distance. As a result, visually C-ACO produces slightly more details on the cloud at the top right corner. C-Firefly seems to produce the most details on the sun-facing side of the tower as well as the shaded side of the tower. C-Firefly again scores the best in intracluster distance in CU and CDC. Again, in CU, C-Firefly offers the most details on the lawn; the copula likewise has most details and reproduces seemingly most accurately on the college facade by C-Firefly. Interestingly, C-Cuckoo has the longest intercluster distance in MSM and CDC. In MSM C-Cuckoo gives the most structural outline of shades and colors on the wall of the chapel, while C-Firefly produces most details on the fortress wall. In CDC, C-Cuckoo and C-Firefly manage to produce the relatively best reflection images over the water, by visual inspect.

The overall results of the experiments described in this paper show that two of the new clustering algorithms observed here, the C-Cuckoo and C-Bat algorithms, which have never been tested by other researchers, are more efficient and accurate than the C-ACO and C-Firefly specifications. This represents a significant contribution to existing knowledge, because it sheds light on the encouraging possibility that optimization techniques derived from nature can be used to improve K-means clustering; we hope that this lays the foundation for more sophisticated bio-inspired optimization methods to be integrated with existing clustering algorithms.

The characteristics of each of the four bioinspired clustering algorithms are listed in the Appendix by way of...
Figure 11: Results of image segmentation by using different nature-inspired clustering algorithms, on a photo called "Le Mont-Saint-Michel."

summary. It is hoped that researchers will find them useful as a source of inspiration for developing better algorithms in future. As the phrase "meta-heuristics" suggests, these bioinspired optimization heuristics come in abstract and general forms. There is ample potential to extend, modify, and even build hybrids of them with other heuristic functions to suit different applications.

4. Conclusion

K-Means clustering algorithms, a classical class of partition-based algorithms used for merging similar data into clusters, are known to have the limitation of getting stuck in local optima. As a matter of intellectual curiosity in computer science, how best to cluster data such that the integrity of the clusters is maximized, has always been a challenging research question. The ideal solution is to find an optimal clustering arrangement which is globally best—so that no other possible combinations of data clustering exist that are better than the global one. One way of achieving this is to try all the possible combinations by brute-force which could be computational intractable. Alternatively, nature-inspired optimization algorithms, which recently rise as a popular research topic, are extended to work with K-means in guiding the convergence of disparate data points and to steer them towards global optima, stochastically instead of deterministically. These two research directions of metaheuristic optimization and data mining do fit like hand and glove. Constrained by the inherent limitation of K-means design and the merits of nature-inspired optimization algorithms, it is feasible to combine them letting them complement and function together. This paper evaluates four hybrid types of clustering algorithms developed by integrating nature-inspired optimization algorithms into K-means. The results produced from the experiments clearly validate the new algorithms possess a performance enhancement, apparently for the C-Bat and C-Cuckoo. The extended versions of clustering algorithms enhanced by nature-inspired optimization methods perform better than their original versions, in two sets of experimental datasets—general purpose and image
Figure 12: Results of image segmentation by using different nature-inspired clustering algorithms, on a photo called “Château de Chenonceau.”

segmentation. Experiments are conducted to validate the benefits of the proposed approach.

Appendix

C-Firefly. (1) In this algorithm, using light intensity as the objective function value, each firefly \( x \) is represented by its location.

(2) Every firefly needs \( O(n^2) \) comparisons, and each one will move towards the brighter one. Therefore, it takes more time for each iteration, but convergence can be reached in very few iterations.

(3) The attractiveness varies according to the inverse square law \( (r) = \beta_0 e^{-\gamma r^2} \), the \( \gamma \) is the absorb coefficient.

C-ACO. (1) This algorithm uses a pheromone matrix as the communication channel.

(2) The pheromone matrix leads the ant to choose the path.

(3) Ants construct the solution step-by-step, and the decision variables of associated mathematical objective function are discrete.

C-Cuckoo. (1) The nests represent the population.

(2) The cuckoo moves using Levy flight, which is a Markov chain in which the next location depends solely on the current location.

(3) This algorithm has the ability to discover foreign eggs and abandon them. This means that the algorithm can avoid local optima.

(4) In each iteration, the worst solution will be replaced by the better one. If there is no worse solution, the better one will be retained for the next iteration.

(5) Every cuckoo only cares about its nest. It is not necessary to communicate.

C-Bat. (1) Each bat has data on velocity and location. This is similar to PSO-clustering.
Velocity is determined by frequency, loudness, and pulse rate.

The solution can be adjusted by frequency, loudness, and pulse rate.

In each iteration, the bat only updates its location once. Therefore, the algorithm runs very quickly.

The bat has the ability to adjust itself. It can sense the surrounding environment.

Similarities among these algorithms.

They all use attribute means to determine the initial centroid.

Optimization algorithms are the key factors in accelerating clustering and avoiding local optima.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper. The authors of this paper do not have a direct financial relationship with the commercial identities mentioned in this paper that might lead to a conflict of interests.

Acknowledgments

The authors are thankful for the financial support from the research Grant no. MYRG152(Y3-L2)-FST11-ZY, offered by the University of Macau, RDAO. Special thanks go to Mr. Rui Tang, who was a research assistant and MSc software engineering graduate of University of Macau, for programming the Matlab codes and conducting the data mining experiments.

References

[1] J. B. MacQueen, "Some methods for classification and analysis of multivariate observations," in Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability, pp. 281–297, University of California Press, 1967.

[2] S. Fong, "Opportunities and challenges of integrating bio-inspired optimization and data mining algorithms," in Swarm Intelligence and Bioinspired Computation, pp. 385–401, Elsevier, 2013.

[3] R. Tang, S. Fong, X.-S. Yang, and S. Deb, "Integrating nature-inspired optimization algorithms to K-means clustering," in Proceedings of the 7th International Conference on Digital Information Management (ICDIM '12), pp. 116–123, Macau, China, August 2012.

[4] J. Senthilnath, S. N. Omkar, and V. Mani, "Clustering using firefly algorithm: performance study," Swarm and Evolutionary Computation, vol. 1, no. 3, pp. 164–171, 2011.

[5] D. Karaboga, "An idea based on honey bee swarm for numerical optimization," Tech. Rep. TR06, Computer Engineering Department, Engineering Faculty, Erciyes University, 2005.

[6] X.-S. Yang, "Firefly algorithms for multimodal optimization," in Stochastic Algorithms: Foundations and Applications, vol. 5792 of Lecture Notes in Computer Sciences, pp. 169–178, Saga, 2009.

[7] X.-S. Yang, S. Deb, and S. Fong, “Accelerated particle swarm optimization and support vector machine for business optimization and applications,” in Proceedings of the 3rd International Conference on Networked Digital Technologies (NDT ’11), vol. 136, pp. 53–66, Springer CCIS, Macau, 2011.

[8] X.-S. Yang, "A new metaheuristic bat-inspired algorithm," in Nature Inspired Cooperative Strategies for Optimization (NICSO 2010), J. R. Gonzalez, D. A. Pelta, C. Cruz, G. Terrazas, and N. Krasnogor, Eds., vol. 284 of Studies in Computational Intelligence, pp. 65–74, Springer, Berlin, Germany, 2010.

[9] X.-S. Yang and S. Deb, "Cuckoo search via Lévy flights," in Proceedings of the World Congress on Nature and Biologically Inspired Computing (NABIC ’09), pp. 210–214, IEEE, Coimbatore, India, December 2009.

[10] S. Z. Selim and M. A. Ismail, “K-means-type algorithms: a generalized convergence theorem and characterization of local optimality,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 6, no. 1, pp. 81–87, 1984.

[11] M. Dorigo, V. Maniezzo, and A. Colorni, “Ant system: optimization by a colony of cooperating agents,” IEEE Transactions on Systems, Man, and Cybernetics B: Cybernetics, vol. 26, no. 1, pp. 29–41, 1996.

[12] B. B. Mandelbrot, The Fractal Geometry of Nature, W.H. Freeman, New York, NY, USA, 1982.

[13] Z. Anil and S. K. K. Chitade, “Colour based image segmentation using K-means clustering,” International Journal of Engineering Science and Technology, vol. 2, no. 10, pp. 5319–5325, 2010.