A Modular Spatial Clustering Algorithm with Noise Specification

Akhil K
Department of Computer Science and Engineering
PES University
Bengaluru, India
akhilkred@gmail.com

Srikanth H R
Department of Computer Science and Engineering
PES University
Bengaluru, India
srikanthhr@pes.edu

Abstract*— Clustering techniques have been the key drivers of data mining, machine learning and pattern recognition for decades. One of the most popular clustering algorithms is DBSCAN due to its high accuracy and noise tolerance. Many superior algorithms such as DBSCAN have input parameters that are hard to estimate. Therefore, finding those parameters is a time consuming process. In this paper, we propose a novel clustering algorithm Bacteria-Farm, which balances the performance and ease of finding the optimal parameters for clustering. Bacteria-Farm algorithm is inspired by the growth of bacteria in closed experimental farms - their ability to consume food and grow - which closely represents the ideal cluster growth desired in clustering algorithms. In addition, the algorithm features a modular design to allow the creation of versions of the algorithm for specific tasks / distributions of data. In contrast with other clustering algorithms, our algorithm also has a provision to specify the amount of noise to be excluded during clustering.

Keywords - clustering algorithms; modular clustering; noise tolerance in clustering; spatial clustering

I. Introduction

In recent times, clustering has been the center piece of major fields such as data science, machine learning, knowledge discovery, statistics and data mining. In the information age, due to the presence of a plethora of uncleaned, unlabeled data, extraction of insights from this data is very essential in many applications.

Clustering is the process of breaking down data into meaningful subdivisions called clusters based on the similarity between data points. The points in a cluster have a higher similarity than the ones across clusters.

There is always room for improvement in the clustering paradigm where a newer algorithm is more efficient and effective to a certain distribution of data. One of the most important problems faced while designing a clustering algorithm is choosing the parameters of the algorithm. If the algorithm is susceptible to a tiny change in those parameters, the robustness of the algorithm is affected. Due to this problem, most of the time is spent on selecting the ideal parameters for the given data than in clustering. Algorithms such as DBSCAN [1] use parameters that are hard to estimate in a short period of time.

Partitioning clustering algorithms are the simplest kind of clustering algorithms. The idea is to breakdown the entire data into arbitrary k clusters where the partitions optimize a given function. For every cluster, a represent-er in the form of *centroid*, *medoid*, etc. is used to iteratively optimize the clusters with the addition of new data points into the cluster. The advantage of using these algorithms lie in the efficiency of their linearity. But, due to the reliance on the initial configuration of clusters, these algorithms lack robustness. Also, they are not suitable for non-convex data or data with noise.

Hierarchical clustering algorithms produce a nested structure of clustering data points. They contain two types: *top-down* and *bottom-up*. In *top-down* algorithms, initially, the entire data set is taken as a single cluster and it is sequentially broken down into smaller clusters until they are singleton clusters. On the other hand, *bottom-up clusters* consider every point as a singleton cluster and sequentially combines the data points into bigger clusters than in the previous level. The advantages of using these algorithms lie in the flexibility of choosing the most appropriate number of clusters and their sizes from different levels of clusters. Like the *partitioning cluster algorithms*, they are very sensitive to the presence of noise. Also, they might encounter difficulties in handling convex and large data. Hence, they can prove to be ineffective for real data.

Density based clustering algorithms group objects / data points based on the density of the locality rather than the proximity between data points. The high density regions are considered as clusters and low density ones as noise. With the advent of density based algorithms, clustering performance was boosted due to the provision of dealing with noise and non-convex data. But, these algorithms are very sensitive to the input parameters as small changes in the values of the parameters can completely shift the structure of clusters. Nevertheless, the performance of density based algorithms is generally greater than partitioning algorithms.

Distribution-based clustering algorithms group data based on likelihood of data points belonging to a distribution (or cluster). Objects / data points which most likely belong to the same distribution are clustered together. Though their theoretical foundation is sound, they suffer from *over fitting* as complex models are generated easily. Hence, estimation of the complexity of the model is difficult. Moreover, real data may not belong to a precise distribution model and the presence of such models will lead to poor performance in these algorithms [2]. However, distribution-based algorithms work well on complex, spatial data.

With a plethora of clustering algorithms with their own advantages and disadvantages, a general algorithm is desired. In this paper, we introduce a modular design to our model to

accommodate these various needs of clustering algorithms. In this design, we obtain hyper-parameters for the novel algorithm by pre-clustering a fraction of data with the best standard algorithm for that distribution and fine-tuning these results with our algorithm. Along with this design, the model contains a salient feature to specify the amount of noise to be excluded by the algorithm.

This paper is organized as follows: Related work on clustering, modular algorithms are briefly discussed in Section 2. In Section 3, the design and implementation of the new algorithm are comprehensively explained. In Section 4, the performance evaluation of the algorithm when compared to *k*-means and DBSCAN is presented. Section 5 concludes the paper and some ideas for future research are discussed.

II. Related Work and Definitions

A. Related Work

One of the first kinds to enter the clustering paradigm are partitioning clustering algorithms. Reference [3] proposes a partitioning algorithm having *k* clusters. Each cluster is represented by a *medoid* and sum of distances within clusters serves as the optimization function. Reference [4] seeks a *local optima* instead of a *global optima* to enhance clustering performance. Reference [5] implements an efficient version of the *Lloyd's k-means* algorithm to further improve performance. Reference [6] proposes a *k-d tree* organization of data to efficiently find patterns in the data. Reference [7] proposes a *global k-means* clustering algorithm which incorporates a deterministic global optimization method and employs the *k*-means algorithm as a local search method. The main disadvantage of the work until then was the sensitivity of the algorithm to initial *centroid* positions in the k-means algorithm. By using this method, the issue of randomly selecting initial cluster *centroids* is eliminated and the algorithm proceeds in an incremental way to optimally add a new cluster center to the previous stage. Though this reduces the randomness involved in the k-means algorithm, the sequential addition of a cluster center affects the execution performance. Reference [8] proposes an improved *k*-means algorithm which requires some information on the required domain. With this prerequisite, the algorithm incorporates background information in the form of *instance-level* constraints. Reference [9] proposes a method to reduce the *euclidean* distance calculations in the the original *k*-means algorithm.

Reference [10] proposes a *genetic k-means* algorithm, a hybrid *genetic* algorithm that replaces the *crossover* operation with a *k-means operator* to generate an efficient genetic algorithm for clustering. Reference [11] improvises on the *genetic k-means* algorithm by ensuring the convergence to a global optimum among other improvisations over its parent version.

DBSCAN, a density based algorithm proposed in [1] improved performance drastically with noise handling and design for spatial clustering. It set the benchmark for modern clustering. It introduces a sequential algorithm designed to discover clusters of arbitrary shape. Many versions of DBSCAN were proposed over the years with improvements in efficiency, accuracy and the 'power' of the algorithm. Reference [12] proposes a sampling-based DBSCAN which improve time efficiency without compromising accuracy. But, there was a problem with this. The sampling cluster might sometimes not represent the population and the clustering deviates from the expected result. Reference [13] presents a hybrid DBSCAN algorithm called l-DBSCAN which uses two *prototypes* to cluster at coarser and finer levels. With this setup, the algorithmic time efficiency and accuracy is greatly improved. Reference [14] introduces ST-DBSCAN which incorporates extensions of DBSCAN to discover clusters for spatial, non-spatial and temporal data as opposed to just spatial data by its parent algorithm. With all the above improvements, the disadvantages of the original DBSCAN were mitigated. Reference [15] uses rough-set theory to create a hybrid clustering technique to derive *prototypes* using the *leader's clustering* method and use the *prototypes* to derive density based clusters. This split allows a reduction in time complexity from $O(n^2)$ to $O(n)$. Reference [16] introduces MR-DBSCAN which uses the *MapReduce* parallel programming platform to create an efficient implementation of DBSCAN.

Reference [17] presents a revised version of DBSCAN that considerably improves DBSCANs performance in dense adjacent clusters. Reference [18] presents G-DBSCAN which consists of a GPU accelerated algorithm for density-based clustering. It is evident that the DBSCAN algorithm has evolved since its inception in 1996 but one of its core problems, the presence of parameters which are time-consuming to estimate, is yet to be solved.

Other density based clustering algorithms have also found success, such as the one in [19]. Its algorithm, DBRS, incorporates random sampling and checks a points neighborhood to decide whether a point belongs to a cluster or not. Reference [20] presents DBCLASD, a non-parametric algorithm which can form clusters of arbitrary shape by analyzing the distance distributions between data points.

B. Definitions

1) front-runners: front-runners are defined as the points which are "active" during the course of the algorithm. They are the points which exist on the periphery of the cluster.

2) Dormant points: Dormant points are points which are not "active". All the points in the cluster which are not *front-runners* are considered as dormant points. They are called so because we don't calculate distances to dormant points during the clustering process.

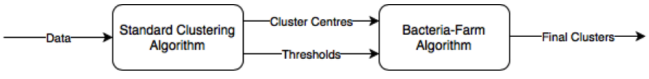

Fig. 1. Flow of control in Bacteria-Farm

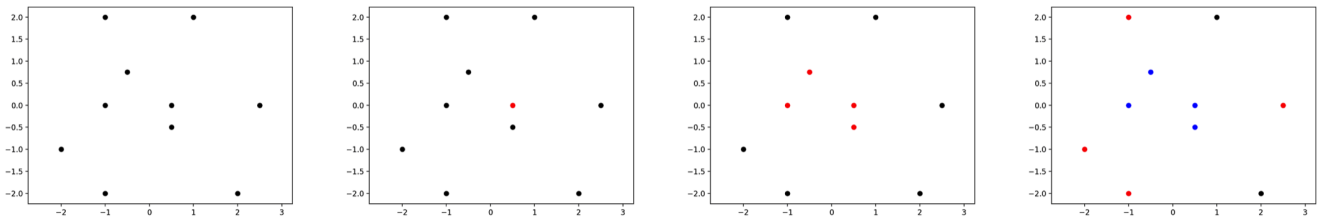

Fig. 2. Illustration of the transition of *front-runners* in the Bacteria-Farm algorithm.

III. A New Modular Spatial Clustering Algorithm

A. Working of the Algorithm

The model is divided into two phases:

In the first phase, we sample a portion (typically 20 percent) randomly from the data and apply a standard clustering algorithm on it. This presents flexibility in our model as every distribution of data has its own optimized algorithm which works wonderfully on it. Once the standard algorithm clusters the sample, we extract two parameters from the result - the clustering *centroids* and the proportion of data points in each of the clusters. The proportions act as the threshold for each cluster used in the second phase of the model. *Figure 1* shows the flow of control in our algorithm.

In the second phase, the core algorithm is executed. It starts with the *centroid* and expands outward. We have defined a parameter called as *front-runners* which are typically the surface points in a cluster. The distance between every point in the data and the *front-runners* is calculated and the nearest point to the *front-runners* (and hence, to the cluster as they represent the cluster) is selected. This point is included into that cluster. Initially when the number of points in the cluster are less than the number of *front-runners* required, every new point in the cluster becomes a *front-runner*. In the later stages when the number of points in the cluster exceeds the number of *front-runners*, **the *front-runner* which is closest to the recently selected point goes dormant and is replaced by the new point as the new *front-runner***. This ensures that the surface points stay as the *front-runners* and the number of *front-runners* stay constant. *Figure 2* illustrates the growth of the cluster in Bacteria-Farm. Iteratively, new points are added to the cluster and the *front-runners* are constantly updated till the exit condition - the number of points in the cluster is equal to the threshold of that cluster (calculated in the first phase) - is satisfied.

Once both phases are completed, the clusters are separated from the data and the remaining points - which is noise - are discarded.

With the flexibility to specify the number of points a cluster can include in itself, a unique property is observed: The difference between the total number of points in the data and the sum of the number of points clustered, can be defined as the noise in the data. We use this property for noise specification in the model. When X percent of data is specified as noise to the model, the model excludes X percent of the total data when the proportions of points in each cluster are calculated.

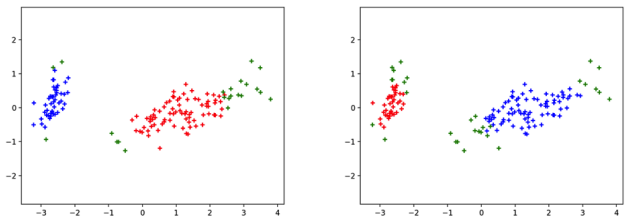

Fig. 3. The Iris data set with 15 percent and 20 percent noise specification respectively.

Suppose there are two clusters with 60 percent and 40 percent as their proportions of number of points. Assume that the noise specified for the model is 10 percent. With the exclusion of noise, the new proportions are 60 percent and 40 percent of the remaining data i.e., 54 percent and 36 percent of the total data. It can be observed that this restriction is on the exit condition of the algorithm and hence, guaranteed (from the design) that the noise specified is excluded only from the periphery of the cluster.

Figure 3 illustrates this property of Bacteria-Farm algorithm. It can be observed that as noise specification increases, the peripheral points are labelled as noise instead of the inner points of a cluster.

B. Pseudo-code of the Algorithm

In the following, we present the pseudo code of the Bacteria-Farm algorithm. Important details are explained separately. The major functions required by Bacteria-Farm are expanded before the core algorithm.

Algorithm 1 sampling

Input data set *df*, number of front-runners n_{fr}, noise factor *n*
Output centroids *centroids*, thresholds *thresholds*

sample = take a sample of data from the data set *df*.
centroids, *threshold* = *retrieve-parameters*(*sample*, *n*).
return *centroids*, *thresholds*

Algorithm 2 retrieve-parameters

Input sample *s*, noise factor *n*
Output centroids *centroids*, thresholds *thresholds*

Use a standard clustering algorithm to fit the sample data and obtain labels for them. The data is divided into *clusters* from this algorithm.
for *cluster* in *clusters* **do**
  centroid = mean of all instances in the *cluster*.
  Append *centroid* to the list *centroids*.
  threshold = ratio of number of points in the *cluster* to the number of points in the *sample* multiplied by the noise factor *n*.
  Append *threshold* to the list *thresholds*.
end for
return *centroids*, *thresholds*

Algorithm 3 Bacteria-Farm

Input data set *df*, number of front-runners n_{fr}
Output clusters *clusters*

centroids, *thresholds* = *sampling(df,*n_{fr}*)*
Initialize a list of lists *clusters* of size *centroids*, to empty lists and each inner list is of size n_{fr}.
for centroid *c* in list *centroids* **do**
  frs = list of corresponding front-runners for each *c* and initialize the first front-runner as *c* itself.
  while true **do**
    if size of cluster corresponding to *c* is greater than threshold corresponding to *c* **then**
      break
    end if
    minInstance = get closest point to the set of front-runners *frs* and add it to current cluster in *clusters*.
    Replace the closest front-runner *fr* (in *frs*) to the *minInstance*, with *minInstance* itself.
  end while
end for

1) Additional Explanation: In retrieve-parameters, we subtract the noise percentage from 100 and multiply that factor with the proportion of points in a cluster obtained from the standard clustering algorithm. This is the threshold for each cluster used in second phase of Bacteria-Farm algorithm.

In the core Bacteria-Farm algorithm, we calculate the distance from the front-runners *frs* to every point in the data set *df* and pick the one with the smallest *Euclidean* distance as *minInstance*. The algorithmic complexity of this step is $O(log(n))$ by using spatial indexing.

Once the closest point minInstance is chosen, the corresponding front-runner *fr* in *frs* which is closest to *minInstance* is replaced with *minInstance* as the new front-runner and this instance is included to the current cluster.

IV. Performance Evaluation

We evaluate Bacteria-Farm according to the major requirements of clustering algorithms - efficiency, input parameters and ability to cluster data of arbitrary shape. We choose *Silhouette Coefficient* and *Calinski-Harabasz Index* as the performance metrics for evaluation. We compare Bacteria-Farm with established algorithms such as DBSCAN and *k*-Means in terms of efficiency and the fore-mentioned performance metrics.

A. Choice of comparison algorithms

We have selected DBSCAN and *k*-Means for comparison as they are the most popular density-based and partitioning clustering algorithms respectively. We chose DBSCAN as it is an established algorithm for clustering data of arbitrary shape and size. Over time, many versions of DBSCAN have been proposed but the core algorithm remains the same. Hence, we decided to choose the vanilla version of DBSCAN for comparison with the vanilla version of Bacteria-Farm. We have chosen *k*-Means as its time complexity is $O(n)$ and helps in estimating the real performance of the Bacteria-Farm algorithm.

B. Choice of performance metrics

We have selected *Silhouette Coefficient* and *Calinski-Harabasz Index* as the two performance metrics. The definition of these metrics along with the reasons for their selection are given below.

1) Silhouette Coefficient: Let $a(i)$ be the average distance between a datum i and all other points in its cluster. $a(i)$ is a measure of the intra-cluster distance. Lower the value of $a(i)$, denser is the cluster and better the assignment of $a(i)$ to the cluster.

Let $b(i)$ be the average distance between the datum i and all other points in any other cluster in the data set. $b(i)$ is a measure of the inter-cluster distance. Higher the value of $b(i)$, better the separation of clusters.

Silhouette Coefficient $s(i)$ can be defined as :

$$s(i) = \begin{cases} 1 - a(i)/b(i) & \text{if } a(i) \leq b(i) \\ 0 & \text{if } a(i) = b(i) \\ a(i)/b(i) - 1 & \text{if } a(i) \geq b(i) \end{cases}$$

For $s(i)$ close to 1, it implies $a(i) \ll b(i)$. A small $a(i)$ means that a datum i is closely matched with other data in the same cluster and a large $b(i)$ indicates that the datum i is poorly matched with data present in other clusters.

Therefore, a high value of $s(i)$ can conclude that the data has clustered well. We chose *Silhouette Coefficient* as it has been a good indicator of clustering performance, in the past.

2) Calinski - Harabasz Index: Let SS_B be the overall inter-cluster variance, SS_W be the overall intra-cluster variance, k be the number of clusters and N be the number of points in the data set.

Calinski-Harabasz Index CH_k for k clusters (with standard notations) can be defined as :

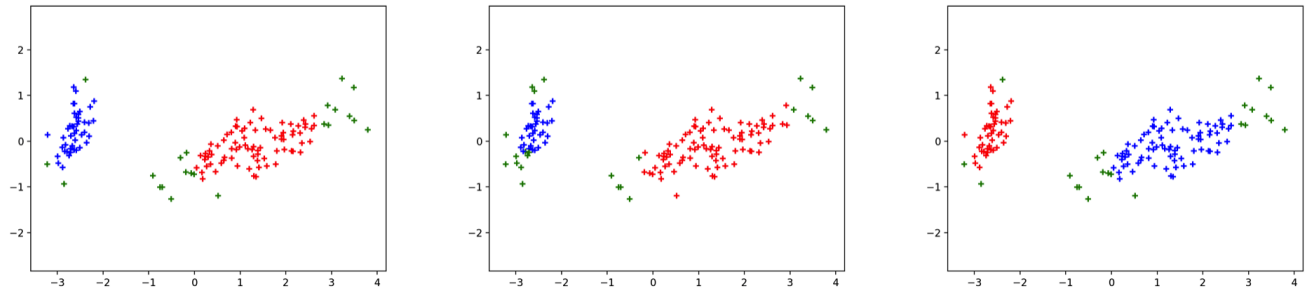

Fig. 4. Comparison of results with varying number of *front-runners* (3,5,7) for the Iris data set to demonstrate robustness.

$$CH_k = \frac{SS_B}{SS_W} \times \frac{N-k}{k-1}$$

A high value for the first fraction in the above equation indicates that $SS_B \gg SS_W$. The inference is that the data between clusters are very different from each other and the ones within the same cluster are very similar. This is another indication of good clustering in the data. We chose *Calinski-Harabasz Index* because we can infer that the ratio of variances in the equation is a good indicator of the compact-ability of a cluster. This is a similar to the previous metric.

Overall, we have taken two performance indicators (along with time taken for the algorithm to execute) to measure the overall performance of Bacteria-Farm for convex as well as non-convex data.

3) Input parameters: It is hard to optimize the parameters for given data in many clustering algorithms. For example, DBSCAN has two parameters - *Epsilon* distance and *minPts*. To explain those parameters, we define a *core* point. A *core* point is a point which has a minimum number of points within a certain distance from itself. *Epsilon* distance specifies how close the points should be to a core point to consider those points as a part of the cluster and *minPts* specifies how many points should be in the *Epsilon* distance from a point for it to become a core point. Both these parameters are continuous values and are hard to optimize. Users usually resort to running the algorithms multiple times to arrive at the optimal values for these parameters or use optimization techniques to obtain the optimized parameters. This process is time consuming and hence, there is a need for "better" parameters.

On the other hand, *k*-Means require the number of clusters *apriori* and this is hard to obtain from visual inspection in higher dimension data.

After considering these problems, we have devised a different approach to obtain the parameters inherently from our model. As discussed earlier, we have two parts in our model - the first phase which runs a standard algorithm to obtain the clusters and the second phase which runs the core Bacteria-Farm algorithm. Due to the modular design of the model, we can use a parameter-less algorithm in the first phase to obtain clusters. Once the clusters are obtained, the *centroids* of those clusters are sent to the second phase.

Effectively, we have two parameters for Bacteria-Farm : the percent of noise to be specified and the number of *front-runners* desired. Also, the robustness of the model allows for some error in choosing the number of *front-runners*. *Figure 4* illustrates the robustness in the algorithm with varying number of *front-runners*. Many clustering models including DBSCAN fail to account for this error and thus are highly sensitive to small changes in their parameters.

4) Ability to cluster data of arbitrary shape: Spatial databases may contain convex, non-convex and other data of arbitrary shape, and good clustering algorithms can cluster any data sufficiently well. We will evaluate DBSCAN and Bacteria-Farm with respect to their ability to cluster data of arbitrary shape.

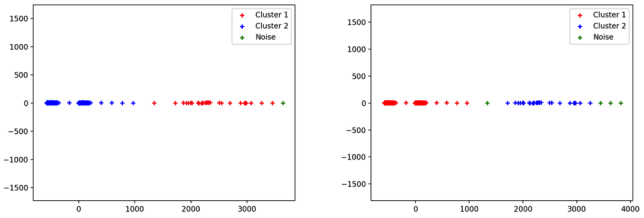

Fig. 5. Visual comparison of performance between DBSCAN and Bacteria-Farm for a data set of arbitrary shape.

We consider a small real data set of small dimensions to illustrate the clustering in Bacteria-Farm algorithm. *Figure 5* shows the clustering comparison between DBSCAN and Bacteria-Farm for the Alcohol data set [21] with 411 instances.

With a noise specification of 1.24 percent for the Bacteria-Farm algorithm, it can be verified visually that the data points are assigned to their correct clusters. It can also be observed that the data points "seen" as noise are not included in any of the clusters in the Bacteria-Farm algorithm. Although it can be observed that some points are mis-clustered, they belong to a minority.

5) Efficiency: DBSCAN and Bacteria-Farm are comparable in time complexity of $O(nlog(n))$, by using spatial indexing. Whereas, K-Means has a time complexity of $O(n)$. All the measurements are done on a single machine to maintain consistency.

Algorithm	Silhouette Coefficient	Calinski-Harabasz Index
K-Means	0.5842	14852.1314
DBSCAN	0.6103	16657.9614
Bacteria-Farm	0.6167	1483.1049

TABLE I
COMPARISON OF ALGORITHMS

6) Performance: We have used 92 different data sets with 200 to approximately 1000 instances to compare the run times and the other performance metrics. *Table I* tabulates the comparison of performance metric averages between K-Means, DBSCAN and Bacteria-Farm over these data sets. We have chosen data with a small number of instances to verify the inferences and clustering progression in all the three algorithms. It can be derived from the algorithm in the earlier section that the time complexity of Bacteria-Farm is $O(nlog(n))$ (by using Spatial Indexing to retrieve distances between points) and we expect their real run times to be in the same neighborhood.

Since the tasks were not too CPU intensive, the performance comparisons were done on a local computer (Apple MacBook Pro Early 2013) with an Intel HD Graphics 4000 GPU.

The *Silhouette Coefficient* of Bacteria-Farm and DBSCAN are comparable, with Bacteria-Farm performing slightly better. This indicates that the algorithm is able to form clusters with high inter-cluster distance and low intra-cluster distance. Since most linear real datasets such as the one in *Figure 5* have a low *Silhouette Coefficient*, we have used an average value to compare the overall performance of the algorithms.

On the other hand, a *Calinski - Harabasz* indices of the algorithms are not comparable. This is due to the linear increase of the index with the size of the data. Though it offers some degree of comparison, it is not as effective as *Silhouette Coefficient*. Intuitively, the metrics should have comparable values for data of same shape but of different sizes. And due to this dependency on the variance of size of the data, *Calinski - Harabasz* is used only as a secondary metric.

V. CONCLUSION AND FUTURE WORK

With the modular design, the versatility of the algorithm to cluster the target distribution of data has a significance improvement. With different "underlying" algorithms suited for different distributions and types of data, suitable parameters of the data are transferred to the "core" Bacteria-Farm algorithm which uses a novel approach to effectively cluster the data.

In this paper, we introduce a novel clustering algorithm / model Bacteria-Farm which is designed to handle noise, introduce parameters which are easy to optimize and display superior performance. Our notion of a cluster depends on the limit of points a cluster can accommodate. The core algorithm is designed to work well with convex and non-convex data. Furthermore, the robustness of the algorithm can be demonstrated by varying the values for *front-runners* as it does not significantly alter the performance of the algorithm. Also, Bacteria-Farm has a provision to specify the amount of noise to exclude from the clusters. As the clusters extend outward, it is guaranteed that the labelled noise mirror the actual noise in the data. This unique property of noise specification enables applications to generate suitable clusters.

Experiments on real data demonstrate that Bacteria-Farm performs *better** than algorithms such as DBSCAN and K-Means in the chosen evaluation metrics. The results also indicate Bacteria-Farm performs well for real data.

Future research can include further optimizing the time complexity of retrieval of the closest point to a cluster from $O(log(n))$ and thus, drastically improve the performance of Bacteria-Farm. Also, *projected clustering* can be used to improve Bacteria-Farm for sparse, high dimensional data.

The use of modular design to improve efficiency in other clustering algorithms, and by extension, other paradigms can be explored. Furthermore, we will consider the application of Bacteria-Farm on non-spatial data and explore suitable designs to improve the performance of clustering in non-spatial data.

* The algorithm performs better than K-Means and is on par, if not better, when compared to DBSCAN; with respect to performance metric Silhouette Coefficient.

ACKNOWLEDGEMENT

We thank the Department of Computer Science and Engineering, PES University for providing the necessary resources to experiment with our models.

REFERENCES

[1] M. Ester, H.P. Kriegel, J. Sander, and X. Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise," in Proc. of 2nd International Conference on Knowledge Discovery and Data Mining, 1996, pp. 226231.

[2] Wikipedia contributors, "Cluster analysis," Wikipedia, The Free Encyclopedia, 2018.

[3] J. A. Hartigan, "Clustering algorithms," Wiley, 1975.

[4] J. A. Hartigan and M. A. Wong, "A k-means clustering algorithm," JSTOR: Applied Statistics, vol. 28, no. 1, 1979, pp. 100108.

[5] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu, "An Efficient k-Means Clustering Algorithm: Analysis and Implementation," IEEE Trans. Pattern Anal. Mach. Intell., vol. 24, no. 7, 2002, pp. 881892.

[6] K. Alsabti, S. Ranka, and V. Singh, "An Efficient K-Means Clustering Algorithm," 1998.

[7] A. Likas, N. A. Vlassis, and J. J. Verbeek, "The global k-means clustering algorithm," Pattern Recognition, vol. 36, no. 2, 2004, pp. 451461.

[8] K. Wagstaff, C. Cardie, S. Rogers, and S. Schrdl, "Constrained K-means Clustering with Background Knowledge," ICML, 2001, pp. 577584.

[9] K. A. Abdul Nazeer, M. P. Sebastian, "Improving the Accuracy and Efficiency of the k-means Clustering Algorithm," Proceedings of the World Congress on Engineering 2009 Vol I, 2009.

[10] K. Krishna and M. N. Murty, "Genetic K-Means Algorithm," 1999.

[11] Y. Lu, S. Lu, F. Fotouhi, Y. Deng, and S. J. Brown, "FGKA: a fast genetic K-means clustering algorithm," Proceedings of the 2004 ACM symposium on Applied computing, 2004, pp. 622623.

[12] B. Borah, D. K. Bhattacharyya, “An improved sampling-based DBSCAN for large spatial databases,” International Conference on Intelligent Sensing and Information Processing, 2004. Proceedings of, 2004.
[13] P. Viswanath and R. Pinkesh, “l-DBSCAN: A Fast Hybrid Density Based Clustering Method,” ICPR (1), 2006, pp. 912915.
[14] D. Birant and A. Kut, “ST-DBSCAN: An algorithm for clustering spatial-temporal data,” Data Knowl. Eng., vol. 60, no. 1, 2007, pp. 208221.
[15] P. Viswanath, V. Suresh Babu, “Rough-DBSCAN: A fast hybrid density based clustering method for large data sets,” Pattern Recognition Letters, vol. 30, issue 16, 2009, pp. 14771488.
[16] Y. He et al., “MR-DBSCAN: An Efficient Parallel Density-Based Clustering Algorithm Using MapReduce,” ICPADS, 2011, pp. 473480.
[17] T. N. Trana, K. Drab, M. Daszykowski, “Revised DBSCAN algorithm to cluster data with dense adjacent clusters,” Chemometrics and Intelligent Laboratory Systems vol. 120, 2013, pp. 9296.
[18] G. Andrade, G. S. Ramos, D. Madeira, R. S. Oliveira, R. Ferreira, and L. C. da Rocha, “G-DBSCAN: A GPU Accelerated Algorithm for Density-based Clustering,” ICCS, vol. 18, 2013, pp. 369378.
[19] X. Wang and H. J. Hamilton, “DBRS: A Density-Based Spatial Clustering Method with Random Sampling,” PAKDD, vol. 2637, 2003, pp. 563575.
[20] X. Xu, M. Ester, H. P. Kriegel, and J. Sander, “A Distribution-Based Clustering Algorithm for Mining in Large Spatial Databases,” ICDE 98: Proceedings of the Fourteenth International Conference on Data Engineering, 1998, pp. 324331.
[21] Project MOSAIC contributors, “Alcohol Consumption per Capita,” Available at: https://vincentarelbundock.github.io/Rdatasets/csv/ mosaicData/Alcohol.csv