High quality clustering of big data and solving empty-clustering problem with an evolutionary hybrid algorithm

Jeyhun Karimov
Computer Engineering Department
TOBB University of Economics and Technology
Ankara,Turkey
jkarimov@etu.edu.tr

Murat Ozbayoglu
Computer Engineering Department
TOBB University of Economics and Technology
Ankara,Turkey
mozbayoglu@etu.edu.tr

***Abstract*—Achieving high quality clustering is one of the most well-known problems in data mining. k-means is by far the most commonly used clustering algorithm. It converges fairly quickly, but achieving a good solution is not guaranteed. The clustering quality is highly dependent on the selection of the initial centroid selections. Moreover, when the number of clusters increases, it starts to suffer from "empty clustering". The motivation in this study is two-fold. We not only aim at improving the k-means clustering quality, but at the same time not being effected by the empty cluster issue. For achieving this purpose, we developed a hybrid model, $H(EC)^2S$, Hybrid Evolutionary Clustering with Empty Clustering Solution. Firstly, it selects $representative$ points to eliminate Empty Clustering problem. Then, the hybrid algorithm uses only these points during centroid selection. The proposed model combines Fireworks and Cuckoo-search based evolutionary algorithm with some centroid-calculation heuristics. The model is implemented using a Hadoop Mapreduce algorithm for achieving scalability when faced with a Big Data clustering problem. The advantages of the developed model is particularly attractive when the amount, dimensionality and number of cluster parameters tend to increase. The results indicate that considerable clustering quality performance improvement is achieved using the proposed model.**

***Keywords*-clustering; k-means; evolutionary algorithms; Cuckoo search; Fireworks algorithm; Hadoop; Mapreduce**

I. Introduction

Even though k-means is not the best performing data clustering algorithm, it is still by far the most widely-used one due to its simplicity, scalability and convergence speed. However, depending on the initial starting centroid points, significantly different results can be obtained, so the algorithm is highly sensitive to its random centroid selection.

Meanwhile there is also another notable problem with k-means algorithm. When the number of cluster formations increase, the possibility of obtaining empty clusters (locations that do not have any data points associated with the clusters) also increases at each iteration. This becomes an unavoidable issue when $k >> 1$. This issue was not addressed thoroughly in most of the studies.

In this study we not only aim to improve the clustering quality of k-means, but also provide a methodology that can overcome "empty clustering" problem without jeopardizing the convergence speed, so the proposed model could be used for all clustering problems regardless of the number of data points, number of data dimensions and number of clusters. We achieved this by using an initial centroid selection algorithm based on a hybrid model that is a combination of evolutionary algorithms (Fireworks and Cuckoo search) and some additional heuristics.

The rest of this paper is as follows. After this brief introduction, we provide the literature review which included previous work performeds on k-means performance and quality improvement in Section II. Section III has the model description. In Section IV, we show the performance of our algorithm by providing charts for clustering quality improvement compared with the standard k-means and other modified models in the literature. We have the conclusion in the last section.

II. Literature Review

Because of its simplicity and applicability k-means [1] is the most widely used algorithm to cluster data. There are some studies implemented on optimizing different objectives of k-means algorithm such as Euclidean k-medians [2], [3] and geometric k-center [4]. Minimization of the sum of distances to the nearest center is the goal for Euclidean k-medians, and minimization of the maximum distance from every point to its nearest center is the one for geometric k-center version. Another research was done to seek a better objective function of k-means [5]. Although there are different versions of k-means that might have advantages, parallelization of algorithm in a single machine resulted in significant performance improvement [6], [7]. Achieving the parallelization over multiple machines results in even better improvements. The framework MapReduce [8], [9] provides significant improvements to scalable algorithms.

There has been several studies for clustering large scale data on distributed systems in parallel on Hadoop [10]. One such approach is Haloop [11], which is a modified version of the Hadoop making the task scheduler loop-aware and by adding various caching mechanisms. Another approach

to cluster data in a distributed system was using Apache Mahout library [12]. Moreover, clustering of big data can be done on cloud also. In [13], the tests were running on Amazon EC2 instances and the comparisons were made to realize the gain between the nodes. Esteves et al. made comparisons over k-means and fuzzy c-means for clustering Wikipedia large scale data set [14]. Both in [14] and [13], the authors used Apache Mahout for clustering data.

Fireworks algorithm, which was inspired by observing fireworks explosion, is a recent meta-heuristic that was proposed by Tan et al [15]. Authors showed that it outperforms Standard PSO and Clonal PSO in experiments. Enhanced Fireworks Algorithm is the improved version of the Fireworks algorithm [16], which we used in our model. Cuckoo search is another meta-heuristic that was proposed recently by Yang et al. [16], which was inspired by obligate brood parasitic behavior of some cuckoo species in combination with the Levy flight behavior of some birds and fruit flies.

III. Proposed Model

Our model, $H(EC)^2S$, Hybrid Evolutionary Clustering with Empty Clustering Solution, consists of 4 parts: Representative Construction (RC) part, Enhanced Fireworks algorithm for Clustering (EFC) part, Cuckoo Search for Clustering (CSC) part and k-means part. Firstly, instance reduction is done to select representatives for existing data. This part does the main job to eliminate Empty Clustering problem, because in latter parts, the centroids are selected among representative points only. Secondly, in EFC part, the solution space is searched using Enhanced Fireworks algorithm. Thirdly, in CSC part, we construct new solutions and based on Cuckoo Search algorithm, place solutions to "host nests". Finally, we pass the selected best firework to the k-means and converge. The pseudo-code of the proposed model is given in Algorithm 2. Some notations used in this paper are given in Table I.

A. RC part

The main purpose of RC part is to select "representative" instances among data which eliminate the instance count significantly. The important point is that, in RC part we can detect outliers and eliminate them also. Algorithm 1 shows the pseudo-code of PC-part. Lines 3-8 show the process of discretizing the values to find how many distinct data point there are at each dimension. Here $s_i[j]$ denotes the j^{th} dimension value of i^{th} data point. Equation 6 maps each data point to a representative value, where min_i and max_i represent min and max elements at i^{th} dimension respectively. To do this, it divides dimension range into F_i and finds in which portion the data is. Lines 9-13 find the representative values $rValue$ for each data point. Lines 13-16 show the elimination part of representatives. That is, if the number of belonging data points are less than the specified threshold, c_o, then the data points can be considered as outliers and representative can be deleted. Else, we find the average of all its belonging data points in each dimension and update representative value. Finally, lines-17-20 show the case where the number of representative values do not fit into the memory, they are more than predefined threshold. In this case we either select another f function which is "more" decreasing or increase c_f or do both.

Algorithm 1 shows the main intuition behind RC part, which can easily converted to MapReduce algorithm. That is, first we calculate distinct values in Mapper and send them to Reducers so that it can combine all distinct values in each dimension. Secondly, in Mapper we find each point's representative and send it to Reducer with the point itself. Reducer gets the representative and its points and calculates the average of the points.

Algorithm 1 RC-Part

Input: $s_i \in S$ data points

1: Let H_k be the HashMap related with k^{th} dimension of data points, where $k = \{1, 2...c_d\}$
2: A_k be a vector, denoting the count of distinct values in k^{th} dimension, where $k = \{1, 2...c_d\}$
3: **for all** $s_i \in S$ **do**:
4: **for** $j = 1$ to c_d **do**:
5: $x_i[j] \leftarrow$ Round $s_i[j]$ to c_f decimal places.
6: Put H_j a new entry with key=x_i, value=1. If already exists, increment value.
7: $A_i \leftarrow \sum_{e_j \in H_i} e_j.value, \forall(i = \{1, 2, ...c_d\})$
8: $F_i \leftarrow f(A_i) \forall(i = \{1, 2, ...c_d\})$
9: **for all** $s_i \in S$ **do**:
10: Compute rID^k $k = 1, 2, ..c_d$ using Equation 6
11: $R_{id} \leftarrow R_{id} \cup rID$
12: $D_{(rID)} \leftarrow D_{(rID)} \cup s_i$
13: **for all** $rID_i \in R_{id}$ **do**:
14: **if** $||D_{(rID_i)}|| \leq c_o$ **then**
15: Data points belonging to representative rID_i are outliers, so eliminate rID_i
16: **else** $rValue_i^k \leftarrow \frac{\sum_{s_j \in D_{(rID_i)}} s_j[k]}{||D_{(rID_i)}||}, \forall k = 1, 2, ...c_d$
17: **if** $representatives$ are higher than predefined threshold (do not fit in memory) **then**
18: $f() \leftarrow f(f())$
19: Increase c_f
20: Restart algorithm

B. EFC part

In this part, we use Enhanced Firework Algorithm (EFW) to search the solution space for "good" initial points. Initially, the model starts with random solution, converts it to $representative$ using Equation 6 and calculates amplitude and and sparks locations. The $s_j[i]$ represents i^{th} dimension

Table I
NOTATIONS.

$rID_i^k \in R_{id}$	The ID of i^{th} representative data point, possibly int values denoting the range index, where k denotes dimension
$rValue_i^k \in R_{value}$	The actual value of i^{th} representative data point, where k denotes dimension
$f : R \to R$	Decreasing function from real numbers to real numbers
c_f	Constant denoting how many decimal places should data be rounded
max_d	Maximum data point in d^{th} dimension
min_d	Minimum data point in d^{th} dimension
N_i	Number of distinct data points in i^{th} dimension
F_i	Number of distinct data points in i^{th} dimension after applying f function to N_i
c_d	Dimension size of a data point
c_o	Threshold value for "representative" values to be used in outlier detection.
$D_{(rID)}$ or $D_{(rValue)}$	Set of belonging data points to representative rID or $rValue$
$E[i][j][n][k]$	Similarity matrix, that shows the shared data points count between i^{th} firework's j^{th} cluster and n^{th} firework's k^{th} cluster
$fw_i^j \in F$	Firework (a possible solution) of index i in set F, where k is the centroid of firework.
K	Number of clusters

Algorithm 2 $H(EC)^2S$

1: Run RC part
2: **for** $t = 1$ to $iter_{max}$ **do**:
3: Run EFC
4: Run CSC
5: Run k-means with the best firework

value for some $s_j \in S$, min_i and max_i are minimum and maximum values in i^{th} dimension. After that, while computing their fitness values, it also derives 4 dimensional similarity matrix E. The important point is that, we use the reduced data set with only representatives for selecting centroids, R_{value}, derived in RC part. Then, we find the amplitude and spark sizes in each dimension of firework and update fireworks. After updating, we convert them to representative values with Equation 6. The pseudo-code for this part is given in Algorithm 3.

The construction of firework fw as a solution is done via concatenating the k centroids. That is, because we seek a solution of k centroids to cluster data, we represent the solution, fw, as a combination of k cluster centroids. Equation 1 controls the A_{min}^k, the amplitude of firework. It starts with a higher value initially to explore the solution space and diminishes gradually. Here t refers the number of function evaluation at the beginning of the current iteration, and $evals_{max}$ is the maximum number of evaluations. A_{init} and A_{final} are the initial and final minimum explosion amplitude, respectively. Equation 2 controls the amplitude so that it does not become too small. Equation 3 calculates the value of amplitude, where g is fitness function, $y_{max} = max(g(X_i))$ and $y_{min} = min(g(X_i))$ are the two constants to control the explosion amplitude and the number of explosion sparks, respectively, and ϵ is the machine epsilon. Equation 4 calculates the number of sparks.

Algorithm 3 EFC-part

1: Initialize random fireworks $fw_i \in F$
2: Calculate fitness of each firework $ft_i \in FT$ using Equation 5, where M is positive constant, s_i^j is j^{th} dimension value of s_i and c_j is j^{th} centroid which s_i^j belong to. While doing this, update the value of E array, such that, if data point belongs to fw_i^k and fw_j^m then, $E[i][k][j][m] \leftarrow E[i][k][j][m] + 1$
3: Calculate s_i, the number of explosions for each firework using Equation 4
4: Calculate A_i for every firework using Equation 3, 2 and 1
5: **for all** fw_i in F **do**:
6: **for** $j = 1$ to s_i **do**:
7: $\Delta X_i^k \leftarrow round(rand(0,1)) * A_i * rand(-1,1) \forall(k)$, calculate displacement, where $1 \le k \le K$ is cluster count
8: **if** X_i^k is out of bounds **then**
9: $X_i^k \leftarrow X_{min}^k + |X_i^k| \% (X_{max}^k - X_{min}^k)$
10: $X_i^k \leftarrow X_i^k + \Delta X_i^k \ \forall(1 \le k \le K)$
11: Perform k-means and update E array
12: Calculate fitness values of fireworks $fw_i \in F$ using Equation 5

$$A_{\min}^k(t) = A_{init} - \frac{A_{init} - A_{final}}{evals_{max}} \sqrt{(2 * evals_{max} - t)t} \tag{1}$$

$$A_i^k = \begin{cases} A_{\min}^k & \text{if } A_i^k < A_{\min}^k, \\ A_i^k & otherwise. \end{cases} \tag{2}$$

$$A_i = \hat{A} \cdot \frac{g(X_i) - y_{min} + \varepsilon}{\sum_{i=1}^{||F||} (g(X_i) - y_{min}) + \varepsilon} \tag{3}$$

$$s_i = M_e \cdot \frac{y_{max} - g(X_i) + \varepsilon}{\sum_{i=1}^{||F||} (g_{max} - f(X_i)) + \varepsilon} \tag{4}$$

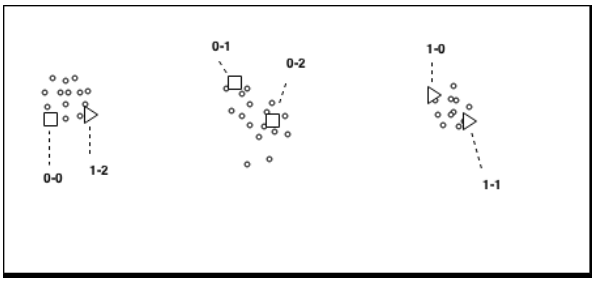
$$J = \frac{\sum_{j=1}^{k} \sum_{i=1}^{n} \left\| s_i^j - c_j \right\|^2}{M} \tag{5}$$

$$rID^i \leftarrow \frac{s_j[i] - min_i}{\frac{max_i - min_i}{F_i}} \forall(i = 1, 2, ...c_d) \tag{6}$$

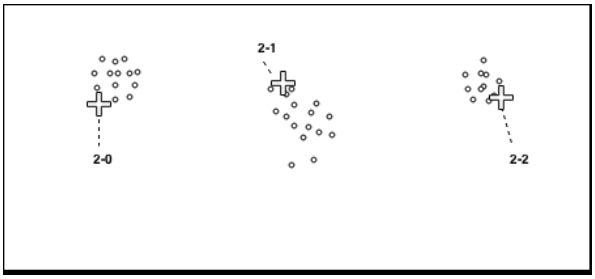
C. CSC part

In this part, new fireworks are constructed and joined to firework set. We use E, 4 dimensional array to construct new solutions and use Cuckoo Search algorithm to select fireworks in F to pass to the next generation. To construct the new fireworks, k different centroids from different fireworks are found and joined. The important point is that, k different centroids cannot share any data points, because in a particle centroids do not have any common data points.

For example, Figure 1 shows the procedure of creating new firework with k=3 and $||F||$=2. Figure 1(a) and Figure 1(b) are before-after situations of constructing new firework. In Figure 1(a), one firework is denoted with rectangle and another with triangle. CSC computes that centroids denoted with $0-0$, $1-1$ and $0-1$ do not share any common data points,where $f-k$ means k^{th} centroid of f^{th} firework. As a result, the new particle can be created as shown in Figure 1(b) with plus sign. Indeed, if this was a separate particle, it would have better fitness value than the existing ones.



(a) Before CSC part.



(b) After CSC part.

Figure 1. Demonstration of creation of new particle in SS part.

We use the similarity array E to form a graph consisting of vertices denoted with centroids and edges having value of that vertex's fitness value. Since the similarity array keeps the sharing data point count between centroids, if the shared element count is zero, then the corresponding centroids can be concatenated to form a new firework. As can be seen from Figure 2, the source node is the double lined circle. It has $||F|| * K$ child nodes, since that is precisely the total number of centroids within the firework set. The source node do not share any points with any of the centroids. As a result, some path, with the minimum sum of edge values, consisting of K consecutive nodes is found, where the incoming edge value of a particular node is its fitness value. Important point is that, the K consecutive nodes can be anywhere in the graph, might not necessarily begin from the source node. To solve this problem, dynamic programming is used and all possibilities are tried like brute force, but in a "careful" way.

$$DP(k, c_p^k) = argmin(DP(k-1, n_p^k) + f_p^k, DP(k, n_p^k)) \forall (n_p^k) \quad (7)$$

As can be seen from Equation (7), finding the minimum length k united nodes is implemented recursively, where n_p^k is the child node of c_p^k. The number of nodes needed and the current node are the two arguments given to function DP. Finding the minimum path length consisting of k nodes can be done by either taking the current node into consideration, running the same function for the rest of its child nodes with $(k-1)$ needed nodes and summing them, or not taking the current node into the consideration and running the function with K and all of current node's child nodes arguments, and giving the minimum of this results as an output. We used memoization in order to get the advantages of dynamic programming. That is, if some value of $DP(k, n_p^k) = c$ then it is stored, and after if it is needed, then the answer is retrieved in constant time and used.

After finding new solutions we use Cuckoo Search to place them. The pseudo-code for Cuckoo search is given in Algorithm 4.

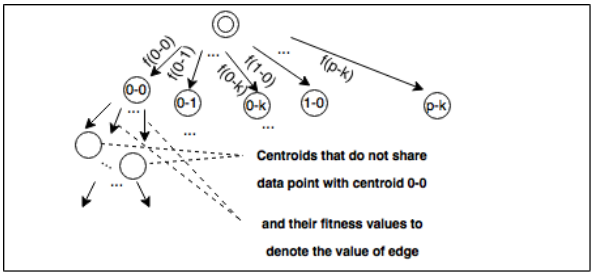


Figure 2. Intuition behind constructing new firework using graph.

IV. Experiments

We used two data sets for test our model. First data set, DS1 [17], is public data set. Second data set, DS2, is the ATM logs data set, we used for case study. First data set contains 17389 instances and 16 attributes. DS2 contains 20 million instances consisting of 50 attributes. DS2 contains the user profile vectors for ATM customers. To construct a vector, we used the count of the customers' clicks for each transaction for a period of time. To test our results, we used Cloudera's Hadoop distribution on 5 Intel i5 machines.

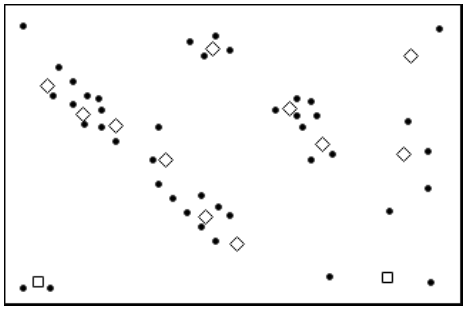


Figure 3. Demonstration of RC part

Algorithm 4 CSC-part

1: Calculate objective functions $f_{obj}(fw_i) \forall fw_i \in F$ with Equation 5.
2: Get $n_i \in N$ new firework solutions with Equation 7
3: **for** $i = 0$ to $iter_{max}$ **do**
4: Chose random n_i
5: Chose random f_i
6: **if** $f_{obj}(n_i)$ $isbetterthan$ $f_{obj}(fw_i)$ **then**
7: Replace fw_i by n_i with some probability
8: Delete p_a fraction of F and build new fireworks randomly.

We used [18], [19], [20] and our model to make comparisons. For simplicity, we called them rk-means, ck-means, $FGKA$ and $H(EC)^2S$ respectively. Because the fitness function, Equation 5, uses minimization objective, the better clustering quality is achieved with lesser fitness value.

Figure 4 shows the effect of increasing k value to the models with DS2. Here, the models with lesser fitness values are better because fitness function's, Equation 5, objective is minimization. As can be seen, the standard k-means cannot survive to k=100 with DS2, because of empty clustering problem. The same can be said to $FGKA$ algorithm after k=100 and etc. The reason is that, DS2 data set is very crowded in specific places and solution space is very large. That is, most of the users do same operations and dividing them into $k >> 1$ clusters is very difficult. The important thing in $H(EC)^2S$ model is that as k value increases, the gap between other models increases linearly. This is because of CSC part, in which new solutions are found and placed into fireworks. So, as k gets bigger, in CSC part, we have many solutions to select among, one of which can be the optimum solution as well. Table II shows the numerical results based on this experiment. Here $-$ sign indicates that the model could not find the solution because of empty clustering problem.

Figure 5 shows the dependence between data size and execution time implemented on DS1 on Hadoop environment based on 5 nodes. We simulated the data to increase its size. Because there are not scalable versions of other models

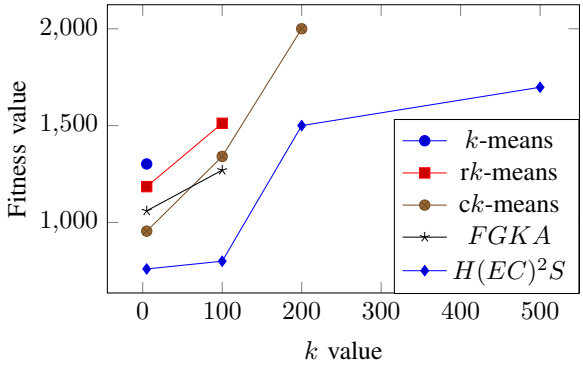


Figure 4. Effect of k to fitness function.

Table II
FITNESS VALUES OF DIFFERENT MODELS W.R.T. k VALUE.

	k-means	rk-means	ck-means	FGKA	$H(EC)^2S$
k=5	1302	1185	955	1060	760
k=100	–	1512	1341	1270	800
k= 200	–	–	2007	–	1500
k = 500	–	–	–	–	1698

given in Figure 4, we only considered k-means parallel [21] and $H(EC)^2S$. It is clear that k-means performs better in terms of performance time because of its greedy nature. However, $H(EC)^2S$ does not increase exponentially with respect to k-means. This is because of its scalability.

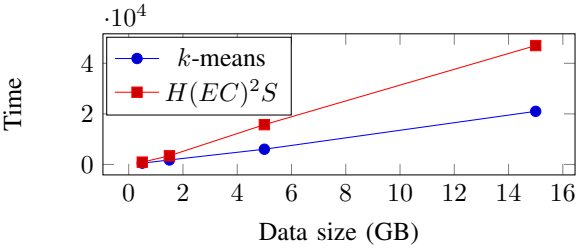


Figure 5. Effect of increasing data size to execution time.

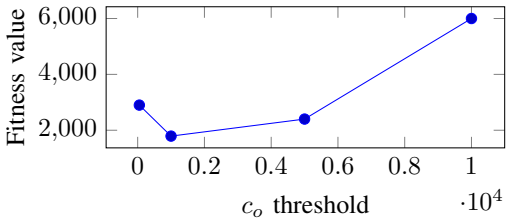


Figure 6. Effect of threshold level c_o on fitness value.

Figure 6 shows the dependence between c_o and fitness function in $H(EC)^2S$ model on DS2. Because c_o is outlier threshold, based on data set, it can vary. DS2 is very crowded data set in specific places of solution space. So,

making outlier threshold bigger, can eliminate the necessary fireworks too. Therefore, after 1000 the fitness value gets worse.

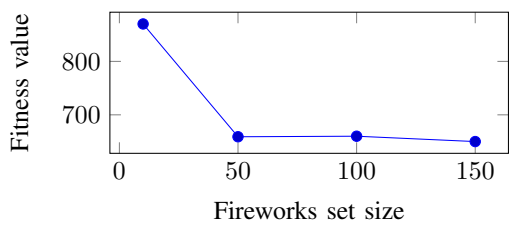


Figure 7. Effect of fireworks set size on fitness value.

Figure 7 shows the dependence between fireworks set size and fitness function value on DS1. As the fireworks set gets bigger, the probability of finding the new better solutions also increases. However, after some certain value, depending on the data set, the increase may not continue. The same applies to this Figure. We can conclude that the fireworks set size for this data set can be chosen nearly 50, because increasing further increases the time complexity also.

V. Conclusion

In this paper we presented a new hybrid solution that improves clustering quality, detects outlier data points and eliminates Empty Clustering problem significantly. Our model has a runtime disadvantage with respect to original k-means, which is linear. To test our model we used two data sets, one of which is very crowded and difficult to cluster when $k >> 1$. We compared our model with original k-means and several other improved clustering solutions. The experiments show that it outperforms other models and shows significant clustering quality gain.

Acknowledgment

This work is supported with a grant from the Ministry of Science, Technology and Industry of Turkey (Grant#STZ.0367.2013-2).

References

[1] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, no. 14. California, USA, 1967, pp. 281–297.

[2] S. G. Kolliopoulos and S. Rao, "A nearly linear-time approximation scheme for the euclidean k-median problem," in *Algorithms-ESA'99*. Springer, 1999, pp. 378–389.

[3] S. Arora, P. Raghavan, and S. Rao, "Approximation schemes for euclidean k-medians and related problems," in *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. ACM, 1998, pp. 106–113.

[4] P. K. Agarwal and C. M. Procopiuc, "Exact and approximation algorithms for clustering," *Algorithmica*, vol. 33, no. 2, pp. 201–226, 2002.

[5] J. A. Hartigan and M. A. Wong, "Algorithm as 136: A k-means clustering algorithm," *Applied statistics*, pp. 100–108, 1979.

[6] X. Li and Z. Fang, "Parallel clustering algorithms," *Parallel Computing*, vol. 11, no. 3, pp. 275–290, 1989.

[7] C. F. Olson, "Parallel algorithms for hierarchical clustering," *Parallel computing*, vol. 21, no. 8, pp. 1313–1325, 1995.

[8] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[9] R. Lämmel, "Google's mapreduce programming model—revisited," *Science of computer programming*, vol. 70, no. 1, pp. 1–30, 2008.

[10] T. White, *Hadoop: the definitive guide: the definitive guide*. " O'Reilly Media, Inc.", 2009.

[11] Y. Bu, B. Howe, M. Balazinska, and M. D. Ernst, "Haloop: Efficient iterative data processing on large clusters," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 285–296, 2010.

[12] S. Owen, R. Anil, T. Dunning, and E. Friedman, *Mahout in action*. Manning Shelter Island, 2011.

[13] R. M. Esteves, R. Pais, and C. Rong, "K-means clustering in the cloud–a mahout test," in *Advanced Information Networking and Applications (WAINA), 2011 IEEE Workshops of International Conference on*. IEEE, 2011, pp. 514–519.

[14] R. M. Esteves and C. Rong, "Using mahout for clustering wikipedia's latest articles: a comparison between k-means and fuzzy c-means in the cloud," in *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*. IEEE, 2011, pp. 565–569.

[15] Y. Tan and Y. Zhu, "Fireworks algorithm for optimization," in *Advances in Swarm Intelligence*. Springer, 2010, pp. 355–364.

[16] S. Zheng, A. Janecek, and Y. Tan, "Enhanced fireworks algorithm," in *Evolutionary Computation (CEC), 2013 IEEE Congress on*. IEEE, 2013, pp. 2069–2077.

[17] H. Fanaee-T and J. Gama, "Event labeling combining ensemble detectors and background knowledge," *Progress in Artificial Intelligence*, vol. 2, no. 2-3, pp. 113–127, 2014.

[18] P. S. Bradley and U. M. Fayyad, "Refining initial points for k-means clustering." in *ICML*, vol. 98. Citeseer, 1998, pp. 91–99.

[19] P. Bradley, K. Bennett, and A. Demiriz, "Constrained k-means clustering," *Microsoft Research, Redmond*, pp. 1–8, 2000.

[20] Y. Lu, S. Lu, F. Fotouhi, Y. Deng, and S. J. Brown, "Fgka: A fast genetic k-means clustering algorithm," in *Proceedings of the 2004 ACM symposium on Applied computing*. ACM, 2004, pp. 622–623.

[21] W. Zhao, H. Ma, and Q. He, "Parallel k-means clustering based on mapreduce," in *Cloud Computing*. Springer, 2009, pp. 674–679.