International Journal of Innovation, Management and Technology, Vol. 2, No. 1, February, 2011
ISSN: 2010-0248

Enhanced K-Means Clustering Algorithm Using Red Black Tree and Min-Heap

Rajeev Kumar, Rajeshwar Puran and Joydip Dhar

***Abstract*—Fast and high quality clustering is one of the most important tasks in the modern era of information processing wherein people rely heavily on search engines such as Google, Yahoo, and Bing etc. With the huge amount of available data and with an aim to creating better quality clusters, scores of algorithms having quality-complexity trade-offs have been proposed. However, the k-means algorithm proposed during late 1970's still enjoys a respectable position in the list of clustering algorithms. It is considered to be one of the most fundamental algorithms of data mining. It is basically an iterative algorithm. In each iteration, it requires finding the distance between each data object and centroid of each cluster. Considering the hugeness of modern databases, this task in itself snowballs into a tedious task. In this paper, we are proposing an improved version of k-means algorithm which offers to provide a remedy of the aforesaid problem. This algorithm employs two data structures viz. red-black tree and min-heap. These data structures are readily available in the modern programming languages. While red black tree is available in the form of *map* in C++ and *TreeMap* in Java, min-heap is available in the form of *priority queue* in the C++ standard template library. Thus implementation of our algorithm is as simple as that of the traditional algorithm. We have carried out extensive experiments. The results so obtained establish the superiority of our version of k-means algorithm over the traditional one.**

***Index Terms*—clustering; k-means algorithm; min-heap; red-black tree.**

I. Introduction

Clustering is one of the most fundamental tasks of data mining. It is the task of segregation of objects into groups of similar or nearly similar objects. The objects within each group should exhibit a relatively higher degree of similarity while the similarity among objects belonging to different clusters should be as small as possible [1].

Given a set of objects, we define clustering as a technique to group similar objects together without the prior knowledge of group definition. Thus, we are interested in finding k smaller subsets of objects such that objects in the same set are more similar to each other while objects in different sets are more dissimilar.

Data Representation by fewer clusters does lose some of the minute details. It, however, simplifies the whole process.

From the point of view of machine learning, clusters correspond to hidden patterns. Clustering is considered to be unsupervised learning. Clustering enjoys a vital position in data mining applications. The data mining tasks where clustering plays a crucial role include data exploration, information retrieval, text mining, applications pertaining to spatial database, web mining, marketing, medical diagnostics, CRM, computational biology etc.

k-means algorithm [2] is one of the most heavily used algorithms for clustering. It is a classic and basic algorithm for clustering. It is a partitioning algorithm [3]. It basically opts for an iterative process. For most practical purposes, it proves to be fast enough to generate good clustering solution. However, due to heavy numerical computation required, the efficiency of k-means algorithm becomes a point of concern in case of large datasets. It starts to perform poorly for large datasets.

In past, researchers have made several attempts in order to improve the efficiency of k-means algorithm. In paper [8], the authors have proposed an ingenious way to improve the execution time of k-means algorithm. This algorithm, in particular is useful in solving the clustering problems in gene expression datasets. The authors, in paper [9], gave yet another improved version of k-means algorithm named k-means++ algorithm. They achieved the improvement in performance by augmenting k-means with a very simple, randomized seeding technique. This algorithm is found to be

$\Theta(\log k)$-competitive with the optimal clustering. In paper [10], the authors have tried to improve the performance by indigenously defining the initial k centroids. They noted that different initial centroids lead to different results differing heavily in the quality of clustering solutions obtained. In paper [11], the authors have attempted to solve the problem of local optimum in document clustering. They proposed a novel Harmony k-means Algorithm (HKA). This new algorithm deals with document clustering based on Harmony Search (HS) optimization method .It is proved by means of finite Markov chain theory that the HKA converges to the global optimum. In paper [12], the authors have proposed an incremental approach to clustering that dynamically adds one cluster center at a time through a deterministic global search procedure consisting of N (with N being the size of the data set) executions of the k-means algorithm from suitable initial positions. They also proposed modifications of the method to reduce the computational load without significantly affecting

Rajeev Kumar is with the Department of Information Technology, ABV-Indian Institute of Information Technology and Management, Gwalior (M.P.), INDIA. E-mail: rajeevkumariiitm@gmail.com

Rajeshwar Puran is with the Department of Information Technology, ABV- Indian Institute of Information Technology and Management, Gwalior (M.P.), INDIA. E-mail: puran.iiitm@gmail.com.

Joydip Dhar is with the Department of Applied Sciences, ABV- Indian Institute of Information Technology and Management Gwalior (M.P.), INDIA.

E-mail: jdhar@iiitm.ac.in.

IACSIT
International Association of Computer Science and Information Technology
WWW.IACSIT.ORG

solution quality. In paper [13], the authors presented yet another approach to improve the speed of the k-means algorithm. Their algorithm basically organizes all the patterns in a k-d tree structure such that one can find all the patterns which are closest to a given prototype efficiently. All the prototypes are considered to be potential candidates for the closest prototype at the root level. However, for the children of the root node, the candidate set may be pruned by using simple geometrical constraints. This approach can be applied recursively until the size of the candidate set is one for each node. In paper [14], the authors have presented an efficient implementation of K-means algorithm which is based on storing the data in a kd-tree [15]. For each node of the tree, they maintained a candidate tree. The candidates for each node are pruned, as they are propagated to the node's children

In this paper, we intend to present an improved version of k-means algorithm. Although this algorithm does not differ with the traditional k-means algorithm [2] in terms of the quality of the clustering solution produced, it does outperform the traditional k-means algorithm in terms of running time. Thus, it enhances the speed of clustering and improves the time complexity of the traditional k-means algorithm.

The rest of this paper is organized as follows. The section 2 and section 3 give overviews of red-black tree and min-heap respectively. Section 4 discusses k-means clustering algorithm. Section 5 describes the problem we are working on. In section 6, we have presented our algorithm. Section 7 speaks about the implementation details. Section 8 analyses the results obtained. Section 9 concludes and discusses the possible future work.

II. Red-Black Tree

Red-black tree is a self-balancing binary search tree with one extra bit of storage per node [4]. This data structure was introduced by Rudolf Bayer in 1972. Although a complex data structure, it has a one of the best worst case running time for dynamic set operations. It takes O(logn) time for search, insertion and deletion , where n is the total number of elements in the tree. Each node of this tree has a color either red or black associated with it. A red-black tree, like all other binary search trees, allows in-order traversal, Left-Root-Right, of their elements.

In addition to the ordinary requirements imposed on the binary search trees [17], a red-black tree has the following additional requirements associated with it [4].

1) Every node is either red or black.
2) The root is black.
3) Every leaf (NIL) is black.
4) If a node is red, then both its children are black.
5) For each node, all paths form the node to descendent leaves contains the same number of black nodes.

These constraints instigate an important characteristic of red-black trees that the longest path from the root to any leaf is at most two times as long as the shortest path from the root to any other leaf in that tree, thus rendering the tree roughly balanced. If there are n nodes in the tree, the height of the tree is at most 2log(n+1). Since operations such as insertion, deletion and searching etc. require a time proportional to the height of the tree, this upper bound on the height allows the red-black trees to be extremely efficient in terms of worst-case complexity, unlike ordinary binary search trees [17] where the worst case complexity is O(n).

The following picture depicts a typical red-black tree.

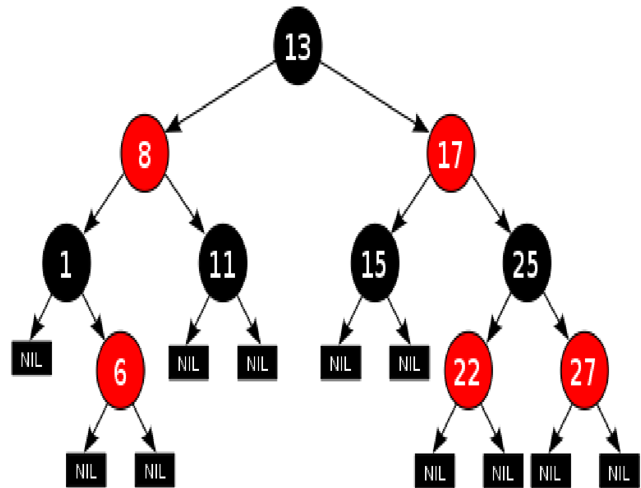

Figure 1. Red-Black Tree

Red-black tree is readily available in the form of *map* in the C++ standard template library [6] and *TreeMap* in Java [7]. Thus they are quite easy to implement.

III. Min- Heaps

The min-heap data structure [5] is an array object which can be visualized as an almost complete binary tree. The following figure depicts a typical min-heap.

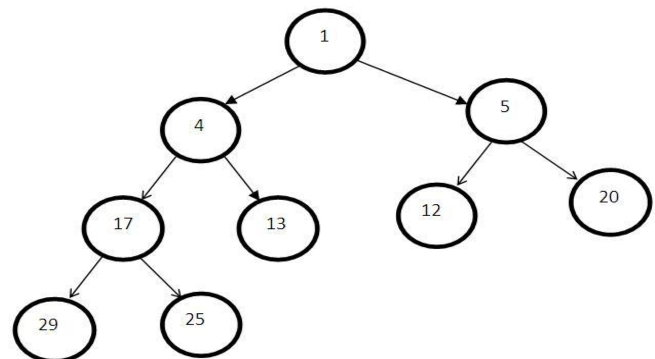

Figure 2. Min-Heap Tree

Each node of the tree stores a value corresponding to an element of the array. The tree is filled on all levels except possibly the lowest level where it is filled from the left up to a point where the values exhaust. An array A corresponding to a min-heap is an object with two attributes: length [A] which is the number of elements in the array and heap-size [A], the number of elements in the heap sorted within array A. That is, although A[1…., length [A]] may contain valid numbers, no element after A[heap-size[A]] , where heap- size[A] ≤ length [A] , is an element of the min-heap.

The property which makes min-heap a valuable data structure is that if B is a child node of A then key [A] ≤ key [B]. Thus, the smallest element of the array is always found to be in the root node.

The operations generally performed with a min-heap are

1) find-min – popping out the minimum element of the min-heap in $\Theta(1)$ time.
2) delete-min: removing the root node in $\Theta(\log n)$ time.
3) decrease-key: updating a key in $\Theta(\log n)$ time.
4) insert: adding a new key in $\Theta(\log n)$ time.
5) merge : joining two heaps to form a valid new heap containing all the elements of both in $\Theta(n)$ time.

Min-heaps are readily available in the form of *priority*_queue in the C++ standard template library [6]. However, sadly they are neither available in Java nor in C#. Thus, people trying to implement the algorithm proposed in this paper in Java or C# will have to implement their own version of min-heap.

IV. The K-Means Clustering Algorithm

The K-means algorithm [2] is a classical example of a clustering algorithm. It is one of the most widely used clustering algorithms in data mining. It is a simple, non-supervised learning algorithm. It is basically a partitioning algorithm. The basic aim is to divide the given n objects into k groups through an iterative process using certain similarity measures.

This algorithm basically works in two phases. In the first phase, k objects out of the given n objects are chosen. These objects are declared to the centroids of the initial clusters. Now, each data object is bound to its nearest cluster. The distance between a data object and a cluster is usually determined using the Euclidean distance.

$$d(x_i, y_i) = \left[\sum_{i=1}^{n} (x_i - y_i)^2\right]^{1/2} \qquad (1)$$

When each data object is assigned to some cluster, the first phase is over. Now the second phase starts. This phase consists of multiple iterations. In each iteration, new centroids of the clusters are calculated. The distance between each object and each cluster is recalculated as the clusters have changed. Now the objects are assigned to the cluster which is nearest to it. Iterations continue until a stage reaches when no more movement of data objects between the clusters take place. This is considered to be the end of the K-means algorithm.

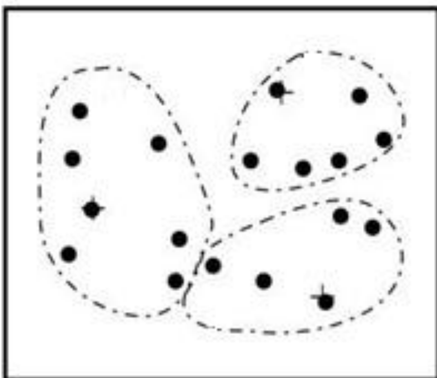

Figure 3. Working of k-means clustering algorithm

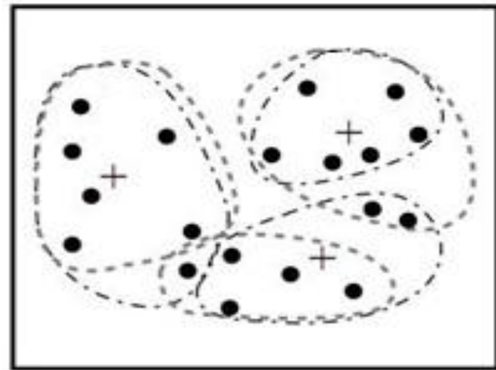

Figure 4. Working of k-means clustering algorithm

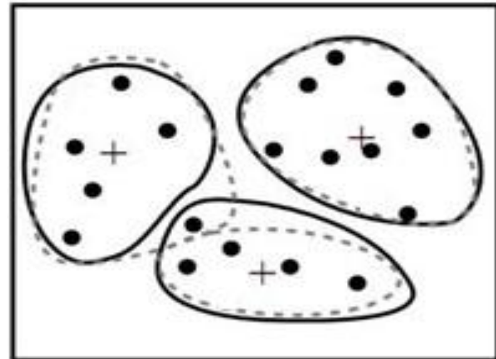

Figure 5. Working of k-means clustering algorithm

V. Problem Description

The k-means algorithm relies on calculation of distance between each data object and each cluster during each iteration. This task in itself can cause the algorithm run out of time in case of large datasets. Suppose, we need to partition 100000 objects into 100 clusters. Suppose the algorithm runs for 100 iterations. Thus the algorithm will have the calculate distance for a massive 100000 $\times$ 100 $\times$ 100 i.e. 1000000000 times.

If one carefully analyses the working of the k-means algorithm, one can deduce that there is no need to repeatedly calculate the distances between each data object and each cluster. Suppose in an iteration, only one data object moved from one cluster to another cluster, all other k-2 clusters being unaltered. One can easily understand that there is no need of calculation of distances between these k-2 clusters and data objects in the next iteration. However, the k- means algorithm still makes these unnecessary calculations. This is a huge destruction of time. In this paper, we propose an improved version of k-means algorithm in which we have astutely eliminated the need to recalculate the distances between data objects and clusters. For this purpose, we have used red-black tree [4] and min-heap [5] in our algorithm.

VI. Proposed Methodology

Our algorithm basically aims at reducing the computational overhead arising out of unnecessary calculation of distances between data objects and clusters in each iteration. We, first of all, choose k data objects to serve as centroids of k initial clusters. Now we calculate the Euclidean distance of each data object from these centroids.

IACSIT
International Association of Computer Science and Information Technology
WWW.IACSIT.ORG

We assign each data object to its nearest cluster based on the Euclidean distance just calculated. We initialize an empty red black tree. Now we insert, into this tree, labels of the objects as keys and a min-heap corresponding to each key as corresponding values. The min-heap in turn contains pairs of labels of clusters and distances of their centroids from the data object (key) as its values. Now if in an iteration, an object moves from one cluster to another cluster, we recalculate the centroids of these two clusters. Now we calculate the new distances between these two clusters and data objects. We, now, replace the old distances saved in the min-heaps with these new distances. We continue this process. It is noteworthy that we calculate the new distances corresponding to only those clusters which have altered due to movement of data objects. In the next iteration, we pop out the minimum element of each min-heap corresponding to each object put in the red-black tree as key. This popped out element is a pair of a cluster label and distance of its centroid from the object. Now the cluster corresponding to this class label will act as the new cluster for the object. Thus no recalculation of distances between the objects and clusters is required. Suppose a run of k-means algorithm consists of only one iteration. Suppose, this iteration in turn consists of only one movement of a single object. Our algorithm in this case, will calculate new distances corresponding to only those two clusters which have altered due to movement of data objects. However the traditional version of the K-means algorithm will calculate the distances of each object with each cluster. Thus, our version of K-means algorithm provides huge advantage in terms of time over the traditional K-means algorithm. Our algorithm ends in the same way as the traditional K-means algorithm i.e. when no object moved from one cluster to other cluster in an iteration.

A simple question can be raised why we used a red-black tree. We could have used a simple array. What advantage does a red-black tree provide over the array? The answer lies in the structure of the red-black tree. A red-black tree, as discussed earlier, consists of pairs of key and value. The key can either be a numerical value or a string value or both. It also does not need to be in any particular order. One key can be “India” while other key can be “train”. In other words a key can be anything. However in an array, only numerical values can act as keys. Also they have to be in order i.e 0,1,2,3,4,5,6…… Now an object label can be anything. Thus an array would prove to be highly ineffective in this case, while a red-black tree will serve the purpose with full strength.

A. *Pseudocode of the Proposed Algorithm*

Input:
k : the number of clusters required.
n : number of given data objects
D : a data set containing objects.
Output: A set of k clusters

Step1: Choose K random objects from given n data objects.

Step2: Assign each data object to its nearest cluster based on the Euclidean distance.

Step3: Initialize an empty red black tree.

Step4: Fill the tree with object labels as key and min-heaps as value.

Step5: Fill the min-heaps with the pairs of cluster labels and distance between the cluster and the key.

Step6: Repeat step7 to step 13 until no object moved between clusters.

Step7: Repeat steps8 to steps10 for each object.

Step8: Pop the topmost element i.e. the minimum element of its corresponding min heap. If the cluster label contained in this element is the same as the present cluster label of the object, do nothing. Otherwise move the object into the cluster corresponding to the cluster label obtained.

Step9: Calculate new centroids of the two clusters which have suffered alteration i.e. the original and the new cluster of the object just moved.

Step10: Calculate the distances of each object from these two clusters centroids and replace the old ones with these just calculated distances.

Step11: Do step 12 and step 13 for each object.

Step12: Pop out the minimum element of each min-heap corresponding to the object put in the red-black tree as key. This popped out element is a pair of a cluster label and distance of its centroid from the object.

Step13: Check the cluster corresponding to this class label. If this cluster is the same as the original cluster of the object, do nothing. Otherwise, Move the object to the new cluster.

VII. Experimental Details

In order to testify our algorithm, we have carried out sophisticated experiments wherein we have compared the working of our algorithm the traditional k-means algorithm .We have coded our version of k-means and the traditional one in Java programming language. We have chosen this language owing to its unique feature of portability. The codes prepared by us, hence, can be run on any operating system, be it windows XP, Vista, Fedora, Ubuntu or Macintosh. We have used a machine possessing 1 GB main memory and a 1.83 GHz dual core processor with windows XP service pack 2 as the operating system.

A. *Input Dataset*

In our experiments, we have used real datasets downloaded from [16]. These datasets are described in the following table.

TABLE 1. Dataset

Dataset	Number of attributes	Number of instances
Abalone	8	4177
Annealing	38	798
Dermatology	33	366
Mechanical Analysis	8	209

VIII. Results And Analysis

We plot the graphs between the traditional k-means

clustering algorithm and our improved k-means clustering algorithm for each dataset. They are depicted in figure 6-9.

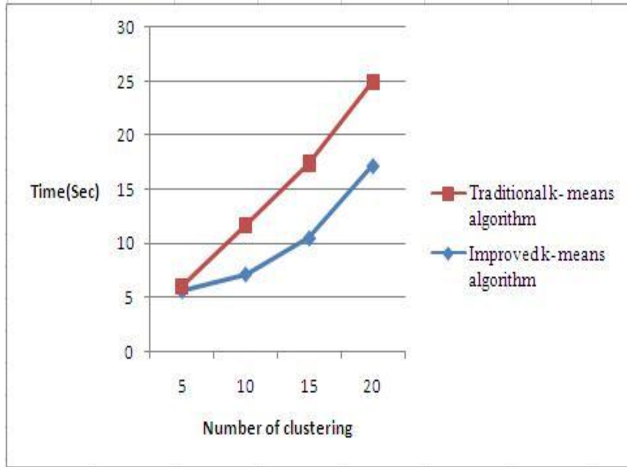

Figure 6. Performance comparison of Traditional k-means algorithm and improved k- means algorithm for Abalone dataset

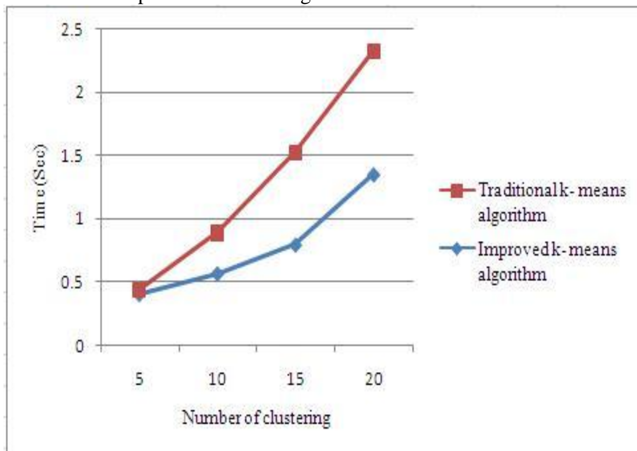

Figure 7. Performance comparison of Traditional k-means algorithm and improved k- means algorithm for Annealing dataset

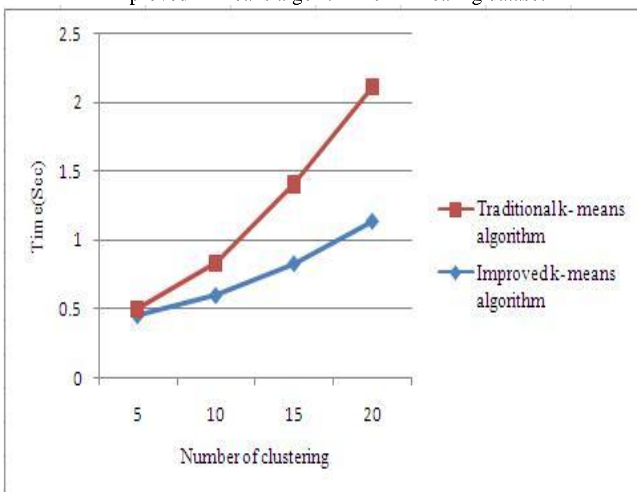

Figure 8. Performance comparison of Traditional k-means algorithm and Improved k- means algorithm for Dermatology dataset

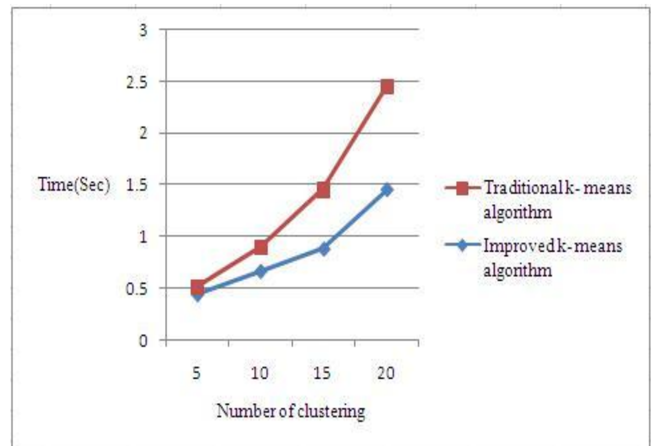

Figure 9. Performance comparison of Traditional k-means algorithm and improved k- means algorithm for Mechanical Analysis dataset

One can easily conclude from the above graphs that our algorithm improves the time complexity of the k-means algorithm.

IX. Conclusions And Future Work

The repeated calculation of distances between each data object and each cluster renders the k-means algorithm as computationally demanding. In this paper, we have proposed an improved version of k-means which offers a remedy to the aforesaid problem. This algorithm employs red-black tree and min-heaps in its implementation. These data structures are readily available in programming languages. Thus the implementation of this algorithm is as easy as the normal k-means algorithm. We have performed sophisticated experiments wherein we have compared the performances of our version of k-means with the traditional version. We have used both synthetic dataset and real dataset. Our algorithm is found to be outperforming the traditional k-means in terms of running time.

Our algorithm saves the distances between data objects and clusters. It then dynamically changes them when required. However, the saving of the distances requires much space. Thus, although our algorithm is superior to the traditional k-means algorithm in terms of time complexity, it appears to be lagging behind in terms of space complexity. In future, research work may be oriented to sort out this drawback of our algorithm.

References

[1] Han, J., Kamber,M.: Data Mining Concepts and Techniques. Morgan Kaufmann Publishers, San Francisco (2006).

[2] J. Hartigan and M. Wong, “Algorithm AS136: A k-means clustering algorithm,” Applied Statistics, 1979, pp. 100-108.

[3] A. K Jain, M. N. Murty, and P. J. Flynn, “Data Clustering: A Review,” ACM Computing Survey, Vol. 31, No. 3, 1999, pp. 264-323.

[4] Bayer, R., McCreight, E. M.: Organization and maintenance of large ordered indexes. Acta Informatica, 1(3): pp. 173-189 (1972).

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms. MIT Press, Cambridge, MA, 2nd edition (2001).

[6] Schildt, H.: C++: The Complete Reference. 4th edition, McGraw-Hill, Berkeley (2003).

[7] Schildt, H.: Java: The Complete Reference. 7th edition, McGraw-Hill, Berkeley (2007).

[8] A .M.Bagirov, K.Mardaneh, Modified global k-means algorithm for clustering in gene expression datasets, in: WISB’06, Australian Computer Society, Inc., Darlinghurst, Australia, 2006, pp.23–28.

IACSIT
International Association of Computer Science and Information Technology
WWW.IACSIT.ORG

[9] D. Arthur and S. Vassilvitskii, “K-means++: the advantages of careful seeding,” In: ACM-SIAM symposium on discrete algorithms, 2007.

[10] Yuan F, Meng Z. H, Zhang H. X and Dong C. R, “A New Algorithm to Get the Initial Centroids,” Proc. of the 3rd International Conference on Machine Learning and Cybernetics, pp. 26–29, August 2004.

[11] M. Mahdavi and H. Abolhassani, “Harmony k -means algorithm for document clustering,” Data Mining and Knowledge Discovery 2009.

[12] A. Likas, N. Vlassis, and J. J. Verbeek. The global k-means clustering algorithm. Pattern Recognition, 36(2),2003.

[13] K. Alsabti, S. Ranka, V. Singh, An efficient K-means clustering algorithm, In 11th international parallel processing Symposium.

[14] T. Kanungo, D.M. Mount, N.S. Netanyahu, C. Piatko, R. Silverman, A.Y. Wu, An efficient k-means clustering algorithm: Analysis and implementation, IEEE Transaction on Pattern Analysis and Machine Intelligence, 24 (2002).

[15] J.L. Bentley, 'Multidimensional Binary Search Trees Used for Associative Searching, ºComm. ACM, vol. 18, pp. 509-517, 1975.

[16] UCI Repository of Machine Learning Databases, http://archive.ics.uci.edu/ml/datasets.html.

[17] Knuth, D. E.: Sorting and searching, volume 3 of The Art of Computer Programming, Addison-Wesley, Reading, MA (1973).