K*-Means: An Effective and Efficient K-means Clustering Algorithm

Jianpeng Qi, Yanwei Yu*, Lihong Wang,and Jinglei Liu
School of Computer and Control Engineering, Yantai University, Yantai, Shandong 264005, China
Email: jianpengqi@126.com, yuyanwei@ytu.edu.cn, wanglh_000@163.com, jinglei_liu@sina.com

K-means is a widely used clustering algorithm in field of data mining across different disciplines in the past fifty years. However, k-means heavily depends on the position of initial centers, and the chosen starting centers randomly may lead to poor quality of clustering. Motivated by this, this paper proposes an optimized k-means clustering method along with three optimization principles named k^*-means. First, we propose a hierarchical optimization principle initialized by k^* cluster centers ($k^* > k$) to reduce the risk of randomly seeds selection, and then utilize proposed top-n method to merge the nearest clusters associated with the shortest n edges in each round until the number of clusters reaches at k. Second, we propose a cluster pruning strategy for improving efficiency of k-means by omitting the farther clusters to shrink the adjustable searching space for each point in each iteration. Third, we implement an optimized update theory to optimize the k-means iteration updating, which leverages moved points updating instead of recalculating mean and SSE of cluster to minimize computation cost. Our comprehensive experimental studies, using 2 synthetic datasets and 4 real world datasets from the UCI Machine Learning Repository, demonstrate that our method outperforms state-of-the-art methods in both effectiveness and efficiency.

***Index Terms*—Data mining, Clustering, k-means, Top-n merging, Cluster pruning.**

I. Introduction

Clustering is to group data objects into different classes or clusters and is one of most task in data analysis, such as pattern discovery, pattern recognition, data summary and image processing [1]. Many branches are developed to enrich these fields, include partitional methods, hierarchical methods, density-based methods, etc. Partitional method is the simplest and most foundational version of cluster analysis, and many algorithms are proposed to accelerate its process like k-means[2], k-medoids[3], k-means++[4].

K-means is a most widely used and well studied method in data mining. Given a dataset $D = \{\boldsymbol{p_i} \mid i = 1 \dots n\}$, $\boldsymbol{p_i}$ in d-dimensional space $\Re^d$, k-means is to assign the set of points into k clusters with arbitrary selected k initial centers, so as to minimize sum of squared error (SSE) shown in formula (2), where $\|\boldsymbol{p_i} - \boldsymbol{m_j}\|$ is the distance from point $\boldsymbol{p_i}$ to cluster center $\boldsymbol{m_j}$, δ_{ij} is the cluster indicator variable with $\delta_{ij} = 1$ if $p_i \in C_j$ and 0 otherwise, and $\boldsymbol{m_j}$ is the mean of cluster C_j and be calculated by formula (1).

$$\boldsymbol{m_j} = \frac{\sum_{p_i \in C_j} \boldsymbol{p_i}}{|C_j|} \tag{1}$$

$$SSE = \sum_{j=1}^{k} \sum_{i=1}^{n} \delta_{ij} \|\boldsymbol{p_i} - \boldsymbol{m_j}\|^2 \tag{2}$$

Due to simple yet effective way, k-means is widely accepted and becomes a popular member for performing clustering across different disciplines over the past 50 years [5][6][7][8]. However k-means suffers from a serious limitation, random initial centers selection may lead to get trapped in poor local minimal, specially in a bad initialization. A most obvious observation is that k-means often divides a large cluster into serval small groups or merges small adjacent clusters into a larger one to get minimal SSE. See example shown in Fig.1, starting with a bad initialization, on the upper-right two near clusters are mistaken to merge into one cluster, and on the bottom right corner one cluster are divided into 2 subclusters.

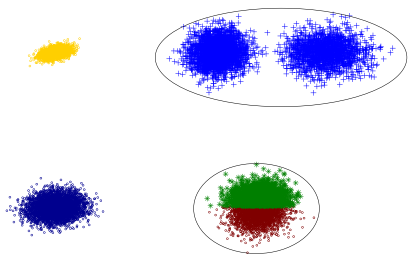


Figure 1: Example of clusters begin with a bad initialization

Many works attempt to resolve the sensitivity of initialization of k-means [1][9]. Arthur et al. [10] propose a careful seeds selection method, named k-means++, for alleviating the shortcoming of k-means. K-means++ chooses starting centers with specific probabilities. Specifically, they first associate point with a squared distance from the point to the closest center that already chosen called D^2 weighting, and then select initial centers with probability $\frac{D(x_j)^2}{\sum_{i=1}^{n} D(x_i)^2}$. However k-means++ needs to take k passes over the data to select initial centers, which limits its applicability to massive dataset. A recent study [11] proposes a MinMax k-means algorithm to tackle the sensitivity to initial centers by altering the k-means objective. MinMax k-means also starts from arbitrary centers but tries to minimize the maximum intra-cluster sum of squared error sse_{max} instead of total SSE. That is, MinMax k-means tries to minimize the largest intra-cluster variance to reduce the global SSE across a weighting factor with each cluster. However, MinMax does not obtain a better clusters directly, specially for unbalanced cluster distribution. It also still needs to further optimize the result by performing k-means beginning with MinMax k-means result. In addition, the weighting mechanism also leads to additional expensive computation.

Another category of methods attempt to improve the efficiency of k-means by decreasing computational complexity. Hamerly [12] accelerates k-means using distance bounds and the triangle inequality to avoid unnecessary distance computation. Bahmani et al. [4] present a scalable k-means++ to fast obtain initialization in parallel, which leverages parallel computational model to reduce the number of passes for large scale dataset.

In this paper, we propose a novel hierarchical k-means approach, k^*-means, to improve both quality and efficiency of clustering. We first start k-means with a larger input parameter k^* ($k^* > k$), and then merge the clusters associated with top-n nearest distances and further refine clusters by k-means, next repeat last process until the number of clusters reach at k. To improve k^*-means performance, we further deploy three optimization principles to minimize the computation cost. First, in top n merging process, we use the feature values ($mean$) of nearest clusters to fast generate feature information of the merged cluster instead of recomputing. Second,we propose a cluster pruning strategy by the distance between two cluster centers to shrink the adjusting searching space for each point in k-means iterations. And we also provide a lower bound of pruning distance along with theoretical analysis. Third, we formalize the optimized update principle based on the fact that the number of points moved across clusters decreases sharply with iteration progress. Proposed principle maintains feature values of each cluster just by the moved points to avoid to recompute the $mean$ from scratch. Furthermore, we also explore selecting bound of parameter k^* value by extending experiments.

II. Related Work

Initialization methods for k-means. Forgey et al. [13] first group the points into k clusters uniformly at random, and then choose the centroids of these clusters as initial centers. MacQueen [2] proposes two different initialization methods. The first method chooses the first k points in dataset as the starting centers. The second one randomly chooses the initial centers from dataset. Obviously, the first one is sensitive to the order of points, and the second one may selects outliers or points that are too close to each other, as above example in Fig.1. Bradley and Fayyad [3] propose an initialization method, which first randomly partitions the dataset into J groups, and then performs k-means with MacQueen's second initialization method in each group. The obtained centers are combined into superset that is then clustered by k-means J times. In final, members of the centers that have the least SSE are chosen as initial centers.

As previously mentioned, k-means++ introduces a probabilistic seeds selection method. It chooses the first center randomly and then selects other centers of using a probability of $\frac{D(x_j)^2}{\sum_{i=1}^{n} D(x_i)^2}$, where $D(\boldsymbol{x_j})$ is the shortest distance from the point $\boldsymbol{p_j}$ to the closest center that already chosen. Therefore, a point that is far away from all selected centers have a high probability to be chosen as a center. They claim the method yields an $O(\log k)$ approximation. However, when dimension of dataset is high, the complexity of the k-means++ would be much greater than $O(\log k)$ [14].

Su et al. [15] propose two hierarchical approaches, PCA-Part and Var-Part methods. PCA-Part iteratively obtains clusters based on PCA (Principal Component Analysis), which starts from an initial cluster containing all data points, and then chooses cluster repeatedly with greatest SSE and cuts it into two subsets using a hyperplane that passes through the cluster centroid and is orthogonal to the principle eigenvector of cluster covariance matrix. Var-Part is an approximation version of PCA-Pmart, which divides the selected cluster into two subclusters by assuming the covariance matrix is diagonal. Lu et al. [16] propose a hierarchical initialization method, which treats the clustering problem as a weighted clustering problem to find better initial cluster centers based on the hierarchical approach. Redmond et al. [17] present a method that incorporates kd-trees to perform density estimation of the data at various location and then chooses initial centers using maximin method from densely populated leaf buckets.

Recent study MinMax k-means[11] uses the objective of maximum sse_{max} instead of total SSE of all clusters, which aims to overcome the sensitivity to the random initialization. Because MinMax k-means tries to minimize the maximum intra-cluster variance, it expects to obtain balanced clusters with respect to their variance. Thus it is undesirable for dataset with unbalanced clusters. In extreme cases, when the empty or singleton cluster emerges during the process of iteration, it would leads to MinMax k-means restarts even fails to get clustering result sometimes. MinMax k-means also introduces weight mechanism into objective, which causes huge of computation costs.

Optimized variants of k-means on efficiency. Kanungo et. al [18] propose an efficient filtering k-means method by using kd-tree to store data points and then utilizing the index to filter or prune the candidate centers to reduce the computation. Hung et al.[19] propose a simple partitioning k-means method, which partitions the dataset into serval unit blocks (UB) and then locates the centroids of the UBs (CUB) using a simple calculation. All CUBs form a reduced dataset that is used to compute the final converged centroids instead of the original dataset. Hamerly[12] accelerates k-means using distance bounds and the triangle inequality to avoid unnecessary distance computations. Another group of methods try to improve efficiency of k-means from parallel or distributed perspective. Zhao et al. [20] propose a parallel k-means algorithm based on popular distributed platform MapReduce. However these algorithms also suffer from the serious random initialization limitation of k-means. To apply to large scale dataset, Bahmani et al. [4] present a parallel version of k-means++ in MapReduce model of computation, which focuses on the initialization stage of k-means to fast obtain the probabilistic initial centers in parallel by reducing number of passes.

III. The Algorithm

Since k-means chooses initial centers randomly, it is difficult to avoid choose outliers or points that are too close to each other. As shown in Fig.1, the situations of mistaken merging or dividing always occur once two seeds are chosen in a cluster, and the cluster is far from other points. The risk of random initialization increases significantly especially for unbalanced or skewed data distribution.

Let n_i denote number of points in cluster $C_i(i = 1, 2, \ldots, k)$ in k-means initialization. So the probability that seed fall into cluster C_j is $p = \frac{n_j}{\sum n_i}$. If cluster C_j has a few points, meaning that it has a little opportunity to obtain a seed, which further directly leads that there exist some clusters get more than one initial centers. Obviously, to avoid this situation happen, the direct thought is that how to improve the ratio to make small cluster get a initial center as far as possible. The straightforward way is to extend the value of k. Namely, we can choose more than k initial centers denoted k^*, to increase the probability $\frac{n_j}{\sum n_i} \cdot \frac{k^*}{k}$.

The simple solution easily relieves the sensitivity to the initialization, but it also means that we need some additional steps to decrease k^* to k to obtain final k clusters. In Section III-A we introduce our Top-n nearest clusters merging to implement the transformation with minimal cost.

A. Top-n Nearest Clusters Merging

In our k^*-means algorithm, we first obtain k^* clusters by performing k-means with randomly chosen $k^*(k^* > k)$ initial centers. To obtain k clusters in final, we propose top-n nearest clusters merging strategy. Before introduce top-n method, we first define the *edge* to describe the distance between two clusters.

Definition 1. ***(edge)*** *Given two cluster C_1, C_2, and their center c_1, c_2, the distance between C_1 and C_2 is defined as $d = |c_1 - c_2|$, we call the distance edge which connected c_1 and c_2, denoted $e_{1,2}$.*

By Def. 1, we only need to find top-n shortest edge to get top-n nearest clusters. We first use an ascending list structure to store the edges associated with corresponding clusters. Then the top-n shortest edges are selected and clusters associated with the edges are merged.

However, top-n merging does not always reduce n clusters but at most n clusters. For example, three edges $e_{i,j}$, $e_{i,k}$ and $e_{j,k}$ all shortlist top-3 edges, then the cluster C_i, C_j and C_k should be merged into one cluster at the same time, reducing 2 clusters. We still name the process top-n merging because we precisely merge n shortest edges. But we can not guarantee to merge n clusters. Instead the number of reduced cluster ranges from $\lceil\frac{\sqrt{1+8*n}-1}{2}\rceil$ to n. Therefore, we first need to determine the selection range of n value. If n is beyond the range, the number of clusters may be less than k after merging process. Here we deduce that n should be smaller than $\lceil k^* - k\rceil$ in each round.

Proof. Suppose the top n shortest edges involve m clusters. In extreme case of reducing minimum number of clusters, namely the shortest n edges (one edge corresponding to a combination of two clusters) are just the combinations of m clusters, then the number of involved clusters is the least, and all m clusters should be merged into one cluster. Hence we have $n = C_m^2 = \frac{m\cdot(m-1)}{2}$, then we get $m = \frac{1+\sqrt{1+8\cdot n}}{2}$. So the number of reduced clusters is $m - 1 = \frac{\sqrt{1+8\cdot n}-1}{2}$. Therefore, if the n edges just are a subset of combinations of m clusters, then the number of reduced clusters is $\lceil\frac{\sqrt{1+8*n}-1}{2}\rceil$. Consider the cases of reducing maximum number of clusters, the first case is that the n edges involves $2n$ clusters, namely there are no duplicated clusters on the n edges. Intuitively merging process would reduce n clusters in this cases. The second case is that the m clusters associated with n edges are merged into one cluster but each of the m clusters appears at most 2 times on the n edges. Thus we deduce that $m = 2n - (n - 1) = n + 1$. So the number of reduced clusters also is n in this case. Therefore, the number of clusters reduced by our top-n nearest clusters merging ranges from $\lceil\frac{\sqrt{1+8*n}-1}{2}\rceil$ to n, and further we obtain that $n \leq \lceil k^* - k\rceil$ to assure the number of clusters is not less than k after merging process. □

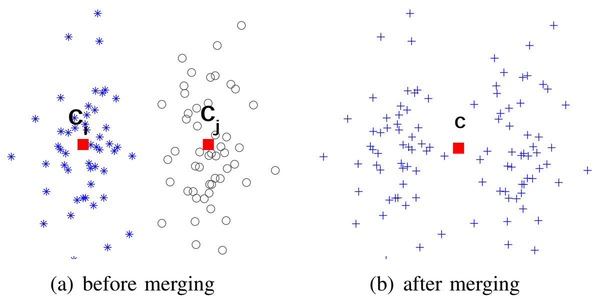


Figure 2: An example of cluster merging

As shown in Fig. 2, suppose clusters C_i and C_j belong to top-n nearest clusters, and C is the merged cluster from C_i and C_j. Therefore, we can get Lemma 1.

Lemma 1. *Given clusters C_i and C_j, m_i and m_j are means of C_i and C_j respectively, and sse_i, sse_j are sum of squared error of C_i and C_j respectively, if C is merged from C_i and C_j, then we have*

$$\boldsymbol{m_c} = \frac{\boldsymbol{m_i} \cdot |C_i| + \boldsymbol{m_j} \cdot |C_j|}{|C_i| + |C_j|} \tag{3}$$

where $\boldsymbol{m_c}$ is means of C.

Proof. Without loss of generality, let $C_i = \{\boldsymbol{p_u} \mid u = 1, 2, \ldots \mid C_i \mid\}$ and $C_j = \{\boldsymbol{q_v} \mid v = 1, 2, \cdots \mid C_j \mid\}$.
First, since $\boldsymbol{m_i}$ and $\boldsymbol{m_j}$ are means of C_i and C_j respectively, we have

$$\boldsymbol{m_i} = \frac{\sum_{u=1}^{|C_i|} \boldsymbol{p_u}}{|C_i|}; \ \boldsymbol{m_j} = \frac{\sum_{v=1}^{|C_j|} \boldsymbol{q_v}}{|C_j|}.$$

Then we get formula (4).

$$\sum_{u=1}^{|C_i|} \boldsymbol{p_u} = \boldsymbol{m_i} \cdot |C_i|; \ \sum_{v=1}^{|C_j|} \boldsymbol{q_v} = \boldsymbol{m_j} \cdot |C_j| \tag{4}$$

Note that $\boldsymbol{m_c}$ is the means of merged cluster C, we have

$$\begin{aligned} \boldsymbol{m_c} &= \frac{\sum_{\boldsymbol{o} \in C} \boldsymbol{o}}{|C|} = \frac{\sum_{u=1}^{|C_i|} \boldsymbol{p_u} + \sum_{v=1}^{|C_j|} \boldsymbol{q_v}}{|C_i| + |C_j|} \\ &= \frac{\boldsymbol{m_i} \cdot |C_i| + \boldsymbol{m_j} \cdot |C_j|}{|C_i| + |C_j|} \end{aligned}$$

And we are done. □

Lemma 1 implies that the feature values of merged cluster can be computed directly according to those of previous sub-clusters.

B. Cluster Pruning Strategy

To reduce the computational costs, we now introduce optimization principle, regarding to minimization of distance comparison, termed cluster pruning strategy.

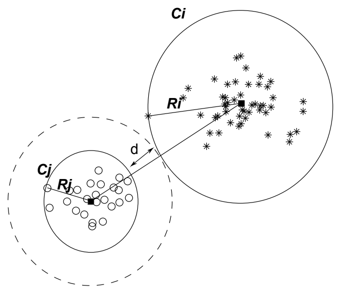


Figure 3: Pruning distance between two clusters

In k-means iterations, each point needs to be examined if it is closer to its center than any other centers. Hence, each point has a larger searching space. For example, if there are k clusters, each point needs to calculate and compare distance $(k-1)$ times in each iteration. However, most of these computation are redundant. As exactly shown in Fig. 4, only very few points would be moved among clusters in adjusting iteration. We also observe that if two clusters are distant from each other, the points in the two cluster are no need to compare with each other in adjusting steps.

Next, we introduce our pruning optimization method for each point at cluster level based on Def. 2 and Lemma 2.

Definition 2. ***(Radius)*** *Given a cluster C_i, the radius of the cluster C_i is the maximum distance from the center of C_i to its points, denoted R_i.*

Lemma 2. *Given two clusters C_i, C_j, and their radius R_i, R_j, if $e_{i,j} \geq R_i + R_j + |R_i - R_j|$, then there is no point in $C_i(C_j)$ that is moved into $C_j(C_i)$ in adjusting iteration.*

Proof. Suppose there are two clusters, C_i and C_j, their radius are R_j and R_i, respectively, as shown in Fig. 3. Without loss of generality, we assume $R_i > R_j$, let d denote the gap between two clusters, thus $d = e_{i,j} - R_i - R_j$. Obviously all points in C_j are closer to $\boldsymbol{m_j}$ (the center of C_j) than $\boldsymbol{m_i}$ (the center of C_i).

Now, we consider the farthest point in C_i, denoted $\boldsymbol{p}$. By the definition of radius, the distance from $\boldsymbol{p}$ to the center of C_i is R_i, and the distance from $\boldsymbol{p}$ to the center of C_j must not be less than $d + R_j$, namely $distance(\boldsymbol{p}, \boldsymbol{m_j}) \geq d + R_j$.

If $distance(\boldsymbol{p}, \boldsymbol{m_j}) \geq R_i$, then all points in C_i is not moved into C_j certainly. Hence, if $d + R_j \geq R_i$, then $distance(\boldsymbol{p}, \boldsymbol{m_j}) \geq R_i$ must hold.
Since

$$\begin{aligned} d + R_j \geq R_i &\Rightarrow e_{i,j} - R_i - R_j + R_j \geq R_i \\ &\Rightarrow e_{i,j} \geq R_i + R_j + |R_i - R_j| \end{aligned}$$

Therefore, $e_{i,j} \geq R_i + R_j + |R_i - R_j|$ guarantees that there is no point that be adjusted between C_i and C_j. □

Lemma 2 guides our pruning rule for k^*-means iteration. For each cluster C_i, if distances between C_i and other clusters, namely $edges$ associated with C_i, is greater than the pruning distance of $R_i+R_j+|R_i-R_j|$, then we eliminate these further clusters from searching space in k^*-means iteration.

Note that pruning strategy needs to maintain additional $radius$ for each cluster. For this, we use a simple way to maintain this indicator, namely, we first obtain the $radius$ when compute $mean$ at the beginning of k^*-means, and then we update the indicator $radius'$(initial with $radius$) in time. Which means $radius'$ can be updated when point moving. And assign $radius'$ to $radius$ at the end of each iteration.

C. Optimized Update Principle

From extending experiments on UCI data (see details in section V-A), we observe that number of moved points declines sharply during the k-means iteration processes. Fig. 4 shows two samples on two real world datasets. As k-means iteration goes on, the points in clusters trend towards stability, thus the number of moved points also goes towards zero. In particular, the number of points moved across clusters is very low after the forth iteration. And only average 3% of points are adjusted from fifth iteration to end. Motivated by this observation or k-means characteristic, we propose an optimized update principle, which updates feature values in each iteration using moved points instead of re-computation from all of points.

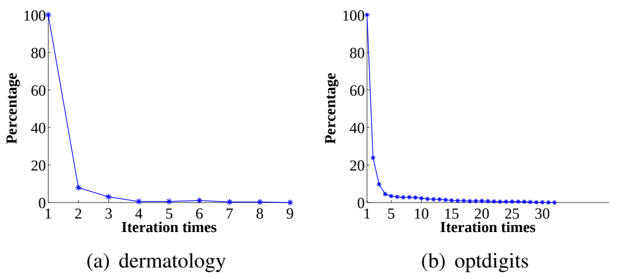


Figure 4: Number of moved points in k-means iterations

To understand the optimized update principle, we first state Lemma 3 and Lemma 4.

Lemma 3. *Given a cluster C_i, and its center $\boldsymbol{m_i}$, if a point $\boldsymbol{p}$ is added into C_i, then we get the new center $\boldsymbol{m_i'}$ as formula (5), where $|C_i|$ is number of points in C_i.*

$$\boldsymbol{m_i'} = \boldsymbol{m_i} + \frac{\boldsymbol{p} - \boldsymbol{m_i}}{|C_i| + 1} \tag{5}$$

Lemma 4. *Given a cluster C_i, and its center $\boldsymbol{m_i}$, if a point $\boldsymbol{p}$ is moved out from C_i, then we get the new center $\boldsymbol{m_i'}$ as formula (6), where $|C_i|$ is number of points.*

$$\boldsymbol{m_i'} = \boldsymbol{m_i} + \frac{\boldsymbol{m_i} - \boldsymbol{p}}{|C_i| - 1} \tag{6}$$

Lemma 3 and lemma 4 depict updating method of $mean$ when a point is moved in or out cluster, respectively. Therefore, the feature values of a cluster can be maintained incrementally via the moved points during k-means iteration, rather than re-computed from all points in the cluster.

Armed with Lemma 3 and Lemma 4, we propose the optimized update method to reduce computation costs. In each iteration, we maintain the copies of feature values incrementally according to moved points. Optimized update method obviously saves much computation resource when number of moved points is very low. However, at beginning iterations of k-means, specially, the first round k-means, moved points amount is relatively larger, and the optimized update method may cost more time than completed method. Therefore, we use a threshold θ, the ratio of moved points in each iteration, to determine whether start to perform optimized update method in the first round k-means. On experiential, from Fig. 4, we set $\theta = 0.05$.

IV. K*-MEANS ALGORITHM

Next, we present our extending k^*-means algorithm based on our proposed three optimization strategies. The detail pseudo-code of k^*-means is shown in Algorithm 1.

In k^*-means, we also use the random initialization method to choose k^* starting centers, and first assign all points into k^* clusters. Then we get feature values of $mean$ for each cluster, as shown in lines 1-3. Next, k^*-means performs hierarchical clustering along with k-means adjusting iteration in lines 4-27 and nearest clusters associated with top-n distances merging in line 28-30. Lines 6-10 describe the proposed cluster pruning strategy. We use collections CS to store the neighbor clusters of specific cluster which after prune remote clusters by Lemma 2. Then, we only need to verify the adjustable clusters in search space of CS for each point in C_i. Once percentage of moved points is lower than given θ in first round k-means optimized update principle will be started and $radius'$ will be updated during this process(lines 11- 19). At the end of each iteration, each cluster mean m and it's $radius$ is replaced by $m', radius'$ directly. Lines 28-30 show top-n nearest clusters merging which reduce number of clusters from $|C|$ to $|C| - [\lceil \frac{\sqrt{1+8*n}-1}{2} \rceil, n]$. Therefor, parameter n is not fixed, but ranges from given n to 1. For each round refining of k^*-means, we use a decrease strategy to determine value of n, and a top-1 may be performed at final round to make number of clusters reach at k.

V. EXPERIMENTS

A. Experimental Setup and Methodologies

All algorithms are implemented by Java, and all experiments are performed on platform equipped with 2.4G Intel Core i3 CPU, 4G memory, and Windows 7 operate system.

Datasets. We use 6 data sets including 2 synthetic datasets and 4 real world datasets from the UCI Machine Learning Repository [21] to evaluate the effectiveness and efficiency of our k^*-means. As described in Table I, Smalldata and Bigdata are generated by our data generator using mixture Gauss model for covering different situations, such as unbalanced clusters, clusters with arbitrary shapes.

Algorithm 1 K^*-means Algorithm

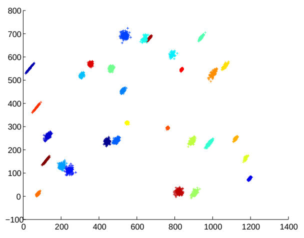
```
Input: the number of initial centers $K^*$; the number of
    clusters $K$; data set $D$; parameters of Top-n method $N$.
Output: $K$ clusters.
1: arbitrary choose $K^*$ initial centers;
2: distribute all points in $D$ to it's nearest cluster;
3: update $mean$ of each cluster;
4: while not convergence do
5:     for each $C_i \in C$ do
6:         for each $C_j \in C$ and $i \neq j$ do
   ▷ //Cluster pruning strategy
7:             if $e_{i,j} < R_i + R_j + |R_i - R_j|$ then
8:                 add $C_j$ to cluster set $CS$;
9:             end if
10:        end for
11:        for each point $p \in C_i$ do
12:            get distance $d$ from $p$ to the nearest center in $CS$;
13:            if $d < |p - m_i|$ then
14:                move p into the nearest cluster $C_j$;
15:                if $|p - m_j| > C_j.radius'$ then
16:                    update $C_j.radius'$;
17:                end if
   ▷ //Optimized update principle
18:                update $m'$ of $C_i$, $C_j$ by lemma 3, 4;
19:            end if
20:            if $p$ not move out && $|p - m_i| > C_i.r$ then
21:                set $C_i.r = |p - m_i|$;
22:            end if
23:        end for
24:        set $radius' = C_i.r$ and $C_i.r = 0$;
25:    end for
26:    use $m', radius'$ instead of $m, radius$ in cluster $C_i, C_j$;
27: end while
28: if number of clusters $> K$ then
29:     Top-n($C$,$N$); goto line 4;
30: end if
```

Table I: Descriptions of synthetic and real world datasets

No.	Dataset	# of points	# of attributes	# of classes
D1	Smalldata	3851	2	30
D2	Bigdata	120000	2	50
D3	Ecoli	307	7	4
D4	Optdigits	5620	64	10
D5	Dermatology	366	34	6
D6	Folio1	300	500	15

As shown in Fig.5(a), D1 includes 30 clusters that be composed by 3851 points. In this dataset, each cluster contains less than 200 points with Gauss distribution and number of clusters is balanced. We denote the small dataset as **Smalldata**.

The forth dataset shown in Fig.5(b) comprises massive data points of 120 thousands, including 50 clusters with arbitrary shapes. Moreover, there is no obvious gaps for some clusters, and the number of points in clusters also is unbalance. We


(a) D1

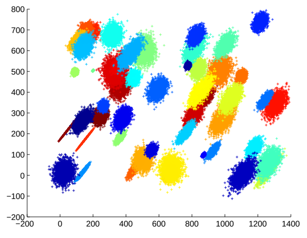

(b) D2

Figure 5: Demonstrations of 2 synthetic datasets

denote the big dataset with arbitrary shape clusters as **Bigdata**.

Ecoli (UCI) [21] is calculated from the amino acid sequences, which includes 8 classes that be comprised of 336 proteins. The data has 8 attributes(includes one label), and the number of classes is different, thus it is an unbalanced dataset.

Optdigits (UCI) [21] is optical recognition of handwritten digits by extracting normalized bitmaps of handwritten digits from a preprinted form. Each preprinted form is represented as a 8×8 matrix, where every element is an integer. The dataset includes 5620 instances in 64 dimensions space.

Dermatology (UCI) [21] contains 366 patients with 34 attributes, which is partitioned into 6 types of Erythemato-Squamous disease. The dataset is also unbalanced.

Folio1 (UCI) [21] are leaves images taken from different plants in the farm of the University of Mauritius and nearby locations. We first extract the SIFT descriptors from these images, and then use the bag of 500 visual words to represent the data. Folio1 contains 300 leaves from 15 different types of plants, balanced.

Experimental Methodologies. For effectiveness evaluation of our algorithm, we use three metrics of average SSE, NMI (Normalized Mutual Information) and SC (Silhouette Coefficient) [22] to estimate the quality of clustering. SSE utilized is described as formula (2) in Introduction section, and we take the average SSE that divides total SSE by the number of points. Formula (7) depicts computing method of NMI, where w_k is the obtained cluster k, c_j is the labeled class j, N is the size of dataset, and the $\mathcal{H}_M$ and $\mathcal{H}_C$ represent the entropy of clusters and classes.

$$NMI = \frac{\sum_k \sum_j \frac{|w_k \cap c_j|}{N} log \frac{N|w_k \cap c_j|}{|w_k| \cdot |c_j|}}{[\mathcal{H}_M + \mathcal{H}_C]/2} \tag{7}$$

$\mathcal{H}$ is defined as formula (8).

$$\mathcal{H}_M = -\sum_k \frac{|w_k|}{N} log \frac{|w_k|}{N} \tag{8}$$

Therefore, NMI score ranges from 0 to 1.0, more higher value indicates more better quality of clustering.

As shown in formula (7), we need labelled class information for computing NMI score. But it is difficult to obtain data with annotations in many real world applications. So we also adopt another widely used metrics SC(Silhouette Coefficient), which describes both similarity among points in intra-cluster and differences among clusters without cluster labels. Let $a(i)$ be the average distance of point i to all other points within the same cluster. Let $b(i)$ be the minimum average distance of point i to the rest clusters. Formula (9) shows the SC for point i.

$$s(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}} \tag{9}$$

The average Silhouette Coefficient of all points $\frac{\sum_{i=1}^{N} s(i)}{N}$ is regard as the SC value. Thus SC $\in (-1, 1)$, similar with NMI, higher SC value indicates the obtained clusters are more distinguished clearly.

For efficiency evaluation, we measure the common metrics of CPU consumption. All experiments are conducted 10 times on each dataset, and we take the average CPU time. Moreover, we discuss the selection of k^* value by analysing the impact of k^* in Section V-D.

Alternative algorithm. We compare our algorithm against k-means, k-means++ and MinMax k-means. We also apply k-means++ initialization method into our k^*-means, named k^*-means++ algorithm. That is, k^*-means++ chooses k^* initial centers using D^2 weighting probability instead of random selection.

B. *Effectiveness Evaluation*

First, we evaluate the clustering quality of our proposed k^*-means on 6 datasets when k^* is fixed to $2k$, $n = 2$ and $\theta = 0.05$. Note that we set memory lever $\beta = 0.3$ for MinMax k-means. Table II, III and IV show experimental results. Asterisk ($*$), dagger (†) and double dagger (‡) superscripts denote that k^*-means or k^*-means++ has a statistically significant difference to k-means, k-means++ and MinMax k-means respectively, according to the standard deviation. A line above (below) these symbols stands for a higher (lower) average. To distinguish the best result, the top-2 best results on each dataset are shown in bold.

NMI. The results of NMI on 6 datasets is shown in Table II, our k^*-means is superior to k-means, k-means++, MinMax k-means on all datasets. In particular, NMI scores on 2 synthetic data demonstrate that k^*-means has better ability of handling various scenarios compared to other tree algorithms. Even on real world datasets, k^*-means also achieves robustly higher NMI score. This clearly displays that k^* initial centers can better cover all of clusters and multiple-round merging improves the quality of clustering.

For k-means++, its probability initialization process improves performance to a certain extend compared to randomly starting of k-means. MinMax k-means aims to get lower maximum sse of clusters but may not yields better clusters due to the randomly initial centers. K^* (more) centers improve the probability of obtaining best local optima, and our multi-round refining further approaches the optimum gradually. As expected, k^*-means++ also outperforms k-means, k-means++ and MinMax k-means on most of datasets and achieves the best quality on 5 datasets.

SC. As shown in Table III. Again, k^*-means outperforms k-means, k-means++ and MinMax k-means in term of SC value. In particular, our algorithm exhibits better SC value than k-means and MinMax k-means on all datasets, and better than k-means++ on all datasets but Optdigits. However, SC values of k^*-means are very close to those of k-means++ on this

Table II: Clustering quality on NMI

Dataset	k-means	k-means++	minmax k-means	k^*-means	k^*-means++
Smalldata	0.933± 0.01	0.967±0.01	0.932±0.02	**0.991±0.00**$\overline{*}\overline{\dagger}\overline{\ddagger}$	**0.996±0.00**$\overline{*}\overline{\dagger}\overline{\ddagger}$
Bigdata	0.889± 0.01	0.905±0.01	0.904±0.00	**0.913±0.00**$\overline{*}\overline{\dagger}\overline{\ddagger}$	**0.915±0.00**$\overline{*}\overline{\dagger}\overline{\ddagger}$
Ecoli	0.616± 0.02	0.617±0.03	0.550±0.01	**0.635±0.03**$\overline{*}\overline{\dagger}\overline{\ddagger}$	**0.617±0.01**$\overline{*}\overline{\dagger}\overline{\ddagger}$
Optdigits	0.718± 0.02	**0.724±0.02**	0.708±0.00	**0.719±0.01**$\overline{*}\underline{\dagger}\overline{\ddagger}$	0.711±0.01$\underline{*}\underline{\dagger}\overline{\ddagger}$
Dermatology	0.861± 0.05	0.828±0.04	0.775±0.01	**0.863±0.04**$\overline{*}\overline{\dagger}\overline{\ddagger}$	**0.878±0.02**$\overline{*}\overline{\dagger}\overline{\ddagger}$
Folio1	0.532±0.02	0.534±0.01	0.540±0.00	**0.550±0.02**$\overline{*}\overline{\dagger}\overline{\ddagger}$	**0.555±0.02**$\overline{*}\overline{\dagger}\overline{\ddagger}$

Table III: Clustering quality on SC

Dataset	k-means	k-means++	minmax k-means	k^*-means	k^*-means++
Smalldata	0.741±0.02	0.813±0.02	0.748±0.05	**0.841±0.01**$\overline{*}\overline{\dagger}\overline{\ddagger}$	**0.849±0.01**$\overline{*}\overline{\dagger}\overline{\ddagger}$
Bigdata	0.554±0.03	0.559±0.02	0.556±0.01	**0.610±0.01**$\overline{*}\overline{\dagger}\overline{\ddagger}$	**0.577±0.00**$\overline{*}\overline{\dagger}\overline{\ddagger}$
Ecoli	0.362±0.02	0.365±0.02	0.343±0.01	**0.371±0.01**$\overline{*}\overline{\dagger}\overline{\ddagger}$	**0.378±0.00**$\overline{*}\overline{\dagger}\overline{\ddagger}$
Optdigits	0.175±0.02	**0.186±0.00**	0.172±0.00	**0.183±0.01**$\overline{*}\underline{\dagger}\overline{\ddagger}$	0.177±0.01$\overline{*}\underline{\dagger}\overline{\ddagger}$
Dermatology	0.254±0.03	0.228±0.04	0.204±0.01	**0.269±0.01**$\overline{*}\overline{\dagger}\overline{\ddagger}$	**0.286±0.02**$\overline{*}\overline{\dagger}\overline{\ddagger}$
Folio1	0.388±0.04	0.417±0.02	0.407±0.01	**0.426±0.02**$\overline{*}\overline{\dagger}\overline{\ddagger}$	**0.445±0.01**$\overline{*}\overline{\dagger}\overline{\ddagger}$

Table IV: Clustering quality on SSE

Dataset	k-means	k-means++	minmax k-means	k^*-means	k^*-means++
Smalldata	1.08E+03±383.47	2.77E+02±112.35	1.21E+03±878.40	**1.44E+02±35.14**$\underline{*}\underline{\dagger}\underline{\ddagger}$	**1.03E+02±2.21**$\underline{*}\underline{\dagger}\underline{\ddagger}$
Bigdata	2.14E+03±576.16	9.73E+02±114.08	**9.02E+02±26.58**	1.22E+03±65.59 $\underline{*}\overline{\dagger}\overline{\ddagger}$	**8.26E+02±10.22**$\underline{*}\underline{\dagger}\underline{\ddagger}$
Ecoli	5.06E-02±0.00	**4.99E-02±0.00**	5.30E-02±0.00	5.07E-02±0.00$\overline{*}\overline{\dagger}\underline{\ddagger}$	**4.96E-02±0.00**$\underline{*}\underline{\dagger}\underline{\ddagger}$
Optdigits	6.69E+02±18.54	**6.58E+02±3.55**	6.66E+02±0.11	**6.64E+02±11.70**$\underline{*}\overline{\dagger}\underline{\ddagger}$	6.73E+02±14.13$\overline{*}\overline{\dagger}\overline{\ddagger}$
Dermatology	9.79E+00±0.25	**9.65E+00±0.32**	1.01E+01±0.01	9.67E+00±0.19$\underline{*}\overline{\dagger}\underline{\ddagger}$	**9.30E+00±0.18** $\underline{*}\underline{\dagger}\underline{\ddagger}$
Folio1	2.69E-03±0.00	**2.34E-03±0.00**	2.50E-03±0.00	2.37E-03±0.00$\underline{*}\overline{\dagger}\underline{\ddagger}$	**2.21E-03±0.00**$\underline{*}\underline{\dagger}\underline{\ddagger}$

two datasets. K^*-means++ also improves clustering quality in 83% of tested cases. This is because carefully selection of k^* initial centers further optimizes probability range of best local optima by avoiding centers that are too close to each other compared to random initialization.

SSE. Table IV shows the SSE metrics of algorithms on 6 datasets. K^*-means obtains better(smaller) SSE on 2 datasets and k-means++ reaches better results on 5 datasets. This is because carefully seeding more easily guides k-means++ to converge to a better local minima of SSE than random initialization. MinMax k-means produces a better maximum sse of cluster but not a better SSE value. However, our proposed k^*-means++ exhibits best SSE values on all datasets but optdigits.

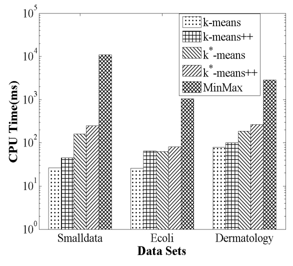
In summary, the above comprehensive experiments confirm the effectiveness of our proposed k^*-means in attaining clusters of better quality. Furthermore, our k^*-means integrated with k-means++ initialization robustly achieves best quality clusters in most real applications.

C. Efficiency Evaluation

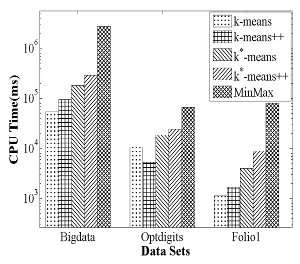
Next, we evaluate the efficiency of our k^*-means compared against k-means, k-means++ and MinMax k-means on 10 datasets when $k^* = 2k$, $n = 2$ and $\theta = 0.05$. Fig.6 shows the results of running time on 2 groups of dataset. We can see that k-means costs lowest CPU time and MinMax consumes much more time compared to other algorithms. However, our k^*-means costs same order of magnitudes CPU time compared to k-means and k-means++. This is because k^*-means utilizes merging optimization, cluster pruning strategy and optimized update theory to reduce the computation cost significantly, while MinMax k-means adds a maximization step that updates objective with weighting mechanism in each iteration, which increases much more computation costs. Specially when dataset is unbalanced, MinMax k-means is more inefficient. In particular, k^*-means and k^*-means++ just cost 2 or 3 folds CPU time compared to k-means on most datasets.

D. Determining of parameter k^*

Finally, we conduct experiments to explore the selection of k^* value on 4 datasets (2 synthetic datasets and 2 real world datasets). We measure the NMI score and SC value by varying k^* from $1k$ to $4.5k$ with $0.5k$ increment when k is fixed to the number of classes, $n = 2$ and $\theta = 0.05$. The results is shown in Fig.7, (a)(b) are NMI scores on the 4 datasets and (c)(d) are SC values on the 4 datasets, respectively. We observe that NMI



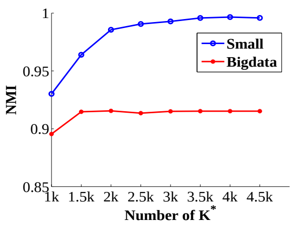
(a) Group 1(log scale)



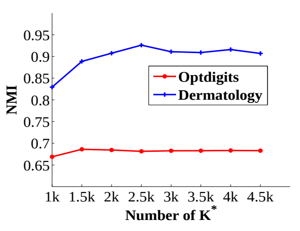
(b) Group 2(log scale)

Figure 6: CPU time of algorithms on 6 datasets

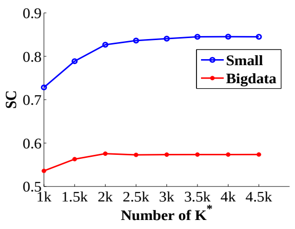
scores of our proposed k^*-means increases as k^* increases, but when $k^* \geq 2k$ the NMI score is no longer growth and remains stable. Similarly, SC value also holds stable when k^* reaches at $2k$ or $2.5k$ on tested datasets. This may be because range of initial seeds could cover all of cluster when k^* enlarges to $2k$. And it dose not further improve the quality of clustering if continue to increase k^*. Namely we can not obtain an better result if k^* is greater than $2k$, that is exactly why we set $k^* = 2k$ in above experiments.



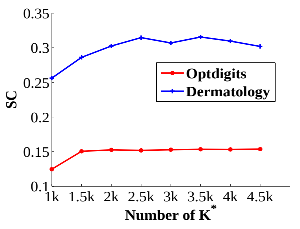
(a) NMI on Group 1



(b) NMI on Group 2



(c) SC on Group 1



(d) SC on Group 2

Figure 7: Impact of k^* on NMI and SC

VI. Conclusions

In this work we propose a novel optimized hierarchical clustering method incorporated with three optimization principles, namely ”top-n nearest clusters merging”, ”cluster pruning strategy” and ”optimized update principle” to achieve both effective and efficient clustering robustly. K^* initial centers effectively improves the probability of obtaining best local optima, and multi-round top-n nearest clusters merging approaches the optimal result gradually. The top-n and update principle optimizations update feature values of clusters by previous clusters or moved objects instead of re-computation from scratch. And the pruning strategy reduces significantly the adjusting searching space for each points in k-means iteration. Our experimental evaluation with 2 synthetic datasets and 4 real datasets demonstrates that the proposed algorithm exhibits better performance than the state-of-the-art in terms of cluster quality and CPU time in different scenarios. Furthermore, Our solution framework beginning with k-means++ initialization can further improve the performance, suggesting an alternative optimal solution.

An interesting direction for future work is to leverage modern distributed multi-core cluster of machines for further improving the scalability of our algorithm.

VII. Acknowledgments

We thank the anonymous reviewers for their many useful comments.

References

[1] Hassan A Kingravi M Emre Celebi and Patricio A Vela. A comparative study of efficient initialization methods for the k-means clustering algorithm. *Expert Systems with Applications*, 40(1):200–210, 2013.
[2] J. Macqueen. Some methods for classification and analysis of multivariate observations. In *In 5th Berkeley Symp. Math. Statist. Prob*, pages 281–297, 1967.
[3] Paul S Bradley and Usama M Fayyad. Refining initial points for k-means clustering. In *ICML*, volume 98, pages 91–99, 1998.
[4] Bahman Bahmani, Benjamin Moseley, Andrea Vattani, Ravi Kumar, and Sergei Vassilvitskii. Scalable k-means++. *VLDB*, 5(7):622–633, 2012.
[5] Anil K Jain. Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, 31(8):651–666, 2010.
[6] Zhipeng Cai, Maysam Heydari, and Guohui Lin. Clustering binary oligonucleotide fingerprint vectors for dna clone classification analysis. *Journal of Combinatorial Optimization*, 9(2):199–211, 2005.
[7] Zhipeng Cai, Randy Goebel, Mohammad R Salavatipour, Yi Shi, Lizhe Xu, and Guohui Lin. Selecting genes with dissimilar discrimination strength for sample class prediction. In *APBC*, pages 81–90, 2007.
[8] Zhipeng Cai, Lizhe Xu, Yi Shi, Mohammad R Salavatipour, Randy Goebel, and Guohui Lin. Using gene clustering to identify discriminatory genes with higher classification accuracy. In *BioInformatics and BioEngineering, 2006. BIBE 2006. Sixth IEEE Symposium on*, pages 235–242. IEEE, 2006.
[9] José Manuel Pena, Jose Antonio Lozano, and Pedro Larranaga. An empirical comparison of four initialization methods for the k-means algorithm. *Pattern Recognition Letters*, 20(10):1027–1040, 1999.
[10] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 1027–1035, 2007.
[11] Grigorios Tzortzis and Aristidis Likas. The minmax k-means clustering algorithm. *Pattern Recognition*, 47(7):2505–2516, 2014.
[12] Greg Hamerly. Making k-means even faster. In *SDM*, pages 130–140, 2010.
[13] E. W. Forgy. Cluster analysis of multivariate data: Efficiency vs. interpretability of classification. *Biometrics*, 21(3):768–769, 1965.
[14] Tobias Brunsch and Heiko Röglin. A bad instance for k-means++. In *Theory and Applications of Models of Computation*, pages 344–352. 2011.
[15] Ting Su and Jennifer G Dy. In search of deterministic methods for initializing k-means and gaussian mixture clustering. *Intelligent Data Analysis*, 11(4):319–338, 2007.
[16] J. F. Lu, J. B. Tanga, Z. M. Tanga, and J. Y. Yanga. Hierarchical initialization approach for k-means clustering. *Pattern Recognition Letters*, 29(6):787C795, 2008.
[17] Stephen J. Redmond and Conor Heneghan. A method for initialising the k-means clustering algorithm using kd-trees. *Pattern Recognition Letters*, 28(8):965–973, 2007.
[18] Tapas Kanungo, David M Mount, Nathan S Netanyahu, Christine D Piatko, Ruth Silverman, and Angela Y Wu. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):881–892, 2002.
[19] Ming-Chuan Hung, Jungpin Wu, Jin-Hua Chang, and Don-Lin Yang. An efficient k-means clustering algorithm using simple partitioning. *Journal of Information Science and Engineering*, 21(6):1157–1177, 2005.
[20] Weizhong Zhao, Huifang Ma, and Qing He. Parallel k-means clustering based on mapreduce. In *Proceedings of International Conference on Cloud Computing*, pages 674–679, 2009.
[21] University of California. Machine learning repository. http://archive.ics.uci.edu/ml/.

[22] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987.