

# On the Lower Bound of Local Optimums in K-Means Algorithm

Zhenjie Zhang      Bing Tian Dai      Anthony K.H. Tung  
School of Computing  
National University of Singapore  
{zhenjie,daibingt,atung}@comp.nus.edu.sg

## Abstract

*The  $k$ -means algorithm is a popular clustering method used in many different fields of computer science, such as data mining, machine learning and information retrieval. However, the  $k$ -means algorithm is very likely to converge to some local optimum which is much worse than the desired global optimal solution. To overcome this problem, current  $k$ -means algorithm and its variants usually run many times with different initial centers to avoid being trapped in local optimums that are of unacceptable quality. In this paper, we propose an efficient method to compute a lower bound on the cost of the local optimum from the current center set. After every  $k$ -means iteration,  $k$ -means algorithm can halt the procedure if the lower bound of the cost at the future local optimum is worse than the best solution that has already been computed so far. Although such a lower bound computation incurs some extra time consumption in the iterations, extensive experiments on both synthetic and real data sets show that this method can greatly prune the unnecessary iterations and improve the efficiency of the algorithm in most of the data sets, especially with high dimensionality and large  $k$ .*

## 1 Introduction

The  $k$ -means algorithm is a popular clustering method used in many different fields of computer science, such as data mining, machine learning and information retrieval. Given a data set  $P$ , the  $k$ -means algorithm, also known as Lloyd's algorithm [13], tries to find  $k$  centers in the space minimizing the *cost*, which is the sum of the squared Euclidean distance from every point in  $P$  to its nearest center. At the beginning of the algorithm,  $k$  random centers are chosen from the original data set. Then the algorithm keeps invoking  $k$ -means iterations. Every  $k$ -means iteration consists of two operations. First, every point in the data set is assigned to the nearest center. Second, points are divided into  $k$  groups according to the nearest center in the previ-

ous step and the geometric centers of all groups form a new set of centers. This procedure continues until the centers do not move any more. The  $k$ -means algorithm is accepted and used in many different applications because of its simplicity and efficiency.

However, there are two main problems for  $k$ -means algorithm. First, in each iteration, much computation time is spent on assigning every point in the data set to its new nearest center. Second, the algorithm is easy to be trapped in some local optimum, which can be much worse than the global optimum. For the first problem, there have been several works on accelerating the nearest center search procedure based on triangle inequality [6] or indexing structures [10, 15], which can work well in low dimensional space. Compared with the first problem, the second problem has not been well addressed yet. Although there are some studies on the choices of initial centers [3, 14] to avoid those local optimums, these methods do not show too much advantage over the simple random selection in the data set.

To address the second problem as mentioned above, we propose a novel computation method on the lower bound of local optimums in the multi-procedure  $k$ -means algorithm, where every procedure is a running of  $k$ -means algorithm with a unique initial center set. With the current center set of a procedure, the method proposed in this paper is able to find out a lower bound on the cost of the local optimum, where the algorithm will converge to. As solution with smaller cost is favored, the current procedure can be terminated if such a lower bound is greater than the minimum cost achieved by other procedures. The computation time on the further iterations to reach those worse local optimums can thus be saved. Here, we simply assume but not limit to the assumption that all procedures are run one by one on a single machine.

In order to lower bound the cost at the local optimum, our algorithm works in two steps. In the first step, it tries to find a maximal region inside which the local optimum will definitely fall in. Despite that the general maximal region for a center set is hard to compute, we propose a special type of maximal region which is looser but easy to calculate. In the

---

**Algorithm 1 Original  $k$ -means algorithm** (data set  $P$ ,  $k$ )

---

- 1: Randomly choose  $k$  points as the center set  $M$
  - 2: **while**  $M$  is not still **do**
  - 3:    $M = \text{SimpleIterate}(P, M)$  //See Algorithm 2 //
  - 4: **end while**
  - 5: Return  $M$
- 

second step, it calculates the lower bound on local optimum by estimating how the movements of the centers can affect the overall cost. Although the searching of such maximal region will result in extra computation time consumption, the benefit from early pruning by the lower bound can significantly speed up the whole  $k$ -means algorithm. This effect can be verified by our extensive experiments on both synthetic and real data sets.

The rest of the paper is organized as follows. Section 2 defines the problem and discusses some intuition. Section 3 proposes the concept of maximal region and derive the lower bound based on a special type of maximal region. Section 4 presents the new iteration algorithm and the new accelerated  $k$ -means algorithm by exploiting the lower bound on local optimums. Section 5 gives some experimental results, and Section 6 reviews some related work. Finally, we conclude this paper in Section 7.

## 2 Problem and Intuition

In this paper, we put our focus on the  $k$ -means clustering algorithm in Euclidean space  $\mathcal{R}^d$ . We use  $p = (p[1], p[2], \dots, p[d])$  to represent a point  $p$  in this space. The distance defined in Euclidean space  $D(p, q)$  can thus be calculated by  $D(p, q) = \sqrt{\sum_{i=1}^d (p[i] - q[i])^2} = \|p - q\|$ .

Given a center set  $M = \{m_1, m_2, \dots, m_k\}$ , we define the cost of the center set with respect to data set  $P$  as the sum of squared Euclidean distance of points in  $P$  to their corresponding nearest center within  $M$ , i.e.,  $C(M, P) = \sum_{p \in P} \min_i D^2(p, m_i)$ . Without ambiguity, we simplify  $C(\{q\}, P)$  as  $C(q, P)$ .

Given a data set  $P$ , we use  $c(P) = \frac{1}{|P|} \sum_{p \in P} p$  to denote the geometric center of  $P$ . It is well known that  $c(P)$  is the optimal solution of 1-mean on data set  $P$ , which has the following property [9, 12].

**Lemma 1** *Given a data set  $P$  and an arbitrary point  $q$  in the same space, we have  $C(q, P) = C(c(P), P) + |P|D^2(q, c(P))$ .*

### 2.1 $K$ -Means Algorithm

In Algorithm 1, we present the detail of the original  $k$ -means algorithm. At the beginning, the original center set

---

**Algorithm 2 SimpleIterate** (data set  $P$ ,  $M$ )

---

- 1: **for** every point  $p$  in  $P$  **do**
  - 2:   Assign  $p$  to the closest center in  $M$
  - 3: **end for**
  - 4: **for** every  $m_i$  in  $M$  **do**
  - 5:   Use the geometric center of all points assigned to  $m_i$  to replace  $m_i$
  - 6: **end for**
  - 7: Return  $M$
- 

$M$  is constructed by randomly choosing  $k$  points from the original data set  $P$ . Then, the algorithm keeps improving the cost of the solution by invoking the **SimpleIterate** (Algorithm 2) algorithm. This procedure stops when the centers do not move any more, i.e., no point changes its assignment in the last iteration.

In **SimpleIterate**, it first assigns the points in  $P$  to the closest center in  $M$  (line 1-3), then updates the centers by replacing the old ones with the new geometric center of the clusters (line 4-6). It is easy to verify that such an iteration can always decrease the overall cost before the convergence of the  $k$ -means algorithm.

In this paper, we are trying to improve the efficiency of multi-procedure styled  $k$ -means algorithm, where several procedures with different initial centers are run and the best solution is returned as the final result. These procedures are assumed to run one by one on a single machine.

### 2.2 Local Optimums in $K$ -Means Algorithm

Just like the other geometrical optimization problems,  $k$ -means algorithm can not guarantee to converge to global optimum every time. The algorithm is very likely to stop at some local optimum much worse than the global optimum. To clearly illustrate the severity of the local optimum problem in  $k$ -means algorithm, we give some statistics on some real data set here.

We run 20 procedures of  $k$ -means algorithm on KDD99 data set with parameter  $k = 4$ . The results show that 80% of random initial centers lead  $k$ -means algorithm to some local optimum with cost larger than 110,000, while 10% of procedures end with cost smaller than 100,000. Most of the computation time is wasted on the useless iterations if the initial centers are not well chosen!

To discover such bad initial centers as early as possible, we focus on deriving a lower bound on the local optimums achievable in the future iterations. In Fig 1, we give a running example of the accelerated  $k$ -means algorithm with lower bound computation. The curve above shows the costs of the center set after every iteration, while the curve below shows the corresponding lower bounds on the local opti-

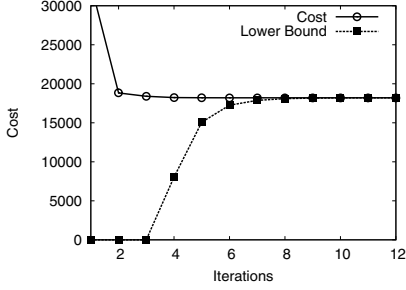


Figure 1: Lower Bound Example

imum computed after every iteration. Here the lower bound at any iteration is guaranteed to be smaller than the final convergence cost. While the original  $k$ -means algorithm can stop only after totally converging to local optimum, the lower bound estimation can be used to terminate the procedure earlier. For example, if the cost of current best solution is below 100,000, there is no need to further iterate after the 5th iteration since it is impossible to find a better solution. The termination in such situation can save more than half of the computation time in this procedure since there are another 7 iterations before convergence.

### 3 Maximal Region and Lower Bound

There are two steps in the computation of the lower bound on local optimums. In the first step, we find a maximal region within which the centers can move in the future iterations. In the second step, we bound the cost of any center set in the maximal region.

#### 3.1 Maximum Region of Local Optimum

To facilitate the analysis on the movements of the centers in  $k$ -means algorithm, we define a  $kd$ -dimensional solution space  $S$  for all center sets of size  $k$ . Given a center set  $M = \{m_1, m_2, \dots, m_k\}$ , we can find a corresponding point in  $S$ ,  $\mu_M = (m_1[1], \dots, m_1[d], \dots, m_k[1], \dots, m_k[d])$ . Without ambiguity, we directly use  $M$  to denote both a center set and its position  $\mu_M$  in  $S$ .

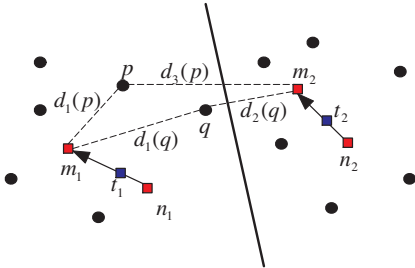


Figure 2: Center movement in one iteration

In Fig 2, we show an example of one 2-means iteration in a 2-dimensional space. With  $N = \{n_1, n_2\}$  as the previous center set, points on the left side of the solid line are assigned to center  $n_1$  while the others are assigned to center  $n_2$ . The geometric centers of the two sets are  $m_1$  and  $m_2$ , and thus  $M = \{m_1, m_2\}$  replaces  $N$  as the new center set. Therefore, we say the center set moves from  $(n_1[1], n_1[2], n_2[1], n_2[2])$  to  $(m_1[1], m_1[2], m_2[1], m_2[2])$  in the solution space  $S$ . We call a center set  $T$  *Intermediate Center Set* between  $N$  and  $M$ , if every center  $t_i$  in  $T$  lies on the line segment joining  $m_i \in M$  and  $n_i \in N$ . Obviously,  $T$  is also on the line segment between  $M$  and  $N$  in solution space  $S$ .  $T = \{t_1, t_2\}$  is such an intermediate center set in Fig 2.

In the following, we denote the neighborhood of center  $m_i$  by  $NH(m_i, M, P)$ , which contains points in  $P$  nearer to  $m_i$  than any other center in  $M$ .

**Lemma 2** Assume  $N$  is the center sets before a  $k$ -means iteration, and  $M$  is the center set after the  $k$ -means iteration. If  $T$  is an intermediate center set between  $M$  and  $N$ , we have  $C(N, P) \geq C(T, P)$ .

*Proof:* Since  $m_i = c(NH(n_i, N, P))$ , by Lemma 1, we have

$$\begin{aligned} C(N, P) &= \sum C(n_i, NH(n_i, N, P)) \\ &= \sum C(m_i, NH(n_i, N, P)) \\ &\quad + \sum |NH(n_i, N, P)| D^2(n_i, m_i) \end{aligned} \quad (1)$$

On the other hand, the cost of the center set  $T$  is

$$\begin{aligned} C(T, P) &\leq \sum C(t_i, NH(n_i, N, P)) \\ &= \sum C(m_i, NH(n_i, N, P)) \\ &\quad + \sum |NH(n_i, N, P)| D^2(t_i, m_i) \end{aligned} \quad (2)$$

Combining (1) and (2), we have  $C(N, P) \geq C(T, P)$  since  $D(n_i, m_i) \geq D(t_i, m_i)$ .  $\square$

The above lemma shows that when  $k$ -means algorithm moves the centers after one iteration, any solution on the line segments between them must be a better solution than the previous one. This further implies that if the current center set are surrounded, in the solution space  $S$ , by a group of center sets with higher costs, then the centers must converge at some local optimum in the “basin”, which is proven rigorously in the following theorem.

**Theorem 1** Assume  $M$  is a  $k$ -means center set in  $S$  and is covered by a closed region  $R \subset S$ . If every center set  $T$  on the boundary of  $R$  has  $C(T, P) > C(M, P)$ , the  $k$ -means algorithm with  $M$  as initial centers must converge at some solution  $M' \in R$ .

*Proof:* If from current center set  $M$ ,  $k$ -means algorithm can converge at some solution out of  $R$ , there must be one iteration from solution  $M_1$  to solution  $M_2$ , where  $M_1$  is in  $R$  but  $M_2$  is not. There must be an intermediate center set  $M_3$  on the boundary of  $R$  between  $M_1$  and  $M_2$ . Since  $C(M_1, D) \leq C(M, D)$  by the property of iterations, and  $C(M_3, D) > C(M, D)$  by assumption, we have  $C(M_3, D) > C(M_1, D)$  which contradicts lemma 2.  $\square$

Based on the above theorem, we know that if we can find a region  $R$  containing the current center set  $M$  in the solution space  $S$  with costs larger than  $C(M, P)$  on the boundary, the movements of the centers are constrained in such a region  $R$ . We call such a region *Maximum Region* of the  $k$ -means local optimum.

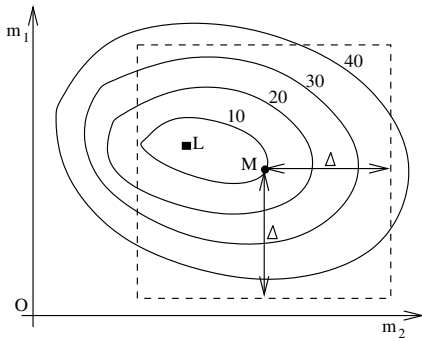


Figure 3: Maximal Region Example

In Fig 3, we show an example of the solution space  $S$  for a 1-dimension 2-means clustering. In the figure, the vertical axis represents the position of the first center and the horizontal axis represents the position of the second center, so any point in the figure is a 2-point center set on 1-dimension space. We use contour lines to present the costs of the center sets in the solution space, and use the square point to represent the local optimum in the space. Assume the circle point  $M$  is the current center set after the last iteration. Since  $M$  is on the contour line of cost 10, we have  $C(M, D) = 10$ . By Theorem 1, any contour line of cost larger than 10 must form a boundary of a maximal region for  $M$ , such as the contour line of cost 20 or 30. The dashed rectangle, whose center is  $M$ , also forms a maximal region, since any center set on the boundary of the rectangle must have cost larger than 10. It is straightforward to verify that the local optimum  $L$  is enclosed by any of such maximal regions.

### 3.2 A Special Type of Maximum Region

As is shown in the example of Fig 3, maximum regions can be of complex shape, which makes it impossible to exhaustively search for such regions in the  $kd$ -dimensional space  $S$ . We therefore propose a special type of maximum regions, which is easier to manipulate. Given a center set  $M$ , we define  $R(M, \Delta) = \{M' =$

$\{m'_1, m'_2, \dots, m'_k\} \mid \forall i, 1 \leq i \leq k, \|m'_i - m_i\| \leq \Delta\}$ . That is, for any  $M' \in R(M, \Delta)$ , there is no center  $m'_i \in M'$  is away from  $m_i \in M$  by  $\Delta$  distance. Therefore, the boundary of  $R(M, \Delta)$  is  $\partial R(M, \Delta) = \cup B(m_i, M, \Delta)$ , where  $B(m_i, M, \Delta) = \{M' \in R(M, \Delta) \mid \|m'_i - m_i\| = \Delta\}$ . By Theorem 1,  $R(M, \Delta)$  is maximum region if  $C(M', P) > C(M, P)$  for all  $M' \in B(m_i, M, \Delta)$  for any  $i$ .

Recall the 1-dimension 2-means example in Fig 3. The dashed rectangle forms the boundary of a maximal region  $R(M, \Delta)$ , since (1) the difference on either axis between  $M$  and any other center set in the rectangle is smaller than  $\Delta$ , and (2) any center set on the rectangle must have cost larger than 10.

We use  $R(M, \Delta)$  as our maximal region not only for its easiness to represent but also for its simpler analysis on the costs of the center sets on the boundary. Assume we are at the beginning of a new iteration in  $k$ -means algorithm and  $L = \{L_1, \dots, L_k\}$  is the point assignment after last iteration, i.e., all points in  $L_i \subseteq P$  are assigned to  $m_i \in M$  and  $m_i$  is the geometric center of the point set  $L_i$ . Given a point  $p \in L_i$ , we define  $d_1(p)$  to be the distance from  $p$  to its current cluster center  $m_i$ ,  $d_2(p)$  to be the distance from  $p$  to its nearest center in  $M$  and  $d_3(p)$  to be the distance to  $p$ 's second nearest center in  $M$ .

To find out whether  $R(M, \Delta)$  is a maximal region provided the value of  $\Delta$ , we divide the data set  $P$  into three subsets  $P_1, P_2$  and  $P_3$  according to the value of  $\Delta$  used in  $R(M, \Delta)$ .  $P_1$  contains all the points which will be assigned to some other center after the current  $k$ -means iteration, i.e.,  $P_1 = \{p \in P \mid d_1(p) > d_2(p)\}$ .  $P_2$  contains all the points in  $P - P_1$  that might be assigned to other cluster if centers move to  $M' \in \partial R(M, \Delta)$ , i.e.,  $P_2 = \{p \in P \mid d_1(p) > d_3(p) - 2\Delta\}$ .  $P_3$  contains all the other points not in  $P_1$  and  $P_2$ . In Fig 2, when the centers move from  $N = \{n_1, n_2\}$  to  $M = \{m_1, m_2\}$ ,  $q$  is in  $P_1$  since  $d_1(q) > d_2(q)$ , while  $p$  may be in  $P_2$  if  $d_1(p) > d_3(p) - 2\Delta$ , otherwise  $p$  is in  $P_3$ .

Let  $X(p) = d_1^2(p) - d_2^2(p)$  for any point  $p \in P_1$ , we have the following lemma.

**Lemma 3** Given the assignment  $L$  and center set  $M$  before a  $k$ -means iteration, the cost of  $M$  with respect to  $P$  is

$$C(M, P) = \sum_i C(m_i, L_i) - \sum_{p \in P_1} X(p)$$

*Proof:*  $\sum_i C(m_i, L_i)$  is the cost by assigning every point  $p \in L_i$  to  $m_i \in M$ , while  $X(p)$  is the cost reduction by reassigning  $p \in P_1$  to the nearest center in  $M$  instead of  $m_i$ . The difference gives the cost of  $C(M, P)$ .  $\square$

We also define the following two functions for points in  $P_1$  and  $P_2$  respectively. For point  $p \in P_1$ , we define

$$Y(p) = \begin{cases} (d_1(p) + \Delta)^2 - (d_2(p) - \Delta)^2 & \Delta \leq d_2(p) \\ (d_1(p) + \Delta)^2 & \Delta > d_2(p) \end{cases}$$

For point  $p \in P_2$ , we define

$$Z(p) = \begin{cases} (d_1(p) + \Delta)^2 - (d_3(p) - \Delta)^2 & \Delta \leq d_3(p) \\ (d_1(p) + \Delta)^2 & \Delta > d_3(p) \end{cases}$$

**Lemma 4** *Given the current center set  $M$ , its point assignment  $L$ , and any center set  $M' \in B(m_i, M, \Delta)$ ,  $C(M', P)$  is lower bounded by*

$$\sum_j C(m_j, L_j) + |L_i| \Delta^2 - \sum_{p \in P_1} Y(p) - \sum_{q \in P_2} Z(q)$$

*Proof:* Given  $M = \{m_1, \dots, m_k\}$ ,  $L = \{L_1, \dots, L_k\}$ , and  $M' = \{m'_1, \dots, m'_k\}$ , we first assign every  $p \in L_j$  to  $m'_j$ , whose cost is at least  $\sum_j C(m_j, L_j) + |L_i| \Delta^2$ . This is because  $m_i$  must move to  $m'_i$  by distance  $\Delta$ , and  $m'_j$  can stay at  $m_j$  for all  $j \neq i$ .

For every point  $p \in P_1 \cap L_j$ ,  $1 \leq j \leq k$ , the maximum distance to  $m'_j$  is  $d_1(p) + \Delta$ . The minimum distance to any center  $m'_i \neq m'_j$  is  $d_2(p) - \Delta$  if  $d_2(p) > \Delta$ , otherwise the minimum distance is 0. So, the cost reduction by reassignment for  $p$  can be no more than  $Y(p)$ .

For any point  $q \in P_2 \cap L_j$ , the maximum distance to  $m'_j$  is still  $d_1(q) + \Delta$ . But the minimum distance to  $m'_i \neq m'_j$  is  $d_3(q) - \Delta$  if  $d_3(q) > \Delta$  and is 0 otherwise.  $Z(q)$  can fully capture such cost reduction. So, the actual cost of  $M' \in B(m_i, M, \Delta)$  is lower bounded by the function above.  $\square$

**Theorem 2**  *$R(M, \Delta)$  is a maximum region if*

$$\Delta^2 \min_i |L_i| - \sum_{p \in P_1} (Y(p) - X(p)) - \sum_{q \in P_2} Z(q) > 0 \quad (3)$$

*Proof:*  $M' \in B(m_i, M, \Delta)$  for  $1 \leq i \leq k$  consist of the whole boundary of  $R(M, \Delta)$ ,  $R(M, \Delta)$  is a maximum region if  $C(M', P) > C(M, P)$  for any  $i$  and  $M' \in B(m_i, M, \Delta)$ . By combining Lemma 3 and Lemma 4, for  $M' \in B(m_i, M, \Delta)$ , we have

$$C(M', P) - C(M, P) \geq \Delta^2 |L_i| - \sum_{p \in P_1} (Y(p) - X(p)) - \sum_{q \in P_2} Z(q)$$

If we iterate all the boundary faces  $B(m_i, M, \Delta)$   $1 \leq i \leq k$ , the only difference in the inequality above is the number of points in  $L_i$ . If the inequality above can be satisfied for the smallest  $|L_i|$ , it must be valid for all boundary faces. So, inequality (3) is a sufficient condition to prove  $R(M, \Delta)$  is a maximal region.  $\square$

Since  $Y(p)$  and  $Z(p)$  are actually functions on  $\Delta$ , we call left hand of inequality (3) the *incremental function*  $f(\Delta)$ . Then  $R(M, \Delta)$  is a maximum region if  $f(\Delta) > 0$ . In the following part of the section, we will look at how we can lower bound the cost of the local optimum in a maximal region  $R(M, \Delta)$ .

### 3.3 Bounding Local Optimum in $R(M, \Delta)$

If a  $\Delta$  is found to satisfy the condition of Theorem 2, we can lower bound the cost of local optimum from the current center set  $M$  by the following theorem.

**Theorem 3** *Given a positive  $\Delta$  satisfying  $f(\Delta) > 0$ , if  $k$ -means algorithm converges to a center set  $M'$ ,  $C(M', P) \geq C(M, P) - |P| \Delta^2$ .*

*Proof:* Since the distance between every pair of corresponding centers  $m_i \in M$  and  $m'_i \in M'$  is no more than  $\Delta$ . The difference between the costs  $C(M, P)$  and  $C(M', P)$  is less than  $|P| \Delta^2$  by Lemma 1.  $\square$

The previous theorem provides an obvious lower bound on the local optimums falling in the maximal region  $R(M, \Delta)$ . It is obvious that the smaller  $\Delta$  is, the higher the lower bound can be. So, we should find the smallest  $\Delta$  satisfying  $f(\Delta) > 0$  to give the tightest lower bound. The details about how to find the smallest  $\Delta$  will be covered in the next section.

## 4 Algorithms

In the section, we provide the new algorithms. The first one is the bound iteration algorithm as the substitute of simple iteration algorithm, which can calculate the lower bound on the local optimum during the iteration process. The second one is the accelerated  $k$ -means algorithm as the substitute of the original  $k$ -means algorithm, which invokes the bound iteration algorithm or simple iteration algorithm on necessary.

### 4.1 Bound Iteration Algorithm

To combine the lower bound computation and simple iteration algorithm, we first need to find the information about  $d_1(p)$ ,  $d_2(p)$  and  $d_3(p)$  for every  $p \in P$  by simply calculating the distances during the search of  $p$ 's nearest center.

What is more difficult is how to find the smallest  $\Delta$  satisfying  $f(\Delta) > 0$ . There are two issues which make solving this inequality difficult. First, the sets of  $P_1$  and  $P_2$  are dynamic with different  $\Delta$ . Second, the values of  $Y(p)$  and  $Z(p)$  depend on  $\Delta$  as well.

To facilitate the computation, we first remove the impact of  $\Delta$  on the functions  $Y(p)$  and  $Z(p)$  by further dividing the sets  $P_1$  and  $P_2$ . We construct a subset of  $P_1$ ,  $P'_1 = \{p \in P | d_1(p) > d_2(p) \ \& \ d_2(p) < \Delta\}$ . In the same way, we construct another subset of  $P_2$ ,  $P'_2 = \{p \in P | d_3(p) - d_1(p) < 2\Delta \ \& \ d_3(p) < \Delta\}$ . By such construction, incremental function  $f(\Delta)$  becomes  $f(\Delta) = A\Delta^2 - 2B\Delta - C$ , where

the following parameter equations are easy to verify from Theorem 2.

$$\begin{aligned}
A &= \min |L_i| - |P'_1| - |P'_2| \\
B &= \sum_{P_1} (d_1 + d_2) + \sum_{P_2} (d_1 + d_3) - \sum_{P'_1} d_2 - \sum_{P'_2} d_3 \\
C &= \sum_{P_2} (d_1^2 - d_3^2) + \sum_{P'_1} d_2^2 + \sum_{P'_2} d_3^2
\end{aligned}$$

We note that a point can be in  $P_1$  and  $P'_1$  at the same time and likewise for  $P_2$  and  $P'_2$ . When the sets  $P_1, P'_1, P_2$  and  $P'_2$  are static, the incremental function is a pure quadratic function on  $\Delta$ . Since there are positive solutions for  $\Delta$  in  $f(\Delta) > 0$  only when  $A > 0$ , if we have an interval of  $\Delta \in [i_1, i_2]$  on which  $P_1, P'_1, P_2$  and  $P'_2$  are static, the maximal value of the incremental function must be achieved on either end of the interval. This gives us the intuition that we only need to test the extreme points on each interval with static  $P_1, P'_1, P_2$  and  $P'_2$ . Fortunately, the following lemma shows, the number of such intervals must be no more than  $2|P|$ .

**Lemma 5** *There are at most  $2|P|$  possible configurations of  $P_1, P'_1, P_2$  and  $P'_2$ .*

*Proof:* For every point  $p$ , it can change the configurations of the sets at most twice. For  $p \in P_1$ ,  $p$  is always in  $P_1$  no matter what  $\Delta$  is, and it is in  $P'_1$  when  $\Delta > d_2(p)$ . For  $p \in P - P_1$ , it is in  $P_2$  and  $P'_2$  when  $\Delta > (d_3(p) - d_1(p))/2$  and  $\Delta > d_3(p)$  respectively. Since these changes on the configuration can be sorted by  $\Delta$ , there are at most  $2|P|$  configurations. The bound is tight when no point is in  $P_1$ , which takes place at the convergence of the algorithm.  $\square$

With the analysis above, we give a scan algorithm which can find the minimum  $\Delta$  for  $f(\Delta) > 0$  if such  $\Delta$  exists. Briefly speaking, the algorithm gradually increases the value of  $\Delta$  from 0, keeping  $\Delta$  jumping to next event for configuration update. This can be done by sorting all the configuration update events for every point  $p$  as is shown in the proof of Lemma 5.

The detail of the algorithm is listed in Algorithm 3. For every point  $p \in P$ , the algorithm (Line 6 to 12) stores, in array  $Ar$ , the smallest  $\Delta$ s at which  $p$  changes the configuration, and how such change can influence the parameters in the function  $f(\Delta)$ . Such information is summarized in a 4-attribute element, where the first attribute is the  $\Delta$  value when update happens and the rest three are the differences on the parameters  $\{A, B, C\}$  when update takes place.

By visiting the elements in  $Ar$  in ascending order on the first attribute, the parameters  $A, B$  and  $C$  are updated accordingly. If at any moment, a  $\Delta$  satisfying  $f(\Delta) > 0$  is found, the algorithm returns the new center set  $M'$  as well as the lower bound by Theorem 3, otherwise the lower bound is 0, which is the trivial lower bound.

---

**Algorithm 3 BoundIterate** (center set  $M$ , data set  $P$ , current best cost  $C^*$ )

---

```

1: construct an array  $Ar$ 
2: set  $A = \min |L_i|, B = C = 0$ 
3: for every point  $p \in P$  do
4:   assign  $p$  to its nearest center in  $M$ 
5:   compute  $d_1(p), d_2(p)$  and  $d_3(p)$ 
6:   if  $p$  changes its clustering then
7:      $[d_2(p), -1, -d_2(p), d_2^2(p)]$  is inserted into  $Ar$ 
8:      $B+ = d_1(p) + d_2(p)$ 
9:   else
10:     $[\frac{d_3(p) - d_1(p)}{2}, 0, d_1(p) + d_3(p), d_1^2(p) - d_3^2(p)]$  is inserted into  $Ar$ 
11:     $[d_3(p), -1, -d_3(p), d_3^2(p)]$  is inserted into  $Ar$ 
12:   end if
13: end for
14: recompute the centers of the clusters and store in  $M'$ 
15: sort all the elements in  $Ar$  in ascending order on the first attribute
16: for every element  $r = [r_1, r_2, r_3, r_4]$  in  $Ar$  do
17:    $A+ = r_2, B+ = r_3, C+ = r_4$  and  $\Delta = r_1$ 
18:   if  $A < 0$  then
19:     return  $(M', 0)$ 
20:   end if
21:   if  $A\Delta^2 - 2B\Delta - C > 0$  then
22:     return  $(M', C(M', P) - |P|\Delta^2)$ 
23:   end if
24: end for
25: return  $(M', 0)$ 

```

---

Note that  $B$  and  $C$  must be positive and  $A$  is non-ascending with the increase of  $\Delta$  because any element inserted into  $Ar$  must have a non-positive second attribute. So, when  $A$  becomes negative, it is impossible to find any positive solution for  $\Delta$  any more. The iteration stops here and returns  $(M', 0)$  (Line 18).

The complexity of the bound iteration algorithm consists of four parts: points assignment, element insertion, element sorting and  $\Delta$  searching. Points assignment can be finished in  $O(|P|d)$  time. The insertion and searching take at most  $O(|P|)$  time since there are at most  $O(|P|)$  elements in  $Ar$  and every single operation in them can be done in constant time. The sorting on the elements in  $Ar$  takes  $O(|P| \log |P|)$  time by simply invoking existing sorting algorithms such as *quicksort* [4]. So, the total complexity of the bound iteration is  $O(|P|(\log |P| + d))$ .

## 4.2 Accelerated $K$ -Means Algorithm

Here, we present the complete accelerated  $k$ -means algorithm in Algorithm 4 to exploit the benefit from bound iteration algorithm. Similar to the original  $k$ -means algo-

---

**Algorithm 4 Accelerated  $k$ -means algorithm** (data set  $P$ ,  $k$ , current best cost  $C^*$ )

---

```

1: Randomly choose  $k$  points as the center set  $M$ 
2: while  $M$  does not change any more do
3:   if  $C(M, P) > C^*$  then
4:      $(M, \beta) = \text{BoundIterate}(M, P, C^*)$ 
5:     if  $\beta \geq C^*$  then
6:       Return NULL
7:     end if
8:   else
9:      $M = \text{SimpleIterate}(M, P)$ 
10:  end if
11: end while
12: if  $C(M, P) < C^*$  then
13:   Return  $M$  and Update  $C^*$ 
14: else
15:   Return NULL;
16: end if

```

---

gorithm, it first randomly chooses  $k$  points from the data set as the initial center set. Then, the algorithm keeps iterating until any of the following three cases happens: (1) a lower bound larger than current best cost is found; (2) the algorithm converges to some better solution than ever seen; (3) the algorithm converges but no better solution is found.

The iteration procedure in this algorithm can run either bound iteration algorithm or simple iteration algorithm. When the cost of the current center set is larger than the current best cost seen so far,  $C^*$ , the algorithm invokes the bound iteration algorithm (Line 4). If the lower bound returned by bound iteration is larger than  $C^*$ , it immediately stops the computation and return NULL (Line 5-6). When the cost of the current center set becomes smaller than  $C^*$ , the algorithm switches to the original simple iteration algorithm. This is because any lower bound in future iterations must be smaller than current best solution, which can not help to prune any more (Line 9). If the centers finally converge, the algorithm return the center set if its cost is smaller than  $C^*$ , otherwise NULL is returned (Line 12-15).

### 4.3 Further Optimization

Lemma 5 shows that we only need to test  $O(|P|)$  possible values to find a  $\Delta$  satisfying  $f(\Delta) > 0$ . The number of possible  $\Delta$  values can be further reduced. There are two types of  $\Delta$  values that we do not need to test. The first type contains all  $\Delta$ s whose corresponding  $B$  and  $C$  satisfying the condition  $\Delta < 2B/A$ , since quadratic equation has no positive solution in such situation. Second,  $\Delta$  can not be too large either. When  $\Delta$  is large enough, the lower bound achieved by such  $\Delta$  must be smaller than the best known cost  $C^*$ . Such a bound is useless since we cannot prune any

iteration if the lower bound is below  $C^*$ . The combination of the two ideas leads to the following lemma.

**Lemma 6** *Given the current optimal cost  $C^*$ , to find  $\Delta$  satisfying  $f(\Delta) > 0$  and  $C(M, P) - |P|\Delta^2 > C^*$ , the algorithm only needs to test  $\Delta$  in the interval  $[2 \frac{\sum_{p \in P_1} d_1(p)}{\min |L_i|}, \sqrt{\frac{C(M, P) - C^*}{|P|}}]$ .*

*Proof:* Since  $f(0) = 0$ ,  $0 < A < \min |L_i|$ ,  $B > \sum_{p \in P_1} d_1(p)$  and  $C$  is non-decreasing with the increase of  $\Delta$ . The positive solution of  $\Delta$  for  $f(\Delta) = 0$  must be larger than  $2 \frac{\sum_{p \in P_1} d_1(p)}{\min |L_i|}$ . On the other hand, when  $\Delta \geq \sqrt{(C(M, P) - C^*)/|P|}$ , the lower bound must be larger than  $C(M, P) - |P|\Delta^2 > C^*$ . We have no interests on such lower bound, since it can not be used to prune the current procedure any more.  $\square$

With such a lemma, we only need to sort and test the  $\Delta$  values in a certain interval, which can save much time in the calculation of the lower bound. This makes two changes on the bound iteration algorithms. First, only elements with first attribute smaller than the upper bound of interval are inserted. Second, the elements smaller than the lower bound of interval are directly used to update the parameters in  $f(\Delta)$  without testing  $f(\Delta) > 0$ .

## 5 Experiments

### 5.1 Experimental Setup

Since the result of  $k$ -means algorithm is fairly random, we run any single experimental item 5 times with different random seed for different algorithm. The average results are used in our performance evaluation. In this section, we use **OKM** to denote the original  $k$ -means algorithm and **AKM** to denote the accelerated  $k$ -means algorithm.

The generation of synthetic data sets follows the method used in previous studies on clustering problem, such as BIRCH [16]. We created 4 different data sets with 1,000,000 points on 4, 8, 16 and 32 dimensional spaces, respectively. The value of the points on every dimension is a float value between 0 and 1. Every data set consists of 40 small clusters, each of which occupies about 2.5% of the whole data set. Every cluster follows a Gaussian distribution, whose center and variance follow some uniform distributions. There are also some noisy points uniformly distributed in the space, whose size is 5% of the whole data set.

There are also two algorithm parameters in consideration, the target cluster number  $k$  and the  $k$ -means algorithm procedure number. The procedure number is the time the algorithm chooses the random initial center set and recompute the  $k$ -means result.

The performances of the algorithms were measured on the total computation time, the total number of iterations invoked and the individual numbers of both types of iterations invoked (simple iteration and bound iteration). Since AKM always outputs the correct clustering result, we omit the comparison of costs in these two algorithms.

In addition, we also conducted experiments on KDD98 data set<sup>1</sup> and KDD99 data set<sup>2</sup>. KDD98 data set is a 32 dimensional data set with 95000 records and KDD99 data set is a 41 dimensional data set with 310000 records. Both of the two data sets are collected for previous KDD-cups. Before using them in our experiments, we pruned all the non-numerical attributes and normalized those numerical attributes to float number between 0 and 1.

All experiments were carried out on a PC with 2GHz AMD Athlon processor and 2GB main memory. The programs were compiled with gcc 3.4.3 in Linux system.

## 5.2 Results on Synthetic Data Sets

The synthetic data tests are tested with varying dimensionality ( $D$ ), target cluster number ( $K$ ) and procedure number ( $R$ ). Unless otherwise stated, the default setting of the experiment is  $D = 8$ ,  $K = 16$  and  $R = 20$ .

We first test the effect of dimensionality on the performance both on the original  $k$ -means algorithm and our accelerated algorithm. As is shown in Fig 4(a), AKM does not show too much advantage over OKM when the dimensionality of the data set is small. This is because the extra cost on maintaining the necessary information, and the time to compute the lower bound is more than the time we can save on reducing the number of iteration. This can be verified in Fig 4(b) and Fig 4(c). The total iteration time of AKM is close to that of OKM and most of the iterations invoked by AKM on 2D data set are bound iterations. As the dimensionality grows, AKM becomes much faster than OKM since both the number of total iteration and the ratio of bound iteration decrease.

We also evaluate the impact of the parameter  $k$  in our experiments. The group of experiments are conducted on 16 dimensional data set, varying  $k$  from 8 to 32. These test results in Fig 5(a) and Fig 5(b) show that both the computation time and iteration time of OKM increases with the growth of  $k$ , while those of AKM do not change too much. Although the bound iteration also takes more time to calculate in every iteration, the increase in the ratio of simple iteration ensures that the efficiency of AKM is better than OKM with large  $k$ , as shown in Fig 5(c).

In Fig 6, we present the experiment result when we vary the procedure number for both OKM and AKM. When we run more times of the procedures on the same data set, the

computations of OKM and AKM both increase linearly. From Fig 6(b) and Fig 6(c), we can see that although the total iteration time of AKM is usually less than OKM, the ratio of bound iteration time to simple iteration time rises when procedure number increases.

## 5.3 Results on Real Data Sets

For the tests on real data set, we only vary two algorithm parameter: target cluster number  $k$  and procedure number  $R$ . The default value of  $k$  and  $R$  are 4 and 20, respectively.

Our AKM algorithm shows large advantage over OKM algorithm in the tests on KDD 99 data set. As is shown in Fig 7, when  $k$  is larger than 8, AKM is about 2 times faster than OKM. Similar to the results on synthetic data set, the total number of iterations in AKM is much fewer than that in OKM, which implies the strong pruning ability of our lower bound computation method. When varying the procedure number  $R$  on this data set, we see stable performance on AKM algorithm from Fig 8. This is also consistent with the test results on synthetic data sets.

However, when testing on KDD 98 data set, we see some results quite different. From Table 1, we can find that AKM algorithm can not be faster than OKM. The total iteration time in AKM is almost the same as the OKM. To find out the underlying reason for this phenomenon, we checked the procedures carefully and found that 90% of the procedures converge to the same global optimum in the space. This example implies the limitation of the algorithm proposed in this paper. Our algorithm can not estimate the number of local optimums in the space, which finally wastes much time on procedures converging to the same global optimum with more time-consuming bound iteration algorithm. We also argue that in such data set, there is no need to use multi-procedure  $k$ -means algorithm, since running  $k$ -means algorithm once is enough to find the optimal solution wanted.

## 6 Related Work

With different criterions on the clustering result, there are several independent but classic clustering problems, such as  $k$ -centers,  $k$ -means and  $k$ -medians. Han and Kamber's book [7] provides a good survey on the different clustering problems in data mining.

Lloyd's work [13] is one of the earliest application of  $k$ -means algorithm. To speed up the  $k$ -means algorithm, there are many accelerating methods proposed before. These methods can be divided into two categories. In the first categories, the triangle inequality property of metric space is exploited in the calculation of nearest center. Elkan [6] showed that many distance computation can be skipped if the triangle inequality is applied in the nearest center update process. The second category contains different method

<sup>1</sup><http://www.kdnuggets.com/meetings/kdd98/kdd-cup-98.html>

<sup>2</sup><http://www.ics.uci.edu/kdd/databases/kddcup99/kddcup99.html>

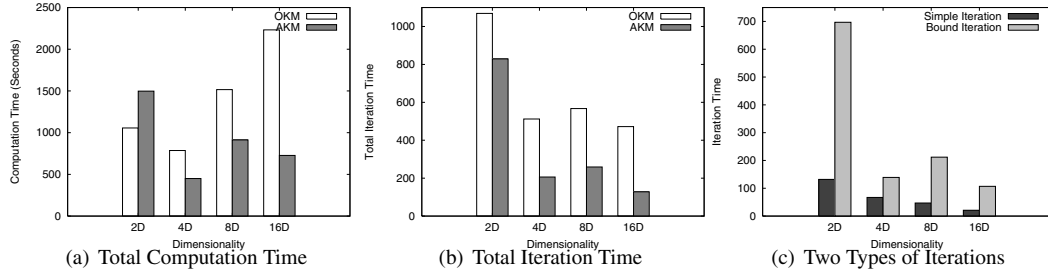


Figure 4: Tests on varying dimensionality on synthetic data set

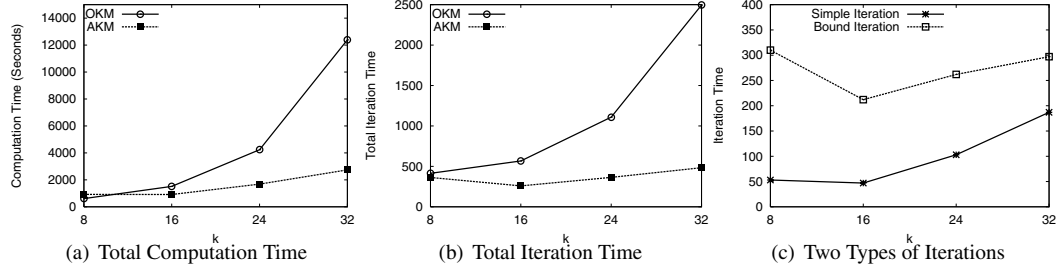


Figure 5: Tests on varying  $k$  on synthetic data set

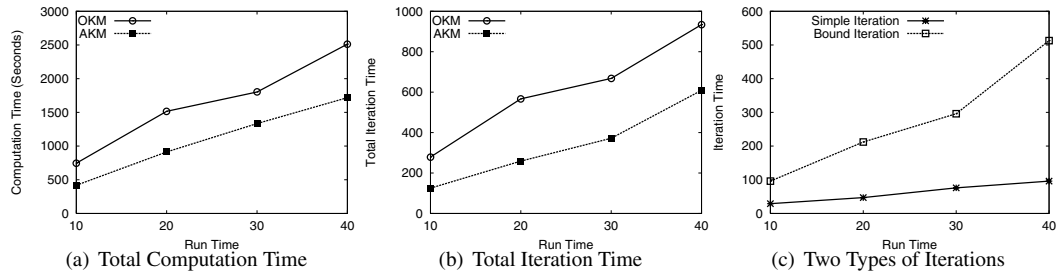


Figure 6: Tests on varying procedure number on synthetic data set

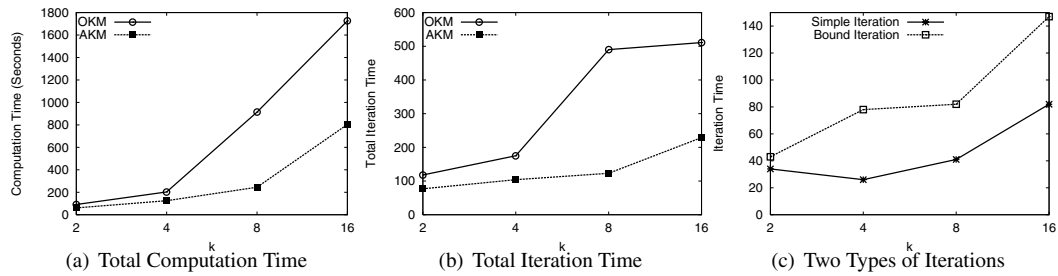


Figure 7: Tests on varying  $k$  on KDD 99 data set

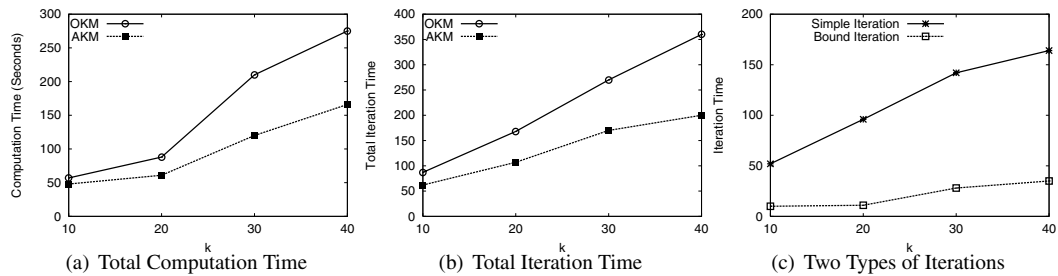


Figure 8: Tests on varying procedure number on KDD99 data set

Table 1: Test Results on KDD98 data set

$k$	OKM		AKM	
	Iteration Time	Computation Time	Iteration Time	Computation Time
2	726	14 Sec	417	9 Sec
4	2942	80 Sec	2939	95 Sec

based on indexing structure [10, 15]. The indexing structure, such as  $kd$ -tree can be used to improve the efficiency of the nearest center search when a group of points have the same nearest center. However, all of the studies mentioned here do not perform well in high dimensional space because of the curse of dimensionality.

The initial centers of  $k$ -means algorithm are very important to the result quality. The method proposed by [3] is a typical initial center refinement algorithm, which chooses the center set with the minimum distortion from a group of clustering results on some small samples of the original data set.

There are also a few of studies on the convergent property of  $k$ -means algorithm. [2] showed that  $k$ -means algorithm works very similar to gradient descent algorithm, which always moves toward the direction reducing most of the cost. [8] first proved some bound on the convergence speed of  $k$ -means algorithm. They gave an  $\Omega(n)$  lower bound on the number of iterations in standard  $k$ -means method, a  $O(n\Delta^2)$  upper bound on one-dimensional standard  $k$ -means method and a  $O(kn^2\Delta^2)$  upper bound on a variant  $k$ -means method, where  $n$  and  $\Delta$  are the size and the spread of the data set respectively. Recently, [1] proved that the lower bound of standard  $k$ -means iteration time is  $O(2^{\Omega(\sqrt{n})})$  by constructing a data set in  $\sqrt{n}$ -dimensional space.

Besides  $k$ -means algorithm, some theoretical computer scientists try to find good  $k$ -means clustering result with other techniques. [9] proposed a  $O(n^{O(kd)})$  algorithm to find the optimal solution and an  $\epsilon$ -approximate 2-means algorithm with  $O(n(1/\epsilon)^d)$  complexity. [12] extended the idea by a  $(1 + \epsilon)$ -approximate randomized algorithm with linear complexity to both dimensionality and data size. [11] proposed an  $(9 + \epsilon)$ -approximate local search algorithm which keep swapping the centers with other points in the data set to improve the clustering result. Ding and He [5] presented the relationship between  $k$ -means and PCA, which can lead to a lower bound on global optimum.

## 7 Conclusion

In this paper, we derive a lower bound on the cost of the local optimum based on the current center set. The  $k$ -means procedure can be terminated if the lower bound of

the cost at the future local optimum is higher than current best solution that has been computed so far. Experiments on both synthetic and real data sets reveal that such a method can greatly improve the efficiency in most of the data sets, especially with high dimensionality and large parameter  $k$ .

## References

- [1] D. Arthur and S. Vassilvitskii. How slow is the  $k$ -means method? In *SoCG*, pages 144–153, 2006.
- [2] L. Bottou and Y. Bengio. Convergence properties of the  $K$ -means algorithms. In *NIPS*, pages 585–592, 1995.
- [3] P. S. Bradley and U. M. Fayyad. Refining initial points for  $K$ -Means clustering. In *ICML*, pages 91–99, 1998.
- [4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms 2nd version*. The MIT Press, 2001.
- [5] C. H. Q. Ding and X. He.  $K$ -means clustering via principal component analysis. In *ICML*, 2004.
- [6] C. Elkan. Using the triangle inequality to accelerate  $k$ -means. In *ICML*, pages 147–153, 2003.
- [7] J. Han and M. Kamber. *Data Mining: Concept and Techniques*. Academic Press, 2000.
- [8] S. Har-Peled and B. Sadri. How fast is the  $k$ -means method? In *SODA*, pages 877–885, 2005.
- [9] M. Inaba, N. Katoh, and H. Imai. Applications of weighted voronoi diagrams and randomization to variance-based -clustering (extended abstract). In *Symposium on Computational Geometry*, pages 332–339, 1994.
- [10] T. Kanungo, D. Mount, N. Netanyahu, C. Piatko, R. Silverman, and A. Wu. An efficient  $k$ -means clustering algorithm: analysis and implementation, 2002.
- [11] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu. A local search approximation algorithm for  $k$ -means clustering. *Comput. Geom.*, 28(2-3):89–112, 2004.
- [12] A. Kumar, Y. Sabharwal, and S. Sen. A simple linear time  $(1+\epsilon)$ -approximation algorithm for  $k$ -means clustering in any dimensions. In *FOCS*, pages 454–462, 2004.
- [13] S. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28:129–137.
- [14] M. Meila and D. Heckerman. An experimental comparison of several clustering and initialization methods, 1998.
- [15] D. Pelleg and A. Moore. Accelerating exact  $k$ -means algorithms with geometric reasoning. In *Knowledge Discovery and Data Mining*, pages 277–281, 1999.
- [16] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: an efficient data clustering method for very large databases. In *SIGMOD Conference*, pages 103–114, 1996.