A Fast Adaptive k-means with No Bounds

Shuyin Xia*, Daowan Peng, Deyu Meng, Changqing Zhang, Guoyin Wang*, Elisabeth Giem, Wei Wei, Zizhong Chen

Abstract—This paper presents a novel accelerated exact k-means called as ”Ball k-means” by using the ball to describe each cluster, which focus on reducing the point-centroid distance computation. The ”Ball k-means” can exactly find its neighbor clusters for each cluster, resulting distance computations only between a point and its neighbor clusters' centroids instead of all centroids. What's more, each cluster can be divided into ”stable area” and ”active area”, and the latter one is further divided into some exact ”annular area”. The assignment of the points in the ”stable area” is not changed while the points in each ”annular area” will be adjusted within a few neighbor clusters. There are no upper or lower bounds in the whole process. Moreover, ball k-means uses ball clusters and neighbor searching along with multiple novel stratagems for reducing centroid distance computations. In comparison with the current state-of-the art accelerated exact bounded methods, the Yinyang algorithm and the Exponion algorithm, as well as other top-of-the-line tree-based and bounded methods, the ball k-means attains both higher performance and performs fewer distance calculations, especially for large-k problems. The faster speed, no extra parameters and simpler design of ”Ball k-means” make it an all-around replacement of the naive k-means.

Index Terms—Ball k-means, k-means, Ball Cluster, Stable Area, Active Area, Neighbor Cluster.

1 Introduction

Clustering problems arise in many fields, such as vector quantization [18], image compression [14], and spatial data mining [24]. Due to its simplicity and high efficiency, the k-means algorithm has become one of the top ten algorithms to solve clustering problems [38]. Stuart Lloyd at Bell Telephone Laboratories first proposed the algorithm in 1957, although it was only published by Lloyd in its widely-recognized form in 1982 [21]. The k-means clustering algorithm is based on the calculation of a distance function [17]. The algorithm proceeds in two phases: an assignment phase which assigns each point to a cluster based on the shortest point-centroid distance, and an update phase to recalculate the centroids for each cluster based on the assignment of each point. Each phase is repeated in order until the centroids stop moving. The cluster centroids in Lloyd's original k-means algorithm are initialized randomly, which can lead to a varying number of iterations and differing cluster results for each set of initial cluster centroids.

Lloyd's k-means algorithm, also referred to as the standard k-means or naive k-means algorithm, has been proven to be an NP-hard problem [1], [26]. The time complexity of the standard k-means algorithm is $O(nkt)$, where n denotes the data size, k represents the number of clusters, and t is the total number of iterations until the algorithm converges. In very large-scale clustering, the value of k is usually a few thousand or more, which can cause enormous time overhead in the standard k-means algorithm. In recent years, many different improved k-means algorithms have been proposed. Most focus on either optimizing the selection of the initial centroids, using approximate methods to accelerate the k-means algorithm, or accelerating the exact k-means algorithm.

We follow the last path. We build our k-means algorithm by introducing the idea of ball clusters and neighbor cluster searching to create an efficient and adaptive algorithm. The main contributions of this paper are as follows:

- We craft the novel ball k-means algorithm by introducing ball clusters as our main clustering tool. Ball k-means has a per-iteration time complexity that drops to sublinear levels as iterations progress, rendering it much more efficient than existing k-means algorithms.
- Ball k-means has no bounds and is parameter-free; that is, it does not need to keep track of any bounds for each data point.
- We introduce the concepts of neighbor cluster searching, an adaptive and efficient process that reduces the number of distance calculations. We also introduce the partitioning of ball clusters into stable and active areas, a process which is exact and does not rely on any bounds, allowing distances to be calculated not for the entire dataset, but rather only a small part. We use the information from previous iterations, resulting in the further reduction of centroid-centroid computations per iteration to less than $O(k^2)$.
- We show that ball k-means outperforms the two current

- *Shuyin Xia, Daowan Peng, and Guoyin Wang are with the Department of Chongqing Key Laboratory of Computational Intelligence, Chongqing University of Posts and Telecommunications, Chongqing 400065, China. E-mail: xiasy@cqupt.edu.cn, daowan_peng@qq.com, wanggy@cqupt.edu.cn*
- *Deyu Meng is with the National Engineering Laboratory for Algorithm and Analysis Technologiy on Big Data, Xi'an Jiaotong University,Xi'an 710049, China. E-mail: dymeng@xjtu.edu.cn*
- *Changqing Zhang is with the College of Intelligence and Computing, Tianjin University, 300072, China. E-mail:zhangchangqing@tju.edu.cn*
- *Zizhong Chen and Elisabeth Giem are with the Department of Computer Science and Engineering, University of California, Riverside,900 University Avenue, Riverside, CA 92521, USA. E-mail: chen@cs.ucr.edu, gieme01@ucr.edu*
- *Wei Wei is with the School of Computer Science and Engineering, Xi'an University of Technology. Xi'an 710048, China. E-mail: weiwei@xaut.edu.cn.*
- *Corresponding author: Guoyin Wang & Shuyin Xia*

state-of-the-art exact k-means algorithms as well as 5 other exact k-means algorithms, especially for large-k problems. Ball k-means is completely adaptive, simple, and easy to implement.

The rest of the paper is organized as follows: we introduce related works in Section 2, and then detail the background theory of ball k-means and neighbor cluster searching in Section 3. Section 4 presents a time complexity analysis of ball k-means. Evaluation results are given in Section 5. We present our conclusions in Section 6.

2 Related Work

k-means is a fundamental algorithm, allowing quite an extensive body of research into the problem to accumulate. There are three main categories of investigation in the literature: selection of the initial centroids, acceleration achieved by approximation, and acceleration of exact algorithms.

2.1 Initial Centroid Selection

Many algorithms have been proposed to improve the selection of the initial centroids in Lloyd's original k-means algorithm in order to help resolve the variance in iteration number and final cluster results due to the random initialization of the original centroids [2], [4], [5], [6], [23]. k-means++ is one of the most prominent of these improved methods [2]. Arthur and Vassilvitskii propose an adaptive sampling scheme called D^2-seeding that is $O(\log k)$-competitive with the optimal clustering. Bachem, Lucic, Hassani, and Krause build on k-means++ by replacing D^2 -seeding with a Markov chain Monte Carlo (MCMC) sampling method [4], and then further improve their work by removing assumptions on the MCMC method and theoretically guaranteeing the solution quality [5]. Another cluster of work is centered on the CLARANS algoritm, which is a two-parameter highly efficient algorithm structured as graph searching that performs best on smaller datasets [24]. Newling and Fleuret improve the CLARANS algorithm in both complexity and runtime, particularly enhancing efficiency for large datasets by utilizing more parameters in combination with the triangle inequality to reduce the number of comparisons [23]. While the subject of initial centroid selection is extensively researched, our method is focused on neighbor cluster searching and does not heavily rely on the initial selection of centroids.

2.2 Approximate k-means

Approximate k-means algorithms accelerate Lloyd's original k-means by approximating the clustering result [8], [12], [16], [19], [25], [28], [29], [31], [32], [34], [35], [37]. Wang, Jing, Ke, Gang, and Li use multiple random projection trees to establish cluster closures to speed up the algorithm [37]. Sculley introduces a mini-batch sampling method that converges to better solutions without increasing the computational cost on large datasets [34]. Hu, Wu, Bai, Zhang, and Cheng propose a multi-stage filtering k-means via a coarse-to-fine search strategy which accelerates the algorithm up to 634 times faster than standard k-means [16]. Deng and Zhao compare each data sample to its nearest neighbors by using an approximate k-nearest neighbor graph [8]. Fahim, Salem, Torkey, and Ramadan define an exclusion criterion which is based on the object-centroid distances in two successive iterations, allowing them to exclude distance calculations to any remaining centroids in an iteration if the distance from a point to a centroid is smaller than in the previous iteration, resulting in a a significant speedup especially on large datasets [12]. Tsai, Yang, and Chiang present an efficient algorithm; in each iteration, data patterns that are close to the centroid of a cluster or that remain in the same cluster for a given number of iterations in a row—and are thus unlikely to be moved from one cluster to another—are removed, and a new data pattern is computed that represents the removed patterns more compactly [35]. Pérez, Pires, Balby, Mexicano, and Hidalgo propose an algorithm which heuristically determines a threshold that decides which objects should be excluded using the probability that centroids will move in each iteration [29]. This algorithm was improved by Ortega et al. through the use of a new heuristic with an equidistance threshold for centroid movement and performs better on large datasets [28]. Approximate k-means has a rich history of research. However, our algorithm is an exact algorithm.

2.3 Accelerated Exact k-means

An alternative solution to speed up k-means is to accelerate an exact k-means algorithm [7], [9], [11], [14], [15], [18], [22], [27], [33], [39]. Pelleg and Moore propose the blacklist algorithm, which builds a kd-tree on the sample points [27]. This algorithm can assign many points to a cluster at once. However, the efficiency of kd-tree construction declines on a high-dimensional dataset, as well as for larger k values, making this algorithm inefficient for higher-dimensional datasets without heavy pre-processing. T. Kanungo et al. present a simple and efficient implementation of Lloyd's k-means clustering algorithm, called the filtering algorithm [18]. This algorithm is easy to implement, requiring a kd-tree as the only major data structure. Curtin proposes a dual-tree k-means algorithm based on four pruning strategies which builds two trees on both the data samples and the centroids [7]. The selected pruning strategies can rule out many centroids for many samples at once, so dual-tree k-means is much more efficient than the blacklist algorithm at large k values. They also suffer from the same inefficiency on some high-dimensional datasets due to the tree-based nature of the algorithm, and in addition the bounds on the pruning strategies can sometimes be too loose, resulting in a loss of efficiency.

Elkan applies the triangle inequality to bounds based on the distance from datapoints to the centroids to avoid some distance computations, resulting in a significant decrease in the amount of distance computations [11]. Each point, however, has k lower bounds, so the algorithm must maintain nk lower bounds per iteration, which is a severe limitation when both n and k are large. Hamerly improves this algorithm by reducing the number of lower bounds to n per iteration [14]. This resulted in Hamerly's algorithm becoming sensitive to big movers, that is, cluster centroids that drift dramatically in a centroid update. Big movers cause the bounds to fail, necessitating a calculation of the distance from the datapoint in question to every centroid [9]. The method still performs best in low-dimensional datasets. To improve Hamerly's algorithm, Newling and Fleuret propose the Exponion algorithm. In addition, they propose a technique for making bounds tighter for all bound based methods , allowing further redundant distance calculations to be eliminated, increasing its performance three times over its closest rival [22]. the Annulus algorithm [10], which applies the triangle inequality to an annulus centered on the origin. Ryšavỳ and Hamerly introduce a new algorithm with tighter bounds that does not use the triangle inequality, but rather takes into account the direction in which the centroid moves. This did result in some speedup, but it was limited and mostly apparent on datasets that were clearly clustered [33]. Ding, Zhao, Shen, Musuvathi, Mytkowicz compromise between Elkan's algorithm and Hamerly's algorithm by introducing three types of filtering, global, group, and local, which strategically reduces the amount of comparison operations [9]. This Yinyang k-means algorithm outperforms most existing k-means algorithms. However, the number of groups is an artificially selected parameter and has a significant influence on the efficiency of the algorithm, reducing the adaptive nature of the algorithm. Searching for the optimal number of groups takes time, which decreases algorithm efficiency. In this paper, we also present an accelerated exact k-means algorithm, so these previous works form the baselines to which we compare our work.

3 Ball k-means Clustering

In this section, we briefly review k-means in general, and then present our new algorithm, ball k-means. We introduce the concept of ball clusters, neighbor cluster searching, ball cluster partitioning, and our mechanism for reducing centroid-centroid distance computations between iterations, the method of finding stable ball clusters.

The standard k-means algorithm can be described as follows: Given a set of n samples, $\{x_1, x_2, ..., x_n\} \subset R^d$, where each sample represents a d-dimensional vector, and a positive number k, the k-means algorithm aims to partition these n samples into k clusters by minimizing the distortion, which is the within-cluster sum of the distances from each sample to its nearest centroid. This is expressed mathematically as:

$$J(x, c) = \sum_{j=1}^{k} \sum_{x_i \in C_j} \|x_i - c_j\|^2 , \tag{1}$$

where c_j is the centroid of cluster C_j, and the closest centroid to x_i. To optimize the objective function in (1), Lloyd's algorithm performs an assignment step and an update step iteratively until the cluster centroids stabilize. The assignment step assigns each sample to a cluster based on the shortest sample-centroid distance, and the update step recalculates the centroids of the clusters based on the new point assignments. These two steps are expressed mathematically as:

♦ *Assignment step:* each sample x_i is assigned to the cluster with the closest centroid:

$$b(x_i) = argmin_{j=1,..,k}\{\|x_i - c_j\|^2\}. \tag{2}$$

♦ *Update step:* each centroid c_j is updated using all samples assigned to cluster C_j:

$$c_j = \frac{1}{|C_j|} \sum_{i=1}^{n} \{x_i | b(x_i) = j\}, \tag{3}$$

where $|C_j|$ represents the number of sample points that are assigned to C_j.

3.1 Ball Clusters

The structure of a ball is characterized by a radius and centroid. We wish to use this structure to facilitate our k-means algorithm; we introduce the concept of the ball cluster, where a ball is used to describe a cluster.

Definition 1. Given a cluster C, we call C as a ball cluster by defining its centroid c and radius r as follows:

$$c = \frac{1}{|C|} \sum_{i=1}^{|C|} x_i, r = \max(\|x_i - c\|), \tag{4}$$

where x_i is a point that assigned to C, and $|C|$ denotes the number of samples in C.

3.2 Neighbor Cluster Searching

We wish to eliminate the distance calculations between points and centroids that are very far from each other. We introduce a method for finding the neighbor clusters of a given cluster, and limit the distance computation to points and their neighbor cluster centroids. We first define a neighbor cluster.

Definition 2. Given two ball clusters C_i and C_j with centroids c_i and c_j, C_j is a *neighbor cluster* of C_i if the radius r_i of C_i satisfies the following inequality:

$$\frac{1}{2} \|c_i - c_j\| < r_i. \tag{5}$$

From Equation 5 we can see that the neighbor relationship is not symmetric. Specifically, any two ball clusters C_i and C_j must have one of the following three relationships, as illustrated in Figure 1:

1) C_i and C_j are mutual neighbor clusters, exemplified by clusters C_2 and C_3 in Figure 1. As C_2 and C_3 are neighbor clusters, some points in C_3 (C_2) may be moved into C_2 (C_3) in the current iteration.
2) C_i is a neighbor cluster of C_j, but C_j is not a neighbor cluster of C_i; this case is demonstrated by clusters C_1 and C_3 in Figure 1. C_1 is a neighbor cluster of C_3, but C_3 is not a neighbor cluster of C_1. In this case, some points in C_3 may be moved into C_1, but no points in C_1 can be moved into C_3.
3) C_i and C_j have no neighbor relationship to each other. This case is illustrated by clusters C_3 and C_4 in Figure 1. In this case, no points in C_3 (C_4) can be moved into C_4 (C_3) in the current iteration.

The relationship assertions above are self-evident, but the restrictions on point movement may not be. We prove the limitations on the movement of points based on the neighbor relationships of clusters in the following theorem.

Theorem 1. Let C_i and C_j be two clusters with centroids c_i and c_j, respectively. For a queried ball cluster C with centroid c and radius r, let C_i be a neighbor cluster of C (that is,$r > \frac{1}{2}\|c - c_i\|$) and let C_j not be a neighbor cluster of C (that is, $r \leq \frac{1}{2}\|c - c_j\|$). Then 1) some points in C may be moved into C_i, and 2) no points in C can be moved into C_j.

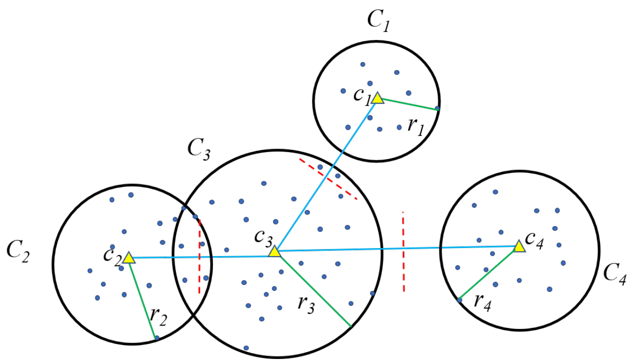


Fig. 1. Schema of neighbor relationships of the queried ball cluster C_3. The dashed red line represents the bisector of the segment connecting the centroids of two ball clusters. The yellow triangle and green line represent the centroid and radius of a cluster respectively.

Proof. $\forall x \in C$, $\|x - c\| \le r$.

1) Let C_j not be a neighbor cluster of C.

Then $r \le \frac{1}{2}\|c - c_j\|$ and we have

$\|x - c\| \le r \le \frac{1}{2}\|c - c_j\| = \frac{1}{2}\|c - x + x - c_j\|$,

and $\frac{1}{2}\|c - x + x - c_j\| \le \frac{1}{2}\left(\|c - x\| + \|x - c_j\|\right)$,

$\Rightarrow \|x - c\| \le \frac{1}{2}\left(\|c - x\| + \|x - c_j\|\right)$,

$\Rightarrow 2\|x - c\| \le \left(\|c - x\| + \|x - c_j\|\right)$,

$\Rightarrow \|x - c\| \le \|x - c_j\|$.

Therefore, for all points $x \in C$, x will not be moved into C_j. We can conclude that no points in a queried ball cluster can be moved into any non-neighbor cluster.

2) Let C_i be a neighbor cluster of C. Then $r > \frac{1}{2}\|c - c_i\|$ and $\exists x \in C$ such that $\|c - x\| > \frac{1}{2}\|c - c_i\|$. Let x be a point on the segment cc_i, which then satisfies:

(i) $\|c - x\| + \|c_i - x\| = \|c - c_i\|$, and

(ii) $\|c - x\| > \frac{1}{2}\|c - c_i\|$.

Then we have:

$\frac{1}{2}\|c - c_i\| = \frac{1}{2}\left(\|c - x\| + \|x - c_i\|\right)$,

and $\frac{1}{2}\|c - c_i\| < \|c - x\|$,

$\Rightarrow \frac{1}{2}\left(\|c - x\| + \|x - c_i\|\right) < \|c - x\|$,

$\Rightarrow \left(\|c - x\| + \|x - c_i\|\right) < 2\|c - x\|$,

$\Rightarrow \|c_i - x\| < \|c - x\|$

Therefore, points satisfying certain properties in C may be moved into C_i. We can conclude that it is possible that some points in a queried ball cluster may be moved into its neighbor clusters. □

The neighbor ball clusters of a queried cluster can be exactly found using Definition 2, so the required distance computations of points in C to the centroids of other clusters are limited to the centroids of the neighbor clusters of C. This results in a significant decrease in the amount of necessary distance computations. Ryšavŷ and Hamerly use a similar method of finding neighbor clusters [33]. Translating into our notation, they use the following method. For two clusters C_i and C_j with centroids c_i and c_j, $||c_i - c_j||$ is the distance between the centroids. Let $s(C_i)$ represent half the distance between C_i and its closest other centroid. If $r_i + s(C_i) \ge 1/2||c_i - c_j||$, then C_j is a neighbor of C_i. By contrast, in this paper, C_j is the neighbor of C_i if $r_i > 1/2||c_i - c_j||$. We see that in comparison with Definition 2, Ryšavŷ and Hamerly require one additional element in their neighbor condition, and therefore that their condition is looser than that in Definition 2. In other words, Definition 2 may result in finding less, but more exact, neighbor clusters than Ryšavŷ and Hamerly.

3.3 Ball Cluster Partitioning

Our goal is to perform the smallest amount of distance calculations possible. To this end, we show that a queried ball cluster can be divided into two parts, the stable area and the active area, in Definition 3. The points in the stable area stay in the assigned cluster, which we prove in Theorem 2. The active area can be further divided into annular areas, defined in Definition 4. We need to calculate the distances from the points in each annular area to only some of the neighbor clusters, which we prove in Theorem 3.

3.3.1 Stable and Active Areas

We first define the stable area and the active area as follows:

Definition 3. Let C_i be a queried ball cluster. $\{N_{C_i}\}$ denotes the centroid set of the neighbor clusters of C_i. If $\{N_{C_i}\} \neq \varnothing$, and if C_j is a ball cluster with centroid c_j and with $c_j \epsilon \{N_{C_i}\}$, then the *stable area* of C_i is defined as the spherical region with centroid c_i and radius $\frac{1}{2}min\left(\|c_i - c_j\|\right)_{c_j \in N_{C_i}}$. The remaining area including those points of C_i that are not in the stable area is defined as *active area* of C_i.

The points in the stable area of a given cluster will remain in that cluster, which we formulate as a theorem and prove now.

Theorem 2. Let C_i be a cluster. The points in the stable area of C_i cannot be moved into any neighbor cluster in the current iteration.

Proof. Let $\{N_{C_i}\}$ denote the centroid set of the neighbor clusters of C_i. If x is any point located in the stable area of C_i, then we have:

$\|x - c_i\| \le \frac{1}{2}min\left(\|c_i - c_j\|\right)_{c_j \in N_{C_i}}$ and

$\frac{1}{2}min\left(\|c_i - c_j\|\right)_{c_j \in N_{C_i}} \le \frac{1}{2}\|x - c_i\| + \frac{1}{2}\min\left(\|x - c_j\|\right)_{c_j \in N_{C_i}}$,

$\Rightarrow \|x - c_i\| \le \frac{1}{2}\|x - c_i\| + \frac{1}{2}\min\left(\|x - c_j\|\right)_{c_j \in N_{C_i}}$,

$\Rightarrow 2\|x - c_i\| \le \|x - c_i\| + \min\left(\|x - c_j\|\right)_{c_j \in N_{C_i}}$,

$\Rightarrow \|x - c_i\| \le \min\left(\|x - c_j\|\right)_{c_j \in N_{C_i}}$.

This means that the points in the stable area will stay in the current cluster for the current iteration. No point in the stable area of a queried ball cluster will be moved into any other cluster in the current iteration. □

We note that in the case when a ball cluster has no neighbor clusters, the stable area is equal to the entire ball cluster.

3.3.2 Active Area Partitioning

We now show how we partition the active area of a queried cluster into some annular areas—generated by neighbor clusters.

Definition 4. Let C be a queried ball cluster with centroid c and radius r, and let $\{N_C\}$ represent the set of centroids of neighbor clusters of C. Let $|\{N_C\}| = k'$ and $k' \neq 0$—for if $k' = 0$ then C has no neighbors and the stable area of C is the entirety of C. Let c_i and c_{i+1} represent the centroids of the i-closest and $(i+1)$-closest neighbor

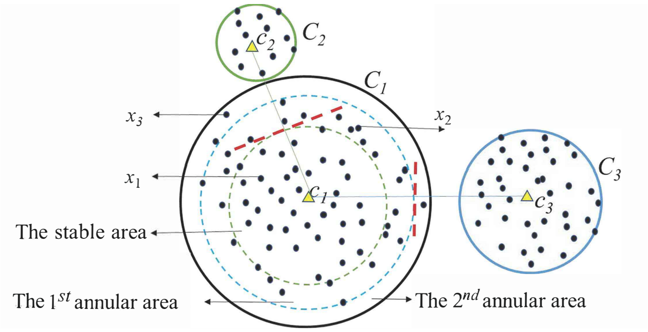


Fig. 2. Schema of the cluster partitions generated by neighbor balls. The dashed red line denotes the bisector of the segment connecting the centroids of two ball clusters.

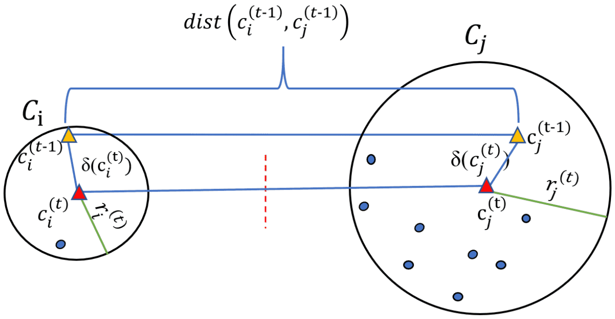


Fig. 3. Schema for avoiding direct centroid-centroid distance calculations. The dashed red line represents the midpoint of $dist(c_i^{(t)}, c_j^{(t)})$. C_j is not a neighbor cluster of C_i in the (t)-th iteration.

clusters of C, respectively ($i < k'$). The i^{th} *annular area* of C, denoted $\Re_C^i$, is defined $\forall x \in C$

$$\Re_C^i = \begin{cases} \frac{1}{2}\|c - c_i\| < \|x - c\| \le \frac{1}{2}\|c - c_{i+1}\|, & 0 < i < k' \\ \frac{1}{2}\|c - c_i\| < \|x - c\| \le r, & i = k' \end{cases}$$

We now introduce a theorem regarding the relative movement of points between these annuli.

Theorem 3. Let C be a queried cluster with centroid c, and let $|\{N_C\}| = k'$. Then the points in the i^{th} annular area of C can only be moved within the first i-closest neighbor clusters and itself ($i \le k'$).

Proof. Let c_i and c_{i+1} represent the centroids of the i^{th} and $(i+1)^{th}$ closest neighbor clusters of C, respectively, with $i < k'$. Let x be a point in the i^{th} annular area of C. We have:

$\|c - x\| \le \frac{1}{2}\|c - c_{i+1}\|$
and $\frac{1}{2}\|c - c_{i+1}\| \le \frac{1}{2}\|c - x\| + \frac{1}{2}\|x - c_{i+1}\|$,
$\Rightarrow \|c - x\| \le \frac{1}{2}\|c - x\| + \frac{1}{2}\|x - c_{i+1}\|$,
$\Rightarrow 2\|c - x\| \le \|c - x\| + \|x - c_{i+1}\|$,
$\Rightarrow \|c - x\| \le \|x - c_{i+1}\|$.

This tells us that points in the i^{th} annular area are closer to the centroid of the queried ball cluster than to the centroid of its $(i+1)^{th}$-closest neighbor cluster. We can conclude that the $(i+1)^{th}$-closest neighbor cluster of the queried cluster is not a neighbor cluster of the i^{th} annular area. Thus, the points in the i^{th} annular area can only participate in assignment step within the queried ball cluster itself and its first-i closest neighbor clusters. □

The stable area and active area partitioning are illustrated in Figure 2. Both C_2 and C_3 are neighbor clusters of C_1. C_2 is the nearest neighbor cluster of C_1, and the area enclosed by the dashed green circle is the stable area. Points in the stable area, such as x_1, do not participate in the assignment step. The 1^{st} annular area is the area between the dashed green circle and dashed blue circle. Points in the 1^{st} annular area, such as x_2, may only move within C_1 and C_2 during the assignment step. The 2^{nd} annular area is the area between the dashed blue circle and the solid black circle. Points in the 2^{nd} annular area, such as x_3, may move within C_1, C_2 and C_3 during the assignment step.

3.4 Reducing Centroid-Centroid Distance Computations between Iterations

As seen in Section 3.2, finding the neighbor clusters of each ball cluster requires the computation of all centroid-centroid distances, which costs $O(k^2)$ per iteration. For large k clustering, this is a non-negligible cost. The purpose of calculating centroid-centroid distances is to find the neighbor clusters for the next iteration. If a non-neighbor relationship in the next iteration can be discovered in advance using the relationship of the ball clusters in the current iteration, then directly calculating the centroid-centroid distances could be avoided. We now develop a method to implement this idea, attempting to find the non-neighbor relationships in advance to avoid unnecessary calculation of centroid-centroid distances. We first list some helpful notation, and then present a theorem towards this end.

We let $c_i^{(t)}$ represent the centroid of the cluster C_i in the t^{th} iteration, and we let $\delta(c_i^{(t)}) = \|c_i^{(t)} - c_i^{(t-1)}\|$ represent the movement of the cluster centroid of C_i between the $(t-1)^{th}$ iteration and the t^{th} iteration. We also say that $dist(c_i^{(t)}, c_j^{(t)})$ represents the distance between c_i and c_j in the t^{th} iteration.

Theorem 4. Let C_i and C_j be clusters. If $dist(c_i^{(t-1)}, c_j^{(t-1)}) \ge 2r_i^{(t)} + \delta(c_i^{(t)}) + \delta(c_j^{(t)})$, then C_j cannot be the neighbor cluster of C_i in the current iteration and the calculation of the centroid-centroid distance between them may be elided.

Proof. With the shift in cluster centroids due to the centroid update, we have:

$dist(c_i^{(t)}, c_j^{(t)}) \ge dist(c_i^{(t-1)}, c_j^{(t-1)}) - \delta(c_i^{(t)}) - \delta(c_j^{(t)})$.
We assumed in the theorem statement that:
$dist(c_i^{(t-1)}, c_j^{(t-1)}) \ge 2r_i^{(t)} + \delta(c_i^{(t)}) + \delta(c_j^{(t)})$.
Combining these, we have:
$dist(c_i^{(t)}, c_j^{(t)}) \ge 2r_i^{(t)} + \delta(c_i^{(t)}) + \delta(c_j^{(t)}) - \delta(c_i^{(t)}) - \delta(c_j^{(t)})$.
However, we can see that:
$2r_i^{(t)} + \delta(c_i^{(t)}) + \delta(c_j^{(t)}) - \delta(c_i^{(t)}) - \delta(c_j^{(t)}) = 2r_i^{(t)}$,
$\Rightarrow dist(c_i^{(t)}, c_j^{(t)}) \ge 2r_i^{(t)}$.

But from Definition 2 and Theorem 1, if $2r_i \le dist(c_i, c_j)$ then C_j cannot be the neighbor of C_i in the current iteration. Therefore, the computation of the distance between c_i and c_j can be avoided. □

We illustrate this theorem in Figure 3. We can see here that C_j cannot be a neighbor cluster of C_i in the current iteration, because the distance between the two centroids in the previous iteration is greater than the shift in centroids added to twice the radius if C_i, that is, $dist(c_i^{(t-1)}, c_j^{(t-1)}) \geq 2r_i^{(t)} + \delta(c_i^{(t)}) + \delta(c_j^{(t)})$.

3.5 Stable Ball Cluster in Subsequent Iterations

In the ball k-means algorithm, we use *stable ball cluster* to refer to a ball cluster in which no points move into the cluster and no points move out of the cluster in the current iteration. Due to the fundamental characteristics of the k-means algorithm, as the number of iterations increases, the general trend is that more and more ball clusters become stable ball clusters, that is, the points within them remain unchanged. Based on the characteristics of the k-means algorithm, we introduce a method of finding these stable ball clusters. A flag corresponding to a ball cluster is used to judge whether a ball cluster is stable. For a queried ball cluster, if no points in the queried ball cluster move into the neighbor clusters, and no points in any other ball cluster move into the queried ball cluster in the current iteration, then the flag is marked `TRUE`.

Theorem 5. Assume ball k-means is implemented on a given dataset D. Let C be a queried ball cluster. If all the neighbor ball clusters of C are stable in one iteration, C will not take part in the distance calculations in the next iteration.

Proof. The proof is straightforward. For a queried ball cluster C, if all the neighbor ball clusters of C are stable, then the partitioning of the stable area and annular areas are the same as those in the previous iteration. Therefore, the assignment step of the queried ball cluster can be avoided. □

As the iterations progress during the execution of the ball k-means algorithm, more and more ball clusters will become stable, and the data points in the stable ball clusters will not figure in to any distance calculations. Therefore, the time complexity of ball k-means per iteration will become sublinear, and ball k-means will execute each iteration faster and faster. We present the ball k-means algorithm in detail in Algorithm 1. The processes of reducing centroid-centroid distance computations between iterations is shown in Algorithm 2, and the process of filtering points by the stable and annular areas is presented in Algorithm 3. We have made all source codes available at https://github.com/syxiaa/ball-k-means.

Algorithm 1 Ball k-means algorithm

Input: data $X = x_1, \ldots, x_n \subset R^d$, the number of clusters k, initial centroids $c_1, \ldots, c_k \in R^d$;
Output: cluster centroids $c_1, \ldots, c_k$

```
After one standard $k$-means iteration
Set t=1 //$t$ is the number of the $t^{th}$ iteration, $t = 1, 2, \ldots$ until the algorithm converges
$flag_i$ = FALSE$(i = 1, \ldots, k)$ //Determine whether the center and radius need to be updated
repeat
  for $i = 1, \ldots, k$ do
    if $flag_i$ = FALSE then
      $c_i^{(t)} = mean(x|x \in C_i)$ // Update the centroids.
      $\delta(c_i^{(t)}) = \|c_i^{(t)} - c_i^{(t-1)}\|$ //$c_i^{(0)}$ represents the initial centroid of $c_i$
      Calculate radius $r_i^{(t)}$ via Equation (4)
    end if
  end for
  if $t = 1$ then
    Calculate the distances between any two centroids $dist(c_i^{(t)}, c_j^{(t)})(i, j = 1, \ldots, k)$ //Initialize the centroid distance matrix
  else
    Update centroid distance matrix according to Alg. 2
  end if
  for $i = 1, \ldots, k$ do
    Set $\{N_{C_i}^{(t)}\} = \varnothing$ as the set of centroids of neighbor clusters for $C_i$ in the current iteration.
    for $j = 1, \ldots, k$ do
      if $dist(c_i^{(t)}, c_j^{(t)}) < 2r_i^{(t)}$ then
        Append $c_j$ to $\{N_{C_i}\}$
      end if
    end for
    Reassign the points in cluster $C_i$ according to Alg.3
  end for
  for $i = 1, \ldots, k$ do
    if $C_i$ is stable then
      $flag_i$ = TRUE
    else
      $flag_i$ = FALSE
    end if
  end for
  t=t+1
until cluster centroids stop changing
```

Algorithm 2 Reducing centroid-centroid distance computations between iterations

```
for $i = 1, \ldots, k$ do
  for $j = 1, \ldots, k$ do
    if $dist(c_i^{(t-1)}, c_j^{(t-1)}) \geq 2r_i^{(t)} + \delta(c_i^{(t)}) + \delta(c_j^{(t)})$ then
      $dist(c_i^{(t)}, c_j^{(t)}) = dist(c_i^{(t-1)}, c_j^{(t-1)}) - \delta(c_i^{(t)}) - \delta(c_j^{(t)})$
    else
      $dist(c_i^{(t)}, c_j^{(t)}) = \|c_i^{(t)} - c_j^{(t)}\|$
    end if
  end for
end for
```

Algorithm 3 Filtering points by the stable and annular areas

```
if $t \neq 1$ and $N_{C_i}^{(t)} = N_{C_i}^{(t-1)}$ and $flag_i = $ TRUE and all $flag_j = $ TRUE$(c_j \in N_{C_i})$ then
    continue //The points in $C_i$ will not change during the current iteration
else
    Sort $\{N_{C_i}\}$ ordering from smallest to largest distance
    for each point $x$ in cluster $C_i$ do
        if $dist(c_i^{(t)}, x) < \frac{1}{2}min(dist(c_i^{(t)}, c_j^{(t)}))(c_j \in N_{C_i})$ then
            continue
        else
            if point $x$ in $h^{th}$ annulus area $(|N_{C_i}| = k', h = 1, \dots, k')$ then
                Calculate the distances from $x$ to its first $h$ closest neighbor cluster centroids, and assign $x$ to the closest cluster
            end if
        end if
    end for
end if
```

4 Time Complexity Analysis

Let n represent the number of data points in a data set. The ball k-means is implemented on this data set. The ball k-means algorithm consists of two main parts: searching neighbor clusters and assigning points. To search for the neighbor ball clusters of a queried cluster, we must compute the distances to up to $k-1$ cluster centroids, which costs $O(k^2)$ in the worst case. In addition, as we showed in Theorem 4 in Section 3.4, we do not need to compute the distances to all of the other $k-1$ cluster centroids except the first iteration, but only to those cluster centroids near to the queried cluster, i.e. those fulling the condition in Theorem 4. This means that, in practice, searching the neighbor clusters requires a time of less than $O(k^2)$ in some iterations, especially in the later iterations of ball k-means.

We denote the average number of neighbor ball clusters of a queried cluster and the number of data points in active areas as m $(1 \leq m \leq k)$ and n' respectively. Computing the distance from all data points in active areas to the centroids of their neighbor clusters is equal to $O(mn')$. Besides, we need to compute all distance of data points to the centroids of their clusters, which will cost a time complexity of $O(n)$.

In addition, as shown in Algorithm 1, we need to sort the neighbor clusters for a queried ball cluster C, ordering from the smallest to largest distance from C. Implementing a quick sorting algorithm such as merge sort on the m distances from the centroid of C to the centroids of its m neighbor clusters will add time of $O(m \log m)$. Multiplying this by all k centroids results in a time complexity of $O(km \log m)$ per iteration for sorting in the worst case scenario. Because the neighbor clusters of a queried cluster will have some stability between two close iterations, we first search the neighbor clusters in the current iteration in the order of the neighbor clusters of the last iteration, then search any remaining neighbor clusters. In this way, the distances of the neighbor clusters to the queried cluster are almost ordered before the sort, with the ordering increasing in later iterations as the clustering results become more stable. Some preexisting order in the set to be ordered leads to a smaller cost for many sorting algorithms. Therefore, the time complexity of sorting the m distances from the centroid of C to the centroids of its m neighbor clusters is generally less than $O(m \log m)$, especially in the later iterations.

In summary, considering the worst cases, adding all other obvious loops from the algorithm, we arrive at a per-iteration total time complexity for ball k-means of $O(k^2 + km \log m + mn' + n)$. In addition, as shown in Theorem 5, during the execution of ball k-means, an increasing number of ball clusters will become stable, and the datapoints in these stable ball clusters will not figure into any distance calculations. The time complexity of ball k-means per iteration will drop to the sublinear level in later iterations. Consequently, ball k-means is very efficient in practice.

The per-iteration time complexities, setup costs, and space costs in the worst case scenarios for our baseline algorithms (Lloyd, Hamerly, Dualtree_kd,Blacklist,Yinyang, Ann and Exp) compared to our ball k-means algorithm are shown in Table 1. As we can see, the standard k-means costs $O(kn)$ per iteration. The worst-case runtime of Hamerly's algorithm is $O(k^2 + kn)$, and the Yinyang and Blacklist algorithms cost $O(kn)$ in the worst case. Although the Annulus and Exponion algorithms are efficient, they have a high time complexity than others. The dualtree algorithm has a competitive time cost similar to ours, but its time complexity depends on some assumptions about dataset-dependent constants—the imbalance of the dualtree [7]. Performance can deteriorate significantly on balanced trees, especially on high-dimensional datasets; experimental results confirm this statement.

5 Evaluation

5.1 Performance on Test Datasets in Contrast with Known k-means Algorithms

We demonstrate the efficiency of ball k-means by evaluating the performance of our approach on a variety of real-world datasets and comparing it with those of seven other exact k-means algorithms. We choose the following as our baselines: the two fastest known k-means algorithms, Yinyang k-means [9] (improved version Syin-ns [22]) and the Exponion algorithm(Exp-ns) [22], the fastest known k-means for large k problems, dualtree_kd [7], another three fast k-means, blacklist [27], Hamerly's algorithm [14], [22], the Annulus algorithm(Ann) [10], and finally the standard k-means algorithm. Hamerly's algorithm is an improvement on Elkan's algorithm [11], eclipsing it in most cases [22], so we have only considered Hamerly's algorithm here. We also

TABLE 2
Dataset information

Data sets	Size	Dimension
Four-class	862	3
Svmguide1	7088	5
Codrna	59535	8
Kegg Network	65554	28
Epileptic	11500	179
Birch3	100000	2
Ijcnn	141690	22

TABLE 1
Per-iteration time complexity and space cost(n: points; k: clusters; -: N/A)

Algorithm	Setup	Worst-case	Space cost
Lloyd	-	$O(kn)$	$O(k+n)$
Hamerly	-	$O(k^2+kn)$	$O(k+n)$
Dualtree_kd	$O(nlogn)$	$O(klogk+n)$	$O(k+n)$
Blacklist	$O(nlogn)$	$O(kn)$	$O(klog(n)+n)$
Yinyang	$O(k^2+kn)$	$O(kn)$	$O(kn)$
Annulus	$O(k+n)$	$O(klog(k)+nlog(k)+k^2+kn)$	$O(k^2+kn)$
Exponion	-	$O(k^2log(k)+nloglog(k)+k^2+kn)$	$O(k^2+kn)$
Ball k-means	$O(k^2+kn)$	$O(k^2+km\log m+mn'+n)$	$O(k^2+kn)$

TABLE 3
Number of distance calculations and speedup over Lloyd

Data sets	k	iter	Lloyd	Hamerly	Dualtree_kd	Blacklist	Syin-ns	Ann	Exp-ns	Ball k-means
Four-class	30	12	3.E+05	7.E+04(4.53)	3.E+04(10.73)	7.E+04(4.46)	5.E+04(6.09)	7.E+04(4.66)	6.E+04(5.20)	**8.E+03(37.35)**
	50	20	9.E+05	2.E+05(4.67)	7.E+04(12.73)	2.E+05(4.49)	9.E+04(9.74)	2.E+05(5.52)	1.E+05(7.59)	**8.E+03(109.62)**
	100	10	9.E+05	3.E+05(2.8)	7.E+04(13.20)	2.E+05(4.61)	1.E+05(7.74)	2.E+05(3.45)	2.E+05(4.46)	**5.E+03(181.53)**
Svmguide1	30	35	7.E+06	2.E+06(4.83)	**6.E+05(11.93)**	1.E+06(6.11)	8.E+05(9.36)	1.E+06(6.01)	1.E+06(5.96)	7.E+05(10.89)
	50	28	1.E+07	3.E+06(3.18)	8.E+05(11.88)	1.E+06(6.89)	1.E+06(9.39)	2.E+06(4.67)	2.E+06(5.08)	**6.E+05(16.23)**
	100	45	3.E+07	9.E+06(3.37)	2.E+06(15.69)	4.E+06(7.38)	2.E+06(18.67)	5.E+06(5.99)	4.E+06(9.01)	**1.E+06(32.11)**
Codrna	30	99	2.E+08	3.E+07(5.98)	1.E+07(14.83)	3.E+07(6.71)	**1.E+07(15.22)**	3.E+07(6.46)	3.E+07(6.36)	3.E+07(5.62)
	50	71	2.E+08	5.E+07(3.88)	**2.E+07(13.66)**	3.E+07(7.6)	2.E+07(12.99)	4.E+07(4.75)	4.E+07(5.08)	2.E+07(8.63)
	100	102	6.E+08	1.E+08(4.14)	4.E+07(17.07)	7.E+07(8.88)	**2.E+07(24.35)**	1.E+08(5.99)	8.E+07(8.02)	5.E+07(11.15)
Kegg Network	30	78	2.E+08	3.E+07(4.81)	**1.E+07(15.02)**	2.E+07(8.42)	2.E+07(6.59)	3.E+07(5.89)	3.E+07(4.96)	3.E+07(5.20)
	50	77	3.E+08	5.E+07(4.85)	**1.E+07(22.22)**	2.E+07(11.43)	2.E+07(11.68)	3.E+07(8.25)	4.E+07(5.65)	3.E+07(9.36)
	100	64	4.E+08	9.E+07(4.54)	**2.E+07(26.03)**	3.E+07(12.72)	3.E+07(16.12)	5.E+07(8.9)	6.E+07(7.19)	2.E+07(18.49)
	300	39	8.E+08	2.E+08(3.64)	3.E+07(27.01)	6.E+07(12.68)	3.E+07(22.68)	9.E+07(8.21)	7.E+07(10.35)	**2.E+07(50.02)**
Epileptic	30	51	2.E+07	8.E+06(2.3)	5.E+06(3.32)	1.E+07(1.25)	6.E+06(3.19)	4.E+06(4.95)	**4.E+06(4.34)**	5.E+06(3.79)
	50	29	2.E+07	1.E+07(1.28)	7.E+06(2.46)	1.E+07(1.43)	8.E+06(2.12)	5.E+06(3.51)	6.E+06(3.00)	**4.E+06(4.64)**
	100	50	6.E+07	4.E+07(1.54)	2.E+07(3.47)	4.E+07(1.6)	2.E+07(3.32)	1.E+07(5.91)	1.E+07(4.97)	**1.E+07(5.48)**
	300	57	2.E+08	1.E+08(1.61)	5.E+07(3.77)	1.E+08(1.64)	5.E+07(3.92)	3.E+07(6.79)	3.E+07(5.64)	**3.E+07(7.07)**
	750	43	4.E+08	3.E+08(1.42)	1.E+08(3.38)	2.E+08(1.6)	6.E+07(6.24)	7.E+07(5.68)	8.E+07(4.70)	**4.E+07(9.96)**
Birch3	30	26	8.E+07	7.E+06(10.57)	**4.E+05(183.49)**	7.E+05(118.83)	6.E+06(13.92)	6.E+06(13.43)	5.E+06(15.13)	2.E+06(36.71)
	50	62	3.E+08	2.E+07(18.03)	**1.E+06(243.11)**	2.E+06(125.41)	1.E+07(32.45)	1.E+07(29.84)	8.E+06(39.50)	4.E+06(83.17)
	100	79	8.E+08	6.E+07(13.69)	**3.E+06(248.74)**	6.E+06(129.39)	2.E+07(43.99)	2.E+07(34.61)	2.E+07(51.78)	5.E+06(151.52)
	300	123	4.E+09	4.E+08(9.21)	1.E+07(329.86)	2.E+07(148.62)	4.E+07(82.61)	9.E+07(43.36)	5.E+07(67.83)	**9.E+06(426.87)**
	750	113	8.E+09	1.E+09(6.7)	2.E+07(478.12)	5.E+07(166.48)	9.E+07(92.94)	2.E+08(37.3)	2.E+08(54.69)	**6.E+06(1493.32)**
Ijcnn	30	44	2.E+08	3.E+07(7.2)	**1.E+07(16.50)**	3.E+07(5.9)	1.E+07(13.27)	3.E+07(6.5)	2.E+07(10.14)	2.E+07(11.19)
	50	68	5.E+08	7.E+07(6.5)	3.E+07(17.47)	8.E+07(5.87)	**3.E+07(18.45)**	8.E+07(6.26)	4.E+07(11.84)	3.E+07(17.71)
	100	88	1.E+09	3.E+08(4.92)	7.E+07(17.76)	2.E+08(5.87)	**4.E+07(29.27)**	2.E+08(5.33)	8.E+07(15.11)	5.E+07(27.03)
	300	88	4.E+09	1.E+09(3.14)	2.E+08(19.19)	6.E+08(6.06)	8.E+07(49.21)	9.E+08(4.19)	2.E+08(21.46)	**6.E+07(64.06)**
	750	83	9.E+09	4.E+09(2.39)	4.E+08(20.46)	1.E+09(6.14)	2.E+08(57.35)	2.E+09(4.01)	3.E+08(25.61)	**7.E+07(123.65)**

use improved and accelerated versions of the Yinyang in [22] and Annulus algorithms [10] in our experiments. The terminal condition that we stop the k-means iterations is that each cluster centroids stop changing.

We use the same initial centroid seeds for all algorithms; because the algorithms are exact, they all converge to the same clustering result after the same number of iterations. We generated the centroid seeds using the method proposed by Bachem et al. [5]. We implemented all of the algorithms in the C++ programming language on a laptop PC with a Eight core Intel CPU I9-9900 with 4GB DRAM. We evaluated them for a variety of k values on seven real-world datasets, six of which are from the UCI Machine Learning Repository [3], and the last of which, Birch3, is a commonly used data set [13]. We provide pertinent information about the datasets in Table 2. In the Yinyang algorithm, we set $t = k/10$, where t is the number of lower bounds per point; this led to the optimal performance [9]. We show experimental results regarding the number of distance calculations and runtime calculations in Tables 3-6. We mark the fastest algorithm on each data set in **boldface** in each of these tables.

Tables 3 and 4 show that our ball k-means algorithm has the smallest number of distance calculations and the highest efficiency in most cases. Even in some cases where the number of distance calculations ball k-means performs is greater than that of other algorithms, ball k-means is still more efficient than other algorithms. This is because ball k-means can exactly and adaptively find a tight set with no bounds: in contrast, in other accelerated k-means algorithms, including the Hamerly, Yinyang, Annulus, and Exponion algorithms, each sample requires the computation of upper and lower bounds. These bounds will cost more computations than the direct distance calculations in ball k-means, but will not be counted in the distance calculation metric. Another contributing factor, as described in Section 4, is that the time complexity of later iterations in ball k-means is lower than $O(n)$.

Tables 5 and 6 show that the advantage of ball k-means is more obvious in large-k problems. This is because the percentage of neighbor ball clusters for a queried ball cluster, out of the overall number of ball clusters, is very likely to be smaller when k increases. This indicates that ball k-means is more efficient than other exact k-means in large-k problems.

Hamerly's algorithm did not perform well on high-

TABLE 4
Speedup over Lloyd and running time ((ms)/iteration)

Data sets	k	iter	Lloyd	Hamerly	Dualtree_kd	blacklist	Syin-ns	Ann	Exp-ns	Ball k-means
Four-class	30	12	0.31	0.29(1.08)	0.3(1.06)	0.2(1.6)	0.08(3.76)	0.08(3.76)	0.08(3.76)	**0.03(12.23)**
	50	20	0.34	0.29(1.08)	0.34(1.02)	0.22(1.58)	0.05(6.89)	0.05(6.89)	0.05(6.89)	**0.03(12.64)**
	100	10	0.7	0.19(1.83)	0.59(1.19)	0.43(1.63)	0.1(6.97)	0.1(6.97)	0.2(3.49)	**0.07(10.3)**
svmguide1	30	35	1.84	1.76(1.09)	1.92(0.96)	1.07(1.72)	0.31(5.86)	0.37(4.96)	0.4(4.6)	**0.24(7.57)**
	50	28	3.05	0.88(2.08)	2.51(1.21)	1.57(1.94)	0.5(6.09)	0.75(4.06)	0.64(4.74)	**0.3(10.05)**
	100	45	4.72	1.46(2.09)	3.53(1.33)	2.31(2.04)	0.64(7.32)	1.13(4.16)	0.98(4.83)	**0.36(12.97)**
Codrna	30	99	17.63	6.97(1.92)	14.32(1.23)	8.88(1.98)	**2.2(8)**	3.77(4.68)	3.45(5.1)	3.15(5.59)
	50	71	27.61	6.24(2.83)	20.62(1.34)	13.8(2)	**3.31(8.34)**	7.59(3.64)	6.38(4.33)	3.42(8.08)
	100	102	50.08	11.34(2.44)	30.82(1.62)	19.15(2.62)	4.24(11.82)	11.42(4.38)	7.92(6.32)	**4.19(11.96)**
Kegg Network	30	78	42.77	49.9(2.83)	29.92(1.43)	20.85(2.05)	**8.56(4.99)**	9.73(4.4)	10.46(4.09)	12.21(3.5)
	50	77	64.1	17.37(2.46)	33.49(1.91)	23.77(2.7)	**8.43(7.6)**	11.52(5.56)	14.99(4.28)	11.77(5.44)
	100	64	122.58	22.17(2.89)	49.78(2.46)	35.67(3.44)	12.67(9.67)	19.94(6.15)	23.05(5.32)	**11.3(10.85)**
	300	39	343.45	37.51(3.27)	120.21(2.86)	92.01(3.73)	26.49(12.97)	61.05(5.63)	48.59(7.07)	**12.14(28.29)**
Epileptic	30	51	41.57	169.53(3.3)	43.62(0.95)	50.39(0.83)	18.78(2.21)	**12.47(3.33)**	13.73(3.03)	15.58(2.67)
	50	29	67.83	23.21(1.79)	82.17(0.83)	74.91(0.91)	46.1(1.47)	28.45(2.38)	32.97(2.06)	**17.38(3.9)**
	100	50	113.31	56.06(1.21)	106.75(1.06)	108.47(1.04)	59.12(1.92)	33.84(3.35)	39.78(2.85)	**26.37(4.3)**
	300	57	318.96	76.83(1.47)	289.75(1.1)	288.4(1.11)	149.39(2.14)	86.67(3.68)	104.12(3.06)	**47.33(6.74)**
	750	43	780.22	201.45(1.58)	874.18(0.89)	733.46(1.06)	239.07(3.26)	258(3.02)	316.44(2.47)	**98.31(7.94)**
Birch	30	26	20.46	694.44(1.51)	5.52(3.7)	4.03(5.08)	3.38(6.04)	2.35(8.72)	2.54(8.06)	**1.18(17.3)**
	50	62	29.79	5.39(3.79)	4.44(6.71)	2.64(11.29)	2.74(10.86)	1.92(15.52)	1.31(22.8)	**0.77(38.66)**
	100	79	55.93	4.22(7.05)	5.87(9.52)	4.52(12.38)	3.3(16.93)	3(18.64)	2.09(26.78)	**0.81(69.17)**
	300	123	160.66	6.85(8.16)	8.88(18.08)	14.19(11.32)	6.64(24.19)	6.37(25.24)	4.04(39.76)	**1.12(143.39)**
	750	113	399.54	21.19(7.58)	14.75(27.09)	40.07(9.97)	17.02(23.48)	16.32(24.48)	12.67(31.53)	**2.77(144.3)**
Ijcnn	30	44	80.22	84.4(6.37)	64.36(1.25)	47.93(1.67)	**8.8(9.12)**	14.84(5.41)	9.8(8.19)	9.82(8.17)
	50	68	112.43	31.35(2.56)	74.44(1.51)	55.04(2.04)	**10.49(10.72)**	24.47(4.59)	12.9(8.72)	14.94(7.52)
	100	88	210.08	32.91(3.42)	117.62(1.79)	94.09(2.23)	**14.06(14.95)**	54.2(3.88)	19.81(10.61)	21.84(9.62)
	300	88	591.96	56.76(3.7)	284.1(2.08)	241.98(2.45)	26.35(22.46)	197.08(3)	40.7(14.54)	**22.91(25.84)**
	750	83	1468.42	212.4(2.79)	647.72(2.27)	584.32(2.51)	62.41(23.53)	508.75(2.89)	87.01(16.88)	**25.97(56.55)**

TABLE 5
Distance calculations under large k

Data sets	k	iter	Lloyd	Hamerly	Dualtree_kd	Blacklist	Syin-ns	Ann	Exp-ns	Ball k-means
Four-class	300	5	1.E+06	6.E+05(2.01)	8.E+04(17.21)	3.E+05(4.55)	3.E+05(4.78)	7.E+05(1.89)	6.E+05(2.07)	**2.E+03(726.84)**
Svmguide1	300	28	6.E+07	3.E+07(2.33)	3.E+06(17.39)	8.E+06(7.56)	3.E+06(18.91)	1.E+07(4.7)	7.E+06(8.59)	**6.E+05(97.19)**
Codrna	300	142	3.E+09	8.E+08(3.27)	1.E+08(18.92)	3.E+08(9.81)	**5.E+07(54.65)**	4.E+08(5.96)	2.E+08(10.91)	9.E+07(26.92)
Kegg Network	500	44	1.E+09	4.E+08(3.74)	5.E+07(27.87)	1.E+08(12.53)	4.E+07(32.20)	2.E+08(8.75)	1.E+08(11.04)	**2.E+07(86.54)**
Epileptic	1000	38	4.E+08	3.E+08(1.54)	1.E+08(3.29)	3.E+08(1.52)	6.E+07(6.76)	8.E+07(5.17)	1.E+08(4.32)	**4.E+07(10.51)**
Birch	1000	131	1.E+10	2.E+09(7.03)	2.E+07(549.89)	7.E+07(176.04)	1.E+08(111.32)	3.E+08(39.22)	2.E+08(52.58)	**5.E+06(2478.51)**
Ijcnn	1000	84	1.E+10	5.E+09(2.31)	5.E+08(22.04)	2.E+09(6.19)	2.E+08(60.63)	3.E+09(4.2)	4.E+08(27.30)	**8.E+07(157.45)**

TABLE 6
Speedup over Lloyd and running time ((ms)/iteration)under large k

Data sets	k	iter	Lloyd	Hamerly	Dualtree_kd	Blacklist	Syin-ns	Ann	Exp-ns	Ball k-kmeans
Four-class	300	5	1.93	1.76(1.09)	1.25(1.54)	1.12(1.72)	0.6(3.21)	1(1.93)	1.6(1.2)	**0.32(6.09)**
svmguide1	300	28	13.38	6.97(1.92)	8.35(1.6)	6.37(2.1)	1.86(7.2)	4.14(3.23)	3.07(4.36)	**0.78(17.12)**
Codrna	300	142	141	49.9(2.83)	72.52(1.94)	49.18(2.87)	7.93(17.78)	32.52(4.34)	16.03(8.8)	**7.23(19.5)**
Kegg Network	500	44	559.5	169.53(3.3)	183.37(3.05)	137.28(4.08)	32.84(17.04)	93.98(5.95)	75.45(7.42)	**14.88(37.59)**
Epileptic	1000	38	1047.16	694.44(1.51)	1267.11(0.83)	1030.2(1.02)	292.55(3.58)	376.87(2.78)	457.68(2.29)	**122.85(8.52)**
Birch	1000	131	537.84	84.4(6.37)	17.13(31.39)	54.72(9.83)	20.46(26.29)	21.3(25.25)	17.5(30.73)	**4.34(123.91)**
Ijcnn	1000	84	1943.4	909.74(2.14)	808.44(2.4)	783.93(2.48)	78.24(24.84)	644.67(3.01)	109.15(17.8)	**33.04(58.83)**

dimensional datasets, as expected [14]. The performance of tree-based approaches suffers in high dimensions because the bounds become looser as the data dimension increases [7]. This fact resulted in the dualtree_kd and blacklist algorithms performing very well on low-dimensional datasets, but worst on those with relatively high dimensionality, such as Epileptic. However, all exact k-means algorithms suffer to some degree from the curse of dimensionality on extremely high-dimensional datasets, which lead to poor performance from exact k-means algorithms [20], [36]. In contrast to some approximate k-means algorithms, exact k-means algorithms are seldom directly used on high-dimensional datasets. Data distribution greatly affects the number of neighbors each ball cluster has in ball k-means, however, so for completeness we provide experimental results and analysis on high-dimensional datasets.

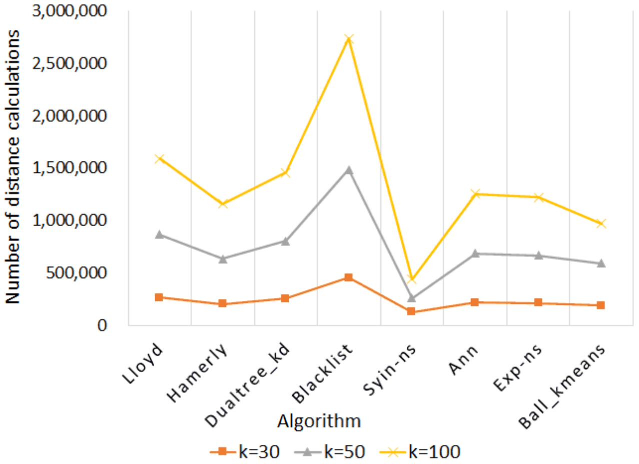


Fig. 4. The number of distance calculations for ball k-means and baselines on the high-dimensional dataset RNA-Seq (20531 dimensions).

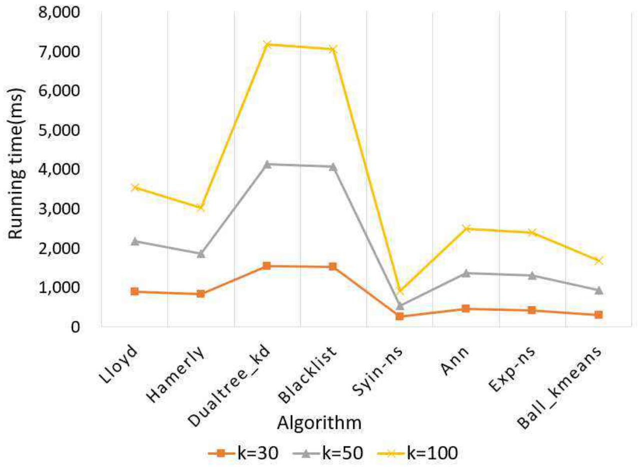


Fig. 5. Running time for ball k-means and baselines on the high-dimensional dataset RNA-Seq (20531 dimensions).

5.2 The Influence of Dimension

Figure 4 shows the number of distance calculations and Figure 5 shows the running time for the compared algorithms on the high-dimensional dataset RNA-Seq from the UCI Machine Learning Repository [3]. The dataset has a dimension of 20531. We observe that the number of distance calculations for Yinyang is smaller than that for ball k-means. High dimensionality can make the difference between the distance to the nearest and farthest points from a centroid small, resulting in a large number of neighbor ball clusters for a queried ball cluster and consequent increase in the number of distance calculations. We see in Figure 5 that the advantages in running time of ball k-means in comparison with other algorithms is also diminished. To investigate the impact of dimension on the ball k-means algorithm in more depth, we define the concept of the neighbor ball ratio as follows:

Definition 5. Let ball k-means be implemented on a given data set D. Let k, t, and K represent the number of clusters, the number of iterations, and the total number of neighbor balls in ball k-means, respectively. The *neighbor ball ratio* denotes the percentage of the average number of neighbor balls of each ball cluster in all of the ball clusters, expressed mathematically as $\frac{K}{tk^2}$.

Figure 6 shows the neighbor ball ratio over datasets with 2 to 2000 dimensions; we see that the ratio increases towards 1, resulting in a concomitant decrease in the efficiency of ball k-means. We also point out the interesting interaction between dimension and k value: the influence of dimension is moderated by the increase in k value. When k=100, the neighbor ball ratio does not exceed 50% for all dimensions. This result also demonstrates the advantages of ball k-means in large-k problems, consistent with earlier results.

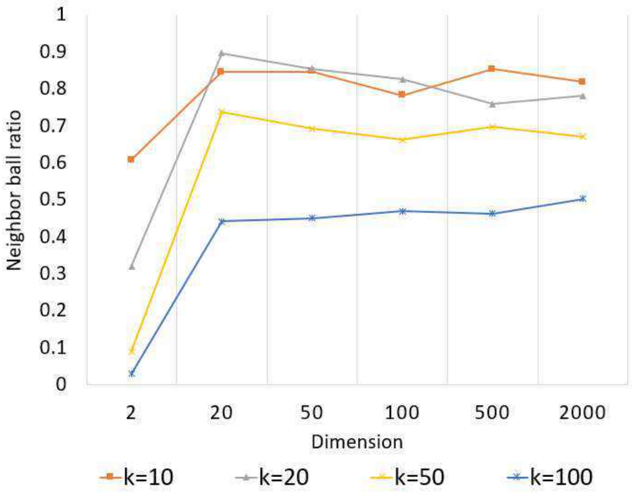
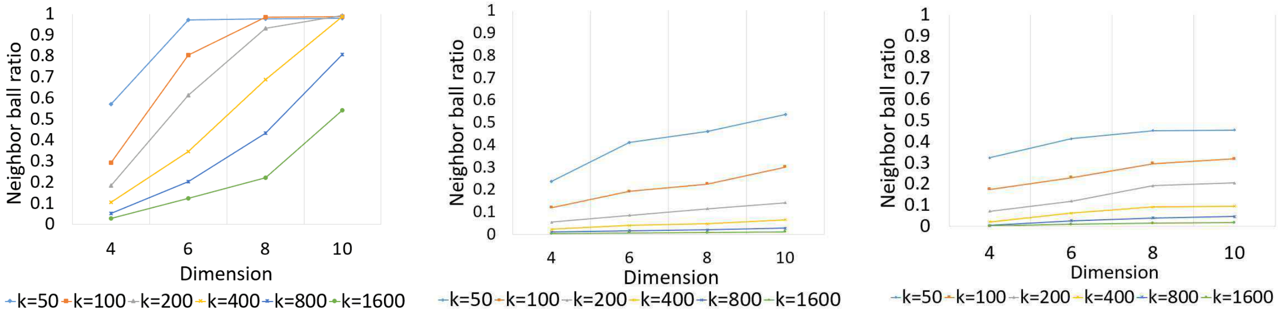


Fig. 6. The influence of dimension on the neighbor ball ratio. The data sets were generated from RNA-Seq by selecting the specified number of random dimensions.

5.3 The Influence of Uniform Distribution

The problems of high dimensionality extend to uniform distribution—indeed, the curse of dimensionality is the uniform nature of the distances obtained in Euclidean space. The high number of neighbors for each ball cluster present in high dimensional space may also be present in uniform space, and for the same reason [30]. We thus also show the behavior of the neighbor ball ratio on uniform data (the size of the uniform data is 200 thousand). Figure 7(a) shows the neighbor ball ratio on an artificially generated perfectly



(a) (b) (c)

Fig. 7. The influence of data distribution on the neighbor ball ratio. (a) The data under uniform distribution (artifical dataset). (b) A cut of the UCI MLR dataset Epileptic. (c) A cut of the UCI MLR dataset kegg.

uniform dataset. The maximum dimension is only a very small number, 10, but already the neighbor ball ratio is close to 1, except for extremely high k values. This indicates that the number of neighbor ball clusters is close to the k value for reasonable k values, and therefore ball k-means is approaching Lloyd k-means and cannot accelerate it very much. In contrast, in Figure 7(b) and Figure 7(c), the data is non-uniformly distributed and the neighbor ball ratio is always smaller than 0.5, even for low k. This indicates that ball k-means will perform much better on non-uniform datasets than those that are uniformly distributed. Ball k-means is an exact k-means algorithm, so it cannot overcome this short-coming. It is better suited for data with spherical or convex internal structure.

6 Conclusions and Future Work

We have presented ball k-means, a simple and fast method of k-means clustering. Ball k-means is exact, has no bounds, and accelerates the standard k-means algorithm through searching neighbor clusters and partitioning queried clusters into stable and active areas. The algorithm is simple and easy to implement. Experimental results show that ball k-means outperforms its competitors on most datasets and for most k values.

Although ball k-means consistently outstrips its competitor algorithms in most cases, there is still interesting work ahead. By design, the algorithm is a good candidate for parallelization. This may help mitigate performance on some datasets with high numbers of distance computations. Another line of inquiry is to investigate those methods used in other clustering algorithms that perform well on high-dimensional datasets, and to combine them with ball k-means to improve performance. While already fast, we want to push ball k-means to the limit of possibility.

7 Acknowledgments

The authors greatly thank the handling associate editor and all anonymous reviewers for their valuable comments. This work was supported in part by the National Natural Science Foundation of China under Grant Nos. 61806030 and 61936001, the Natural Science Foundation of Chongqing Nos. cstc2019jcyj-msxmX0485 and cstc2019jcyj-cxttX0002, the National Key Research and Development Program of China under Grant Nos. 2019QY(Y)0301 and 2016QY01W0200, and NICE: NRT for Integrated Computational Entomology, US NSF award 1631776.

References

[1] Daniel Aloise, Amit Deshpande, Pierre Hansen, and Preyas Popat. Np-hardness of euclidean sum-of-squares clustering. volume 75, pages 245–248. Springer, 2009.

[2] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.

[3] K. Bache and M. Lichman. Uci machine learning repository. *UCI Machine Learning Repository University of California, Irvine, School of Information and Computer Sciences*, 12 2013.

[4] Olivier Bachem, Mario Lucic, S Hamed Hassani, and Andreas Krause. Approximate k-means++ in sublinear time. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[5] Olivier Bachem, Mario Lucic, Hamed Hassani, and Andreas Krause. Fast and provably good seedings for k-means. In *Advances in Neural Information Processing Systems*, pages 55–63, 2016.

[6] Paul S Bradley and Usama M Fayyad. Refining initial points for k-means clustering. In *ICML*, volume 98, pages 91–99. Citeseer, 1998.

[7] Ryan R. Curtin. A dual-tree algorithm for fast k-means clustering with large k. In *Proceedings of the 2017 SIAM International Conference on Data Mining*, pages 300–308. SIAM, 2017.

[8] Cheng-Hao Deng and Wan-Lei Zhao. Fast k-means based on k-nn graph. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 1220–1223. IEEE, 2018.

[9] Yufei Ding, Yue Zhao, Xipeng Shen, Madanlal Musuvathi, and Todd Mytkowicz. Yinyang k-means: A drop-in replacement of the classic k-means with consistent speedup. In *International Conference on Machine Learning*, pages 579–587, 2015.

[10] Drake, Jonathan. Faster k-means clustering. , 2013.

[11] Charles Elkan. Using the triangle inequality to accelerate k-means. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 147–153, 2003.

[12] A.M. Fahim, A.M. Salem, F.A. Torkey, and M.A. Ramadan. An efficient enhanced k-means clustering algorithm. *Journal of Zhejiang University-Science A*, 7(10):1626–1633, 2006.

[13] Pasi Fränti and Sami Sieranoja. K-means properties on six clustering benchmark datasets. In *Applied Intelligence*, 48 (12), pages 4743-4759, December 2018. http://cs.uef.fi/sipu/datasets/

[14] Greg Hamerly. Making k-means even faster. In *Proceedings of the 2010 SIAM international conference on data mining*, pages 130–140. SIAM, 2010.

[15] Greg Hamerly and Jonathan Drake. Accelerating Lloyd's algorithm for k-means clustering. In *Partitional clustering algorithms*, pages 41–78. Springer, 2015.

[16] Qinghao Hu, Jiaxiang Wu, Lu Bai, Yifan Zhang, and Jian Cheng. Fast k-means for large scale clustering. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 2099–2102. ACM, 2017.

[17] RC Jancey. Multidimensional group analysis. *Australian Journal of Botany*, 14(1):127–130, 1966.

[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 In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24 (7) 2002, pp 881-892.

[19] Siddhesh Khandelwal and Amit Awekar. Faster k-means cluster estimation. In *European Conference on Information Retrieval*, pages 520–526. Springer, 2017.

[20] Hans-Peter Kriegel, Matthias Schubert, and Arthur Zimek. Angle-based outlier detection in high-dimensional data. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 444–452. ACM, 2008.

[21] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[22] James Newling and François Fleuret. Fast k-means with accurate bounds. In *International Conference on Machine Learning*, pages 936–944, 2016.

[23] James Newling and François Fleuret. K-medoids for k-means seeding. In *Advances in Neural Information Processing Systems*, pages 5195–5203, 2017.

[24] Raymond T. Ng and Jiawei Han. Efficient and effective clustering methods for spatial data mining. In *Vldb Conference*, 1994.

[25] David Nister and Henrik Stewenius. Scalable recognition with a vocabulary tree. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 2161–2168. Ieee, 2006.

[26] Joaqun Pérez-Ortega, Nelva Nely Almanza-Ortega, Andrea Vega-Villalobos, Rodolfo Pazos-Rangel, CrispínZavala-Díaz and Alicia Martínez-Rebollar. The K-Means Algorithm Evolution. 2019.

[27] Dan Pelleg and Andrew Moore. Accelerating exact k-means algorithms with geometric reasoning. In *Proceedings of the fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 277–281. ACM, 1999.

[28] Joaquín Pérez Ortega, Nelva Nely Almanza Ortega, Jorge A Ruiz-Vanoye, Socorro Sáenz Sánchez, José María Rodríguez Lelis, Alicia Martínez Rebollar, et al. A-means: improving the cluster assignment phase of k-means for big data. *International Journal of Combinatorial Optimization Problems and Informatics*, 9(2):3–10, 2018.

[29] Joaquín Pérez, Carlos Eduardo Pires, Leandro Balby, Adriana Mexicano, and Miguel Hidalgo. Early classification: a new heuristic to improve the classification step of k-means. *Journal of Information and Data Management*, 4(2):94–103, 2013.

[30] Florian Pfender and Günter M. Ziegler Kissing numbers, sphere packings, and some unexpected proofs. In *Notices of the AMS* 51 (8) (2004) pp. 873883.

[31] James Philbin. Scalable object retrieval in very large image collections. *Annales De Zoologie Ecologie Animale*, 93(3):E70–E70, 2010.

[32] James Philbin, Ondrej Chum, Michael Isard, Josef Sivic, and Andrew Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *IEEE Conference on Computer Vision & Pattern Recognition*, 2007.

[33] Petr Ryšavỳ and Greg Hamerly. Geometric methods to accelerate k-means algorithms. In *Proceedings of the 2016 SIAM International Conference on Data Mining*, pages 324–332. SIAM, 2016.

[34] David Sculley. Web-scale k-means clustering. In *Proceedings of the 19th international conference on World wide web*, pages 1177–1178. ACM, 2010.

[35] Chun-Wei Tsai, Chu-Sing Yang, and Ming-Chao Chiang. A time efficient pattern reduction algorithm for k-means based clustering. In *2007 IEEE International Conference on Systems, Man and Cybernetics*, pages 504–509. IEEE, 2007.

[36] Michel Verleysen and Damien François. The curse of dimensionality in data mining and time series prediction. In *International Work-Conference on Artificial Neural Networks*, pages 758–770. Springer, 2005.

[37] Jingdong Wang, Wang Jing, Qifa Ke, Zeng Gang, and Shipeng Li. Fast approximate k-means via cluster closures. 2012.

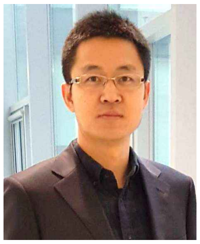
[38] Xindong Wu, Vipin Kumar, J Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J McLachlan, Angus Ng, Bing Liu, S Yu Philip, et al. Top 10 algorithms in data mining. *Knowledge and information systems*, 14(1):1–37, 2008.

[39] Moore, Andrew. The Anchors Hierachy: Using the triangle inequality to survive high dimensional data. *arXiv preprint arXiv:1301.3877*, 2013.

Shuyin Xia received his B.S. degree in 2008 and his M.S. degree in 2012, both in computer science and both from the Chongqing University of Technology in China. He received his Ph.D. degree from the College of Computer Science at Chongqing University in China. He is an associate professor and a Ph.D. supervisor at the Chongqing University of Posts and Telecommunications in Chongqing, China. His research interests include classifiers and granular computing. He has published more than 30 papers in prestigious journals and conferences, such as IEEE-TKDE, and IS. He is the executive deputy director of the Big Data and Network Security Joint Lab of CQUPT and an IEEE Member.

Daowan Peng received his B.S. degree in Computer science from the Chongqing University of Posts and Telecommunications in China. He is currently a student at the College of Computer Science and Technology at the Chongqing University of Posts and Telecommunications. His research interests include machine learning and data mining.

Deyu Meng received his B.Sc., M.Sc., and Ph.D. degrees from Xi'an Jiaotong University in Xi'an, China, where he is currently a Professor with the Institute for Information and System Sciences. His current research interest include self-paced learning, noise modeling, and tensor sparsity. He was a Visiting Scholar at Carnegie Mellon University in Pittsburgh, PA, USA. He is a regular Reviewer of ICML, NIPS, CVPR, ICCV, ECCV, AAAI, IJCAI, ACM MM, AISTATS, ACCV, ACM-L, ICPR, and IEEE T-PAMI, IEEE T-IP, IEEE T-NNLS and others.



Changqing Zhang received his B.S. and M.S. degrees in computer science from Sichuan University, and his Ph.D. degree in computer science from Tianjin University of China. He is an Assistant Professor at the School of Artificial Intelligence in the College of Intelligence and Computing at Tianjin University of China. He has been a Research Fellow at the University of North Carolina at Chapel Hill (UNC-CH). His research focuses on machine learning, computer vision and medical image analysis. He has published over 50 papers in prestigious journals and conferences, including IEEE T-PAMI, T-IP, T-NNLS, T-MI, NIPS, CVPR, ICCV, AAAI, IJCAI, and ICDM.

Elisabeth Giem received two bachelor's degrees, one in pure mathematics and one in music (concentration in performance) from the University of California, Riverside (UCR). She received a master's degree in computational and applied mathematics from Rice University, and a master's degree in pure mathematics from UCR. She joined Zizhong Chen's SuperLab in 2018 as a computer science Ph.D. student, and has been awarded the NSF NRT In Computational Entomology Fellowship. Her research interests include but are not limited to high-performance computing, parallel and distributed systems, big data analytics, computational entomology, and numerical linear algebra algorithms and software.

Guoyin Wang received a B.E. degree in computer software in 1992, a M.S. degree in computer software in 1994, and a Ph.D. degree in computer organization and architecture in 1996, all from Xi'an Jiaotong University in Xi'an, China. His research interests include data mining, machine learning, rough sets, granular computing, cognitive computing, and so forth. He has published over 300 papers in prestigious journals and conferences, including IEEE TKDE, T-IP, T-NNLS, and TCYB. He has worked at the University of North Texas, USA, and the University of Regina, Canada, as a Visiting Scholar. Since 1996, he has been working at the Chongqing University of Posts and Telecommunications in Chongqing, China, where he is currently a Professor and a Ph.D. supervisor, the Director of the Chongqing Key Laboratory of Computational Intelligence, and the Dean of the Graduate School. He is the Steering Committee Chair of the International Rough Set Society (IRSS), a Vice-President of the Chinese Association for Artificial Intelligence (CAAI), and a council member of the China Computer Federation (CCF).

Zizhong Chen received his bachelor's degree in mathematics from Beijing Normal University, master's degree in economics from Renmin University of China, and Ph.D. degree in computer science from the University of Tennessee, Knoxville. He is a professor of computer science at the University of California, Riverside. His research interests include high performance computing, parallel and distributed systems, big data analytics, cluster and cloud computing, algorithm-based fault tolerance, power and energy efficient computing, numerical algorithms and software, and large-scale computer simulations. His research has been supported by the US National Science Foundation, US Department of Energy, Nvidia, and Microsoft Corporation. He received a CAREER Award from the US National Science Foundation and Best Paper Awards from the International Supercomputing Conference and IEEE International Conference on Cluster Computing. He is a senior member of the IEEE and a life member of the ACM. He currently serves as a subject area editor for Elsevier Parallel Computing journal and an associate editor for *IEEE Transactions on Parallel and Distributed Systems*.

Wei Wei received his M.S. and Ph.D. degrees from Xi'an Jiaotong University. He is an Associate Professor with the School of Computer Science and Engineering at the Xi'an University of Technology in Xi'an, China, where he has served as a Principal Investigator and as a Technical Member for many grant funds. His research interests include wireless networks, wireless sensor network applications, image processing, mobile computing, distributed computing, pervasive computing, the Internet of Things, and sensor data clouds. He has published around 100 research papers at international conferences and in international journals. He is an Editorial Board Member of FGCS, AHSWN, IEICE, and KSII. He is also a TPC Member of many conferences and a regular Reviewer of the IEEE TPDS, the IEEE TIP, the IEEE TMC, the IEEE TWC, and many other Elsevier journals.