Yinyang K-Means: A Drop-In Replacement of the Classic K-Means with Consistent Speedup

Yufei Ding * YDING8@NCSU.EDU
Yue Zhao * YZHAO30@NCSU.EDU
Xipeng Shen * XSHEN5@NCSU.EDU
Madanlal Musuvathi ◇ MADANM@MICROSOFT.COM
Todd Mytkowicz ◇ TODDM@MICROSOFT.COM

Department of Computer Science, North Carolina State University *
Microsoft Research ◇

Abstract

This paper presents Yinyang K-means, a new algorithm for K-means clustering. By clustering the centers in the initial stage, and leveraging efficiently maintained lower and upper bounds between a point and centers, it more effectively avoids unnecessary distance calculations than prior algorithms. It significantly outperforms prior K-means algorithms consistently across all experimented data sets, cluster numbers, and machine configurations. The consistent, superior performance—plus its simplicity, user-control of overheads, and guarantee in producing the same clustering results as the standard K-means does—makes Yinyang K-means a drop-in replacement of the classic K-means with an order of magnitude higher performance.

1. Introduction

The classic K-means algorithm (Lloyd's algorithm) consists of two steps. For an input of n data points of d dimensions and k initial cluster centers, the *assignment step* assigns each point to its closest cluster, and the *update step* updates each of the k cluster centers with the centroid of the points assigned to that cluster. The algorithm repeats until all the cluster centers remain unchanged in an iteration.

For its simplicity and general applicability, the algorithm is the most widely used partitional clustering algorithm in practice, and is identified as one of the top 10 data mining algorithms (Wu et al., 2008). However, when n, k,or d is large, the algorithm runs slow due to its linear dependence on n, k,and d in the two steps.

There have been a number of efforts trying to improve its speed. Some try to come up with better initial centers (e.g., K-means++ (Arthur & Vassilvitskii, 2007; Bahmani et al., 2012)) or parallel implementations (Zhao et al., 2009). A complementary approach is to speed up the algorithm itself, which is the focus of this paper.

Prior efforts in this direction include approximation (Czumaj & Sohler, 2007; Sculley, 2010; Philbin et al., 2007; Guha et al., 1998; Wang et al., 2012), structural optimization (Pelleg & Moore, 1999; Kanungo et al., 2002), and incremental optimization (Elkan, 2003; Drake & Hamerly, 2012; Hamerly, 2010). They each have made significant contributions. However, the classic Lloyd's algorithm still remains the dominant choice in practice, exemplified by the implementations in popular libraries, such as GraphLab (Low et al., 2010), OpenCV (OpenCV), mlpack (Curtin et al., 2013), and so on. They offer some seeding options, but are primarily based on the Lloyd's algorithm.

For an alternative algorithm to get widely accepted, we believe that it needs to meet several requirements: (1) It must inherit the level of trust that the Lloyd's algorithm has attained through the many decades of practical use; (2) it must produce significant speedups *consistently*; (3) it must be simple to develop and deploy.

The previous proposals unfortunately fall short in at least one of the three requirements. The approximation methods (Czumaj & Sohler, 2007; Sculley, 2010; Philbin et al., 2007; Guha et al., 1998; Wang et al., 2012), for instance, produce clustering results different from the results of the standard K-means. It is possible that their outputs could be good enough (or even better) for some usage, but in general such a confidence is yet to be established, in both theory and practice: Users still face the uncertainty on whether

Proceedings of the 32^{nd} International Conference on Machine Learning, Lille, France, 2015. JMLR: W&CP volume 37.

the output from these algorithms is good enough on an arbitrary dataset for a real usage, for which, the Lloyd's algorithm has been practically proven to work. Some other prior efforts try to speed up K-means while keeping its semantics. They inherit the level of trust that the standard K-means has gained through decades of usage. They however have failed to show consistent speedups. The KD-tree–based methods (Pelleg & Moore, 1999; Kanungo et al., 2002) for instance does not work well when the number of dimensions is greater than 20 (Kanungo et al., 2002), while the prior triangular inequality–based methods (Elkan, 2003; Drake & Hamerly, 2012; Hamerly, 2010) either does not scale with the number of clusters or perform inferiorly in some scenarios (detailed in Section 5).

This paper introduces Yinyang K-means, an enhanced K-means that meets all the conditions. The key is in its careful but efficient maintenance of the upper bound of the distance from one point to its assigned cluster center, and the lower bound of the distance from the point to other cluster centers. The interplay between the two kinds of bounds forms a two-level filter, through which, Yinyang K-means avoids unnecessary distance calculations effectively. Yinyang K-means features a space-conscious elastic design to adaptively tap into the maximal power of the filters under various space constraints. The name of the method is inspired by the ancient Chinese philosophy, in which, yin and yang are concepts used to describe how apparently contrary forces work complementarily to form a harmony. The carefully maintained lower bound and upper bound in Yinyang K-means are respectively the yin and yang of the distance filter. Their continuous, efficient evolvement and interplay form the key for Yinyang K-means to work effectively.

Experiments on a spectrum of problem settings and machines show that Yinyang K-means excels in all the cases, consistently outperforming the classic K-means by an order of magnitude and the fastest prior known K-means algorithms by more than three times on average. Its simplicity, elasticity (ability for a user to control space overheads), semantics preservation, and consistent superior performance make it a drop-in replacement of the standard K-means.

2. Yinyang K-means

This section presents the Yinyang K-means algorithm and describes the optimizations to both the assignment and the update steps.

2.1. Optimizing the *Assignment* Step

In the standard K-means, the *assignment* step computes the distances between every point and every cluster center in order to find out the closest center to each point. The key idea behind our optimized *assignment* step is the use of two filters to detect which distance calculations are unnecessary and avoid doing them for speedup. These optimizations are based on the triangle inequality.

Triangle Inequality: Let $d(a, b)$ represent the distance between a and b in some metric, such as the Euclidean metric. The triangular inequality states that $d(a, c) \leq d(a, b) + d(b, c)$. In the context of K-means, given a point x and two cluster centers b and c, the triangular inequality gives a way to bound the (unknown) distance between x and c given the distance between x and b and the distance between b and c:

$$|d(x, b) - d(b, c)| \leq d(x, c) \leq d(x, b) + d(b, c)$$

In particular, if b and c represent centers of the same cluster in two consecutive iterations, the bounds above can be used to approximate $d(x, c)$ as shown below.

Triangle inequality has been used for optimizing K-means before (Elkan, 2003; Hamerly, 2010; Drake & Hamerly, 2012). Our design features an innovative way of applying it to the carefully maintained lower and upper bounds of distances, which is key to the consistent speedups that prior solutions fail to provide.

We first introduce some notations for the following detailed discussion. Let C be the set of cluster centers and c be one cluster in the set. For a given point x, let $b(x)$ (for "best of x") be the cluster to which the point is assigned to. Let C', c', and $b'(x)$ represent the corresponding entities in the next iteration respectively. Let $\delta(c)$ represent $d(c, c')$—that is, the shift of cluster center due to the center update.

We next describe *global filter*, a special case of the two filters that we have designed for Yinyang K-means. Its simplicity helps ease our later explanation of the two filters.

2.1.1. Global Filtering

Global filtering identifies whether a point x changes its cluster in an assignment step with a single comparison. For each point x, the algorithm maintains an upper bound $ub(x) \geq d(x, b(x))$ and a global lower bound $lb(x) \leq d(x, c),\ \forall c \in C - b(x)$. One way to initialize these bounds is to use the distance to the best cluster center as the upper bound and the distance to the second-closest cluster center as the lower bound.

Lemma 1 (Global-Filtering Condition)**.** *A point x in the cluster $b = b(x)$ does not change its cluster after a center update if*

$$lb(x) - \max_{c \in C} \delta(c) \geq ub(x) + \delta(b)$$

Proof. Consider a cluster $c \in C - b$. Let c' be its new cluster center after a center update. Let b' be the new cluster

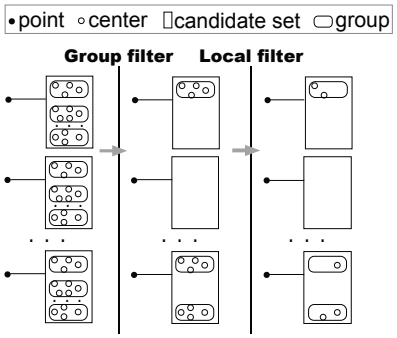

Figure 1. Two filters for optimizing the *assignment* step in Yinyang K-means. A rectangle associates with each point to be clustered, representing the set of centers that are possible to replace the current cluster center of the point as its new cluster center. The centers are put into groups and go through the group filter. The local filter examines each center in the remaining groups to further avoid unnecessary distance calculations.

center of the cluster b. The proof follows from the fact that the r.h.s above is a new upper bound on $d(x, b')$ and the l.h.s is a new lower bound on $d(x, c')$ for all other clusters c'.

By triangle inequality, we have $d(x, c') \geq d(x, c) - d(c, c') = d(x, c) - \delta(c) \geq d(x, c) - \max_{c \in C} \delta(c)$. Similarly, $d(x, b') \leq d(x, b) + \delta(b) \leq ub(x) + \delta(b)$. Thus $d(x, b') \leq d(x, c')$. □

Essentially, Lemma 1 states that it is unnecessary to change the cluster assignment of a point unless the cluster centers drift drastically; the distance calculations related with that point can be hence avoided. Applying this lemma requires computing the δs for each cluster at the end of each iteration, which requires $O(k * d)$ time and $O(k)$ space. In addition, maintaining an upper and lower bounds for each point requires $O(n)$ space.

One challenge in applying the lemma, of course, is to efficiently maintain the upper and lower bounds across iterations. The proof of the lemma already suggests a way to do so: $lb'(x) = lb(x) - \max_{c \in C} \delta(c)$ and $ub'(x) = ub(x) + \delta(b)$. This update requires no need to compute the distances between any point and any center. It is employed in the algorithm.

Although the global filtering can reduce many distance calculations in some cases, its effectiveness gets largely throttled in the presence of *big-movers* (i.e., cluster centers that drift dramatically in a center update.) Because of its use of the largest drift in the update of the lower-bound, even a single big-mover reduces the lower-bound for all points substantially, making the global-filtering ineffective.

2.1.2. Group Filtering

Group filtering is a generalization of the global filtering that addresses its shortcoming through an elastic design.

Group filtering first groups the k clusters into t groups $G = \{G_1, G_2, ..., G_t\}$, where each $G_i \in G$ is a set of the clusters. This grouping is done once before the beginning of the first iteration of K-means (elaborated in Section 3). It then applies the global filtering condition to each group. Specifically, for each group (e.g., G_i) it keeps a group lower bound $lb(x, G_i) \leq d(x, c),\ \forall c \in G_i - b(x)$ for each point x. Similar to global-filtering, $lb(x, G_i)$ is initialized with the distance to the closest cluster in G_i other than $b(x)$ and updated by $lb'(x, G_i) = lb(x, G_i) - \max_{c \in G_i} \delta(c)$. If $lb'(x, G_i) \geq ub'(x)$, where $ub'(x)$ is the new upper bound computed in the global-filtering optimization, then a variant of Lemma 1 shows that x is not assigned to any of the clusters in G_i. If G_i has no big-movers in an iteration, then all its clusters could be filtered in the assignment step. In Figure 1, the first point (on top) has only one group left—no need to compute the distances from that point to any center in other groups. The second point has none left, showing that its assignment won't change in this iteration and hence no need to calculate its distance to any center.

The parameter t provides a design knob for controlling the space overhead and redundant distance elimination. Group filtering reduces to global filtering when t is set to 1. When t increases, the filter uses more space for more lower bounds, and spends more time on maintaining the lower bounds, but meanwhile limits the effects of big movers more and hence avoids more distance calculations. Our experiments in Section 5 show that when t is around $k/10$, the method gives the best overall performance across datasets; at a smaller value, the performance is lower but still significantly higher than the standard K-means and its prior alternatives. Our design further considers the amount of available memory: t is set to $k/10$ if space allows; o.w., the largest possible value is used. This space-conscious elastic design helps tap into the benefits of Yinyang K-means under various space pressure as Section 5 will show.

There are various ways to group the k clusters into t groups. Our investigation shows that clustering on the initial centers is a good choice. Compared to random grouping, it benefits more from the locality of the centers and thus, yields better performance. The grouping is a one-time job, only needed at the beginning of the first iteration of K-means. Regrouping, while feasible, did not help as observed in our experiments.

The group filtering can be easily combined with the global filtering. The algorithm first compares the smallest of all lower bounds (i.e., global lower bound) with the upper bound before examining the lower bound of each group.

If the global lower bound is greater than the upper bound, no reassignment is needed for that point and all the group-level comparisons can be avoided. Section 3 provides the details.

2.1.3. Local Filtering

If a group of cluster centers go through the group filter, one of the centers could be the new best center for the data point of interest. Rather than computing the distances from that point to each of those centers, we design a local filter to further avoid unnecessary distance calculations.

Lemma 2 (Local-Filtering Condition)**.** *A center $c' \in G'_i$ cannot be the closest center to a point x if there is a center $p' \neq c'$ (p' does not have to be part of G'_i) such that*

$$d(x, p') < lb(x, G_i) - \delta(c).$$

Proof. This lemma follows from the triangle inequality. $d(x, c') \geq d(x, c) - d(c, c') \geq lb(x, G_i) - \delta(c) > d(x, p')$. Thus, the point p' is closer to x than c' is. □

The lemma allows us to skip the distance calculations for centers that meet the condition. When a center goes through the local filter, our algorithm computes its distance to the point x. The smallest distance of all the centers in G'_i will then be used to update the group lower bound $lb(x, G'_i)$.

When applying the local filter, the selection of p' has some subtlety. It is tempting to use the so-far-found closest center as p' since it can help detect as many infeasible candidate centers as possible. However, our experiments found that using the so-far-found second closest center as p' consistently gives better overall speed of Yinyang K-means (up to 1.6X speedup). The reason is that it allows the computation of the exact lower bound (i.e., the distance to the second closest center), which makes the group filter more effective in the next iteration.

It is worth noting that the local filter requires no extra lower bounds than what the group filter maintains, and hence adds no extra space overhead.

2.2. Optimizing the *Center Update* Step

The *update* step computes the new center for each cluster. With the *assignment* step gets optimized, this step starts to weigh substantially, but no prior work has optimized it. We enhance it also by leveraging the fact that only some points change their clusters across iterations. Rather than averaging across all points in a cluster, it avoids some points by reusing the old centers as follows:

$$c' = (c * |V| - (\sum_{y \in V - OV} y) + \sum_{y' \in V' - OV} y')/|V'|, \quad (1)$$

where, V' and V represent a cluster in this and the previous iteration, OV is $V \cap V'$, c and c' are the old and new centers of the cluster. All variables on the righthand side of the formula are just side product of the optimized *assignment* step.

This new update algorithm involves fewer computations than the default update if and only if less than half points have changed their clusters. An implementation can easily check this condition in each iteration and use the new algorithm when it holds. In our experiments on real data sets, we have never seen an violation of the condition.

3. Algorithm

Putting the group filter, local filter, and new center update algorithm together, we get the complete Yinyang K-means as follows.

Step 1: Set t to a value no greater than $k/10$ and meeting the space constraint. Group the initial centers into t groups, $\{G_i | i = 1, 2, \cdots, t\}$ by running K-means on just those initial for five iterations to produce reasonable groups while incurring little overhead.

Step 2: Run the standard K-means on the points for the first iteration. For each point x, set the upper bound $ub(x) = d(x, b(x))$ and the lower bounds $lb(x, G_i)$ as the shortest distance between x and all centers in G_i excluding $b(x)$.

Step 3: Repeat until convergence:

3.1: Update centers by Equation 1, compute drift of each center $\delta(c)$, and record the maximum drift for each group $\delta(G_i)$.

3.2 Group filtering: For each point x, update the upper bound $ub(x)$ and the group lower bounds $lb(x, G_i)$ with $ub(x) + \delta(b(x))$ and $lb(x, G_i) - \delta(G_i)$ respectively. Assign the temporary global lower bound as $lb(x) = \min_{i=1}^{t} lb(x, G_i)$. If $lb(x) \geq ub(x)$, assign $b'(x)$ with $b(x)$. Otherwise, tighten $ub(x) = d(x, b(x))$ and check the condition again. If it fails, find groups for which $lb(x, G_i) < ub(x)$, and pass x and these groups to local filtering.

3.3 Local filtering: For each remaining point x, filter its remaining candidate centers with the so-far-found second closest center, compute the distances from x to the centers that go through the filter to find out the new $b(x)$, and update the group lower bound $lb(x, G_i)$ with the distance to the second closest center. For groups blocked by

the group filter, update the lower bounds $lb(x, G_i)$ with $lb(x, G_i) - \delta(G_i)$. Update $ub(x)$ with $d(x, b(x))$.

4. Comparison

The work closest to ours includes the K-means optimized by Elkan (Elkan, 2003) and by Drake and Hamerly (Drake & Hamerly, 2012). They also use triangle inequality to avoid some distance calculations, but differ from our algorithm in some critical aspects. Compared to Elkan, our algorithm shows great advantages in the efficiency of the non-local filter (group/global filter). Figure 2 shows that intuitively. Figures 2 (a) and (b) depict the Voronoi diagrams[1] in two consecutive iterations of K-means. It is easy to see that if a point (e.g., the "x") is in the grey area in Figure 2 (b), its cluster assignment needs no update in iteration j+1. Elkan's algorithm tries to approximate the overlapped Voronoi areas with spheres as the disk in Figure 2 (c) shows. The radius of the sphere is half of the shortest distance from the center to all other centers. In contrast, the lower and upper bounds maintained at each point make Yinyang K-means approximate the overlapped areas much better, and hence more effective in avoiding unnecessary distance calculations. Our experiments show that the group filter in Yinyang K-means helps avoid at least two times (over 6X in most cases) more distance calculations than the non-local filter in Elkan's algorithm as shown in Table 4.

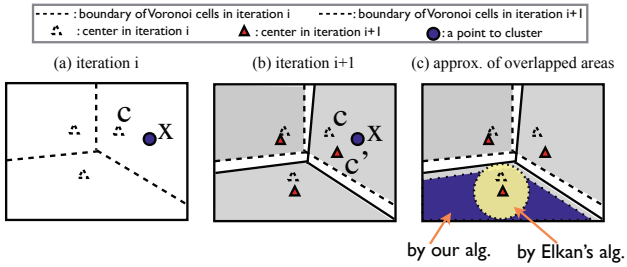

Figure 2. The Voronoi diagrams in two consecutive iterations of k-means, and the approximation of the overlapped areas by our algorithm ($t = 1$) and Elkan's algorithm.

Elkan's algorithm mitigate the inefficiency through a local filter, which however requires k lower bounds for each point and a series of condition checks, entailing large space and time cost. As Table 1 shows, Elkan's algorithm takes $O(n * k)$ space and time to maintain lower bounds, while Yinyang K-means takes only $O(n * t)$. The non-local filtering time cost is $O(k^2 * d + n)$ for Elkan's algorithm, and $O(n) \sim O(n * t)$ for Yinyang K-means, depending on whether the global filter works. The two methods have a similar local filtering time complexity, $O(n * \alpha * k)$, with α for the fraction of points passing through the non-local filter. However, as shown in Table 4, α is much smaller in Yinyang K-means than in Elkan's algorithm (0.2 vs. 0.86 on average), thanks to the more powerful non-local filter of Yinyang K-means. The cost causes Elkan's algorithm to fail or perform inferiorly in some scenarios, as shown in the next section.

Drake's algorithm tries to lower the time and space cost of the local filter of Elkan's algorithm. For each point, it maintains t lower bounds, with the first $(t-1)$ for the distance from the point to each of its $(t-1)$ closest centers, and the tth for all other centers. As Table 1 shows, its time and space costs are lower than Elkan's algorithm (but still higher than Yinyang K-means). It is however still sensitive to "big movers", because the update of the tth lower bound uses the maximal drift, and the impact propagates to other lower bounds due to the order of lower bounds that the algorithm needs to maintain.

In comparison, the grouping-based filter design of Yinyang K-means mitigates the sensitivity to "big movers". It, along with the elasticity, helps Yinyang K-means excel over the prior algorithms. In addition, Yinyang K-means is the only algorithm that optimizes not only the *assignment* step but also the center *update* step of K-means. We provide quantitative comparisons next.

5. Experiments

To demonstrate the efficacy of Yinyang K-means, we evaluate our approach on a variety of large, real world data sets and compare it with three other methods: the fastest prior known K-means algorithm (Elkan (Elkan, 2003)), Drake (Drake & Hamerly, 2012) and standard K-means. All three algorithms are implemented in Graphlab (Low et al., 2010) and can run in parallel. We run all three algorithms on the same data set with the same randomly selected initial center seeds, and thus all algorithms converge to the same clustering result after the same number of iterations.

We use eight real world large data sets, four of which are taken from the UCI machine learning repository (Bache & Lichman, 2013), while the other four are commonly used image data sets(Wang et al., 2012; 2013). Their size and dimension are shown in the leftmost columns in Table 2 (n for number of points, d for dimensions, k for number of clusters). We experiment with two machines, one with $16GB$ memory and the other with $4GB$ memory, detailed in Tables 2 and 3.

Consistent Speedup on the *Assignment* Step The experiments demonstrate that Yinyang K-means provides *consistent* speedup over both standard K-means, Elkan's and Drake's algorithm. By consistent, we mean that our

[1] A Voronoi diagram partitions a plane into regions based on distance to cluster centers.

Table 1. Cost Comparison (n: # points; k: # clusters; t: # lower bounds per point; α: fraction of points passing through the non-local filter; Drake's algorithm has no local filter)

Algorithm	space cost	time cost		
		lower bounds maintenance	non-local filtering	local filtering
Yinyang K-means	$O(n*t)$	$O(n*t)$	$O(n) \sim O(n*t)$	$O(n*\alpha*k)$
Elkan's (Elkan, 2003)	$O(n*k)$	$O(n*k)$	$O(k^2*d+n)$	$O(n*\alpha*k)$
Drake's (Drake & Hamerly, 2012)	$O(n*t)$	$O(n*t) \sim O(n*k+n*t*\log t)$	$O(n*t)$	–

Table 2. Time and speedup on an Ivybridge machine (16GB memory, 8-core i7-3770K processor)
("-" indicates that the algorithm fails to run for out of memory)

Data Set	n	d	k	No. iter	Assignment					Overall Speedup (X) of Yinyang over		
					Standard time/iter (ms)	Speedup (X) over Standard						
						Elkan	Drake	Yinyang K-means $t=1$	Yinyang K-means elastic	Standard	Elkan	Drake
I. Kegg Network	6.5E4	28	4	50	2.7	1.29	1.97	2.08	2.08	1.14	1.09	1.07
			16	52	9.9	1.62	2.13	2.48	2.48	1.61	1.36	1.12
			64	68	28.0	1.78	2.21	2.55	3.37	2.61	1.98	1.56
			256	59	89.6	1.89	1.63	2.23	4.98	4.86	3.60	3.98
II. Gassensor	1.4E4	129	4	16	3.1	4.60	4.34	4.68	4.68	1.13	1.07	1.11
			16	54	5.4	2.84	2.01	2.70	2.70	1.41	1.07	1.27
			64	66	20.3	5.08	3.08	3.17	5.49	3.29	1.82	2.28
			256	55	84.3	6.48	2.06	3.01	10.28	5.40	1.85	4.72
III. Road Network	4.3E5	4	4	24	10.1	**0.72**	1.23	1.36	1.36	1.18	1.24	1.17
			64	154	80.0	**0.85**	3.42	4.10	3.85	3.63	3.82	1.12
			1,024	161	1647.3	1.25	2.14	4.08	8.45	13.59	12.71	5.21
			10,000	74	16256.1	-	1.88	2.80	9.63	12.57	-	6.84
IV. US Census Data	2.5E6	68	4	6	182.0	1.88	1.94	2.08	2.08	1.10	1.04	1.04
			64	56	2176.4	3.57	4.56	4.85	8.47	5.40	2.43	2.14
			1,024	154	37603.9	**0.23**	2.96	3.56	24.89	23.45	89.53	6.33
			10,000	152	432976	-	- (1.64)	2.90	3.05	5.70	-	- (2.15)
V. Caltech101	1E6	128	4	55	111.0	2.44	2.88	3.02	3.02	1.83	1.41	1.04
			64	314	1432.6	5.52	5.07	5.64	10.21	8.65	1.79	1.26
			1,024	369	22816.8	5.56	3.62	3.38	21.99	22.33	6.41	5.71
			10,000	129	316850	-	- (3.25)	3.12	20.24	22.23	-	- (6.74)
VI. NotreDame	4E5	128	4	145	46.8	2.85	3.38	3.69	3.69	2.40	1.65	1.05
			64	232	585.8	5.27	4.57	4.29	6.81	6.16	1.88	1.76
			1,024	149	9334.1	5.66	2.82	2.28	10.44	10.69	3.25	4.19
			10,000	47	126815	-	2.35	2.32	10.81	11.53	-	5.27
VII. Tiny	1E6	384	4	103	277.0	6.67	7.58	8.20	8.20	3.24	1.90	1.21
			64	837	4113.4	14.23	7.39	6.32	15.26	13.89	1.93	1.93
			1,024	488	64078.8	16.02	4.37	2.94	23.64	23.21	2.78	5.14
			10,000	146	781537	-	- (3.45)	2.35	15.51	16.13	-	- (5.96)
VIII. Ukbench	1E6	128	4	62	113.7	2.63	2.86	3.17	3.17	1.94	1.46	1.10
			64	506	1431.1	5.75	7.36	6.61	13.21	10.85	3.12	1.72
			1,024	517	22787.4	5.95	4.28	3.42	23.41	24.26	6.85	5.18
			10,000	208	316299	-	- (3.92)	3.09	28.50	32.18	-	- (6.32)
					average	4.33	3.39	3.51	9.87	**9.36**	**6.12**	**3.08**

approach scales well with the size of the data (n), dimension of the data (d), and the number of clusters (k), and performs well under different levels of space pressure. This is because of the more effective filter design, and its space-conscious elasticity for trading off compute (by eliminating redundant distance computations) with space (the number of lower bounds maintained per point) so that Yinyang K-means is able to effectively adapt to space constraint.

The middle several columns of Table 2 show the speedups of the *assignment* step by the Elkan's, Drake's algorithm and Yinyang K-means. In order to investigate how each algorithm scales, we test various number of clusters (k) from 4 to 10,000. To keep a cluster having a meaningful size, we limit k to no greater than 256 for the first two data sets for their small sizes. The "elastic" columns in Tables 2 and 3 report the speedup of Yinyang K-means over the standard K-means, where, on the 16GB machine, the algorithm selects $t = k/10$ for all cases except that it uses 800 for the largest dataset (IV) when $k = 10,000$; while on the 4GB machine, as a reaction to the smaller space, when $k = 10,000$, it automatically reduces the value of t except for the two small ones (100 for data set IV and 500 for others) so that all its executions fit in memory. Table 3 does not show data set VII because it cannot fit in memory even in the case of the standard K-means on that machine.

Elkan's algorithm, which directly keeps k local lower bounds for each point, is still one of the fastest known exact K-means. In comparison, our results show that Yinyang K-means gives consistent and significant speedup. The consistency manifest in three aspects. First, unlike Elkan's al-

Table 3. Overall speedup over standard K-means on a Core2 machine (4GB mem, 4-core Core2 CPU)
(*: not a meaningful setting for the small data size; -: out of memory)

Data Set		I	II	III	IV	V	VI	VIII
k=4	Yinyang	1.35	1.10	1.09	1.13	1.97	2.60	2.05
	Elkan	1.09	1.08	**0.90**	1.06	1.30	1.44	1.32
	Drake	1.26	1.05	1.05	1.08	1.79	2.44	1.82
k=64	Yinyang	2.34	2.91	2.79	5.23	8.12	5.75	10.39
	Elkan	1.33	2.29	**0.97**	2.25	3.52	3.23	3.43
	Drake	2.04	1.67	2.31	4.42	3.17	2.96	3.28
k=1024	Yinyang	*	*	8.98	20.41	22.64	10.64	27.34
	Elkan	*	*	1.20	–	–	3.52	–
	Drake	*	*	2.18	– (3.18)	3.26	2.48	3.79
k=10,000	Yinyang	*	*	14.74	6.39	17.87	8.20	28.11
	Elkan	*	*	–	–	–	–	–
	Drake	*	*	– (2.01)	– (1.68)	– (2.58)	– (1.73)	– (3.02)

Table 4. Unnecessary distance calculations detected by the non-local filters of Yinyang K-means and two prior algorithms ($k = 64$)

Data Set	I	II	III	IV	V	VI	VII	VIII	average
Yinyang	69.3%	71.0%	88.5%	85.1%	81.7%	77.9%	82.0%	86.2%	80.2%
Elkan	31.1%	32.5%	32.6%	9.2%	0.5 %	2.0E-6	1.6E-6	1.4%	13.4%
Drake	64.2%	58.6%	69.3%	71.7%	73.6 %	68.1%	62.9%	77.7%	68.3%

gorithm which often fails (marked with "-") for running out of memory when k is large due to its $n * k$ space overhead, Yinyang K-means scales with k and shows even greater speedup for larger k values. Second, when the amount of available memory becomes smaller as Table 3 shows, Yinyang K-means still produces substantial speedups on all data sets and k values thanks to its elastic control of t, while Elkan's algorithm fails to run in even more cases. Finally, when d is small as in the third data set, Elkan's algorithm runs slower than the standard K-means. It is because to detect an unnecessary distance calculation, in most cases, Elkan's algorithm requires $6 * k$ condition checks per point, the time overhead incurred by which is even comparable to the distance calculations on that point. The more effective group filter in Yinyang K-means ensures that for most points, only t condition checks are sufficient per point, and hence gives up to 15X speedups for that data set.

Drake's algorithm maintains one upper bound and t lower bounds: $t-1$ lower bounds for the first $t-1$ closest centers, and one for all other centers. As they mentioned in the paper (Drake & Hamerly, 2012), this design can beat Elkan's algorithm in the intermediate dimension $20 < d < 120$ for intermediate k (e.g., $k = 50$). Our technique, on the other hand, shows good speedup in all the cases. Our speedup over Drake's algorithm stems from three sources. Its grouping-based filter makes it less sensitive to "big movers", its local filter helps further avoid distance calculations, and its elasticity makes it better adapt to space constraints. Drake's algorithm sets t to $k/4$ and gradually reduces it to $k/8$. It fails to run on the large data sets when k is large as the "-" shows in Tables 2 and 3. We managed to extend the algorithm with our elastic scheme (reducing t to fit memory constraint) to make the algorithm work. The results are shown in parentheses in Tables 2 and 3. Even with the extension, Yinyang K-means is still over 3X faster than Drake's algorithm on average.

The carefully designed group filtering of Yinyang K-means contributes substantially to the speedup. As Table 4 shows, the filter consistently outperforms the filters in both Elkan's and Drake's algorithms. It avoids 80% of distance calculations on average; in comparison, Elkan's and Drake's algorithms avoid 13% and 68% respectively. Its local filtering gives it a further edge over Drake's algorithm.

Another algorithm based on triangle inequality is by Hamerly and others (Hamerly, 2010). It only maintains one lower bound and one upper bound across iterations. It has no group filtering, local filtering, or optimization of center update. Its performance is similar to our global filtering, and more than 3 times lower than our algorithm.

We also investigate works in the area of approximation algorithm (Czumaj & Sohler, 2007; Sculley, 2010). We compared our method with mini-batch K-means (Sculley, 2010) in terms of performance and clustering quality

Table 5. Yinyang K-means accelerates the center update step by many times
(*: not a meaningful setting for the small data size)

Data Set	I	II	III	IV	V	VI	VII	VIII
k=4	2.4	3.2	2.1	1.8	4.1	4.3	9.0	4.0
k=64	9.6	20.1	8.4	8.0	11.0	9.8	15.4	11.9
k=1024	*	*	107.0	59.9	97.0	43.1	106.6	114.3
k=10,000	*	*	62.5	79.7	66.2	27.5	50.8	100.4

(within-cluster sum of squares or WCSS). Mini-batch K-means takes around 10X or more iterations than Yinyang K-means takes to reach a similar level of clustering quality on the datasets used in our experiments. Because mini-batch K-means uses $4 * k$ samples in each iteration, when k is large ($> n/4000$ in our experiments), mini-batch K-means takes significantly more time than Yinyang K-means does to produce clusters of a similar level of quality. At a high level, Yinyang K-means and approximation-based K-means algorithms are orthogonal and could be used in a complementary manner. For instance, the filters in Yinyang K-means could potentially also help reduce distance calculations in approximation-based algorithms. Approximation-based algorithms could be used in cases where the clustering quality is not critical, while Yinyang K-means can be used in all scenarios where the standard K-means applies. Being able to give the same results as the standard K-means gives but in only a small fraction of its time is powerful: The great number of practical uses of the standard algorithm can directly adopt our algorithm without worrying about any changes in the output. It naturally inherits the trust that the Lloyd's algorithm has established among users through decades of usage.

Optimization of Center Update K-means is a two-step iterative algorithm. As time complexity of the *center update* step is less than that of the *assignment* step, prior work focuses only on improving the *assignment* step. But with that step optimized, the *update* step starts to appear prominent in time. Table 5 reports the speedup obtained by the optimization described in Section 2.2. The speedup ranges from 1.8X to over 100X; substantial speedup show up when k is large because the time cost of center update calculation, instead of the launching process itself, becomes prominent.

Overall Speedup The rightmost three columns in Table 2 report the overall speedup of Yinyang K-means over the standard K-means, Elkan's algorithm and Drake's algorithm, in terms of end-to-end execution time. Yinyang K-means is the fastest in all cases. It is an order of magnitude faster than the standard K-means in most cases, and several times faster than Elkan's and Drake's algorithms.

Sensitivity Study on t The parameter t determines the number of lower bounds maintained per point in Yinyang K-means. This parameter allows automatically balancing redundant computation and the memory and time costs of maintaining such bounds.

Figure 3 shows the averaged assignment times for Yinyang K-means as a function of t. For legibility, it includes the results on the four image data sets only; similar results show on other data sets. As t increases, the performance of Yinyang K-means first improves and then reaches optimal around $t = k/10$ after which performance decreases. This is because although increasing t produces tighter lower bounds which help eliminate redundant distance calculations, it incurs more comparison operations and more lower bound updates, which at some point adversely impact performance. The observation led to the design of the aforementioned policy on selecting the value for t in Yinyang K-means. It is worth mentioning that even when t is 1 (when the group filter reduces to global filter), Yinyang K-means still consistently outperforms the standard K-means on all data sets in all settings, as shown by the "t=1" column in Table 2, which demonstrates the benefits of the new way to approximate the overlapped Voronoi areas.

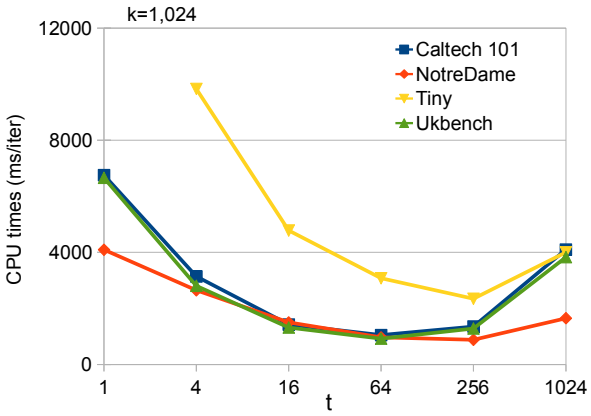

Figure 3. Averaged CPU times of assignment step for Yinyang K-means as a function of t.

6. Conclusion

Overall, this study shows that Yinyang K-means is consistently faster than prior algorithms, regardless of the dimension and size of the data sets, the number of clusters, and the machine configurations. It accelerates both the *assignment* and *center update* steps in K-means, and automatically strikes a good tradeoff between space cost and performance enhancement. Its elastic design makes it automatically maximize its performance under a given space constraint. It preserves the semantic of the original K-means. These appealing properties, plus its simplicity, make it a practical replacement of the standard K-means as long as triangle inequality holds.

The source code of this work is available at http://research.csc.ncsu.edu/nc-caps/yykmeans.tar.bz2
.

Acknowledgement We thank the ICML'15 reviewers for their suggestions. This material is based upon work supported by DOE Early Career Award and the National Science Foundation (NSF) under Grant No. 1464216, 1320796 and Career Award. Any opinions, findings, and

conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DOE or NSF.

References

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

Bache, K. and Lichman, M. UCI machine learning repository, 2013. URL http://archive.ics.uci.edu/ml.

Bahmani, B., Moseley, B., Vattani, A., Kumar, R., and Vassilvitskii, S. Scalable k-means++. In *Proceedings of the VLDB Endow.*, 2012.

Curtin, Ryan R., Cline, James R., Slagle, Neil P., March, William B., Ram, P., Mehta, Nishant A., and Gray, Alexander G. MLPACK: A scalable C++ machine learning library. *Journal of Machine Learning Research*, 14: 801–805, 2013.

Czumaj, Artur and Sohler, Christian. Sublinear-time approximation algorithms for clustering via random sampling. *Random Structures & Algorithms*, 30(1-2):226–256, 2007.

Drake, Jonathan and Hamerly, Greg. Accelerated k-means with adaptive distance bounds. In *5th NIPS Workshop on Optimization for Machine Learning*, 2012.

Elkan, Charles. Using the triangle inequality to accelerate k-means. In *ICML*, volume 3, pp. 147–153, 2003.

Guha, Sudipto, Rastogi, Rajeev, and Shim, Kyuseok. Cure: an efficient clustering algorithm for large databases. *ACM SIGMOD Record*, 27(2):73–84, 1998.

Hamerly, Greg. Making k-means even faster. In *SDM*, pp. 130–140. SIAM, 2010.

Kanungo, Tapas, Mount, David M, Netanyahu, Nathan S, Piatko, Christine D, Silverman, Ruth, and Wu, Angela Y. An efficient k-means clustering algorithm: Analysis and implementation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(7):881–892, 2002.

Low, Yucheng, Gonzalez, Joseph, Kyrola, Aapo, Bickson, Danny, Guestrin, Carlos, and Hellerstein, Joseph M. Graphlab: A new framework for parallel machine learning. *arXiv preprint arXiv:1006.4990*, 2010.

OpenCV. URL http://opencv.org/.

Pelleg, Dan and Moore, Andrew. Accelerating exact k-means algorithms with geometric reasoning. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 277–281. ACM, 1999.

Philbin, James, Chum, Ondrej, Isard, Michael, Sivic, Josef, and Zisserman, Andrew. Object retrieval with large vocabularies and fast spatial matching. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pp. 1–8. IEEE, 2007.

Sculley, D. Web-scale k-means clustering. In *Proceedings of the 19th international conference on World wide web*, pp. 1177–1178. ACM, 2010.

Wang, Jing, Wang, Jingdong, Ke, Qifa, Zeng, Gang, and Li, Shipeng. Fast approximate k-means via cluster closures. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pp. 3037–3044. IEEE, 2012.

Wang, Jingdong, Wang, Naiyan, Jia, You, Li, Jian, Zeng, Gang, Zha, Hongbin, and Hua, X. Trinary-projection trees for approximate nearest neighbor search. *Trans. Pattern Anal. Mach. Intell*, 2013.

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

Zhao, W., Ma, H., and He, Q. Parallel k-means clustering based on mapreduce. *LNCS*, 5931:674–679, 2009.