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

Yufei Ding
North Carolina State University
yding8@ncsu.edu

Xipeng Shen
North Carolina State University
xshen5@ncsu.edu

Madanlal Musuvathi
Microsoft Research
madanm@microsoft.com

Todd Mytkowicz
Microsoft Research
mytkowicz@microsoft.com

Abstract

This paper presents Yinyang K-means, a new algorithm for K-means clustering. By clustering the centers in the initial stage, it leverages efficiently maintained lower and upper bounds between a point and centers, it more effectively avoids unnecessary distance calculations than prior algorithms do. It significantly outperforms the classic K-means and the prior known alternatives (Elkan's, Hamery's and Drake's accelerated algorithms) consistently across all experimented data sets, cluster numbers, and machine configurations. The consistent superior performance—plus its simiplicity, elastic control of space cost, 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

K-means is a ubiquitous algorithm for clustering. Given a set of d-dimensional data, K-means partitions the data into k sets, called clusters, so as to minimize the within-cluster sum of square error. Finding the optimal solution to K-means is **NP**-hard [1]; however, iterative methods based on local search, in particular, Lloyd's algorithm are often, at worst, bounded by a polynomial in the size of the input [2]. Given an input of n data points of d dimensions and k initial clusters, Lloyd's algorithm gives a greedy, two-step iterative solution. In the *assignment step*, the algorithm assigns all points to the closest cluster and in the *update step* the algorithm updates the each of the k cluster centers to the centroid of the points assigned to that cluster. When n, k, or d are large, K-means is slow due to its linear dependence on n, k, and d in the assignment and update steps.

Various prior efforts try to improve its speed through three general techniques, approximation[4, 7, 12–14], structural optimization [9, 11], and incremental optimization [5, 6, 8]. Each of these efforts have made significant contributions. But due to their respective limitations, the classic Lloyd's algorithm remains the dominant choice in practice. The appoximation methods use various trade-offs to approximate the output of K-means; without preserving the semantics of K-means, they cannot be used as drop in replacements for the exact algorithm. The other two classes of methods try to avoid some distance computations through either special data structures (e.g., KD-Tree [9, 11]) or the triangular inequality of distances [5**?** , 6]. They lack performance consistency: methods on KD-Tree, for instance, cannot work well when d is greater than 20 [9], while the method on triangular inequality either cannot scale with k or perform inferiorly in some scenarios (detailed in Section 5). We believe that a practical replacement of the classical K-means must be easy to implement and have a consistent superior performance. Meanwhile, to inherit the level of trust K-means has gained through its decades of usage, it is better to preserve the semantics of the classic K-means.

This paper introduces Yinyang K-means, an enhanced K-means that meets all the conditions. The key is in its careful but efficient maintainence 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 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 filter 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, outperforming the standard K-means by an order of magnitude and the fastest prior known K-means algorithm by 1.2X to 90X. Its simplicity, elasticity, semantics preservation, and consistent superior performance make it a practical replacement of the standard K-means.

2. Yinyang K-means

This section describes the Yinyang K-means algorithm and describes the optimizations to both the assignment and the update steps. These optimizations rely on the use of the triangle inequality with careful tradeoffs between the cost incurred in performing distance calculations and the space required to apply the 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.

2.1 Optimizing the *Assignment* Step

The Yinyang K-means algorithm uses the triangle inequality to generate distance bounds between on point to a group of clusters, and based on the size of the group, the application of the triangle inequality results in three optimizations: *global filtering*, *group filtering*, and *per-cluster filtering*. These three optimizations differ in the tradeoff between the gain in distance-calculation reduction and the space overhead required to apply them.

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.

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 $d(x,b(x)) \leq ub(x)$ 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 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. 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 to maintaining an upper and lower bounds for each point requiring $O(n)$ 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.

Despite its simplicity in terms of additional space and time required, our experiments (Secton 5) show that the global filtering alone is sufficient to reduce 69.6% of the distance calculations on average.

2.1.2 Group Filtering

Our experiments also suggest that the efficiency of global filtering can dramatically reduce in the presence of *big-movers* — cluster centers that drift dramatically in a center update. Even a single big-mover reduces the lower-bound for all points, making the global-filtering ineffective.

This section provides group filtering, a method similar to the global filtering but having the effects of these big-movers reduced. Group filtering partitions the k clusters into t groups $G = \{G_1, G_2, ..., G_t\}$, where each $G_i \in G$ contains a set of clusters. This partitioning is done once before the beginning of the first iteration. 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 can be filtered in the assignment step.

The paramter 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 show that when $k/40 \leq t \leq k/10$, the method gives competitive results. 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 partition the k clusters into t groups, for instance, randomly dividing them to t groups. Our investigation shows that clustering on the initial centers is a good choice. Compared to random distribution, it benefits more from the locality of the centers and thus, yields better performance. Such partition is a one-time job, only needed at the beginning the first iteration. Repartitioning, while feasible, did not help as observed in our experiments.

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 cener $p' \neq c'$ (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, it is tempting to use the so-far-found closest center as p'. Even though it taps into the full potential of the filter, 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 slightly more distance

calculations it entails help tighten the group lower bounds, which in turn reinforce the group filter 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 check this condition in each iteration and use the new algorithm when it holds. However, 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.

- Cluster the initial centers into t groups, $\{G_i | i = 1, 2, \cdots, t\}$ by running K-means on just those centers. We found no need for that K-means to converge; five iterations are enough for it to produce reasonable groups while incurring no much time overhead.
- Run the standard K-means on the points for the first iteration, record one upbound $ub(x)$ and t lower bounds $\{lb(x, G_1), lb(x, G_2), \cdots, lb(x, G_t)\}$ for each point x as described in Section 2.1.2.
- Repeat until convergence:
  - Update centers based on Section 2.2, compute drift of each center, and record the maximum drift for each group.
  - Group filtering: For each point x, update $ub(x)$ and $lb(x, G_i)$ as shown in Section 2.1.2. Assign the temporary global lower bound as $lb(x) = \min_{i=1}^{t} lb(x, G_i)$. If $lb(x) >= ub(x)$, 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 per-cluster filtering.
  - Per-cluster filtering: for each point x and remaining group G_i. Update $lb(x, G_i)$ according to Section 2.1.3, and update $b(x)$ and $ub(x)$ if the point assignment changes.

4. Comparison

The work closest to ours are the prior known accelerated K-means by Elkan [6] and Drake [5]. They also uses triangle inequality to avoid some distance calculations, but differs from our algorithm in some critical aspects. Compared to Elkan, our algorithm shows great advantage in the efficiency of the non-local filter (group/-global filter). Figure 1 shows that intuitively. Figures 1 (a) and (b) depict the Voronoi diagrams 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 1 (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 1 (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 80X in most cases) more distance calculations than the non-local filter in Elkan's algorithm; details in Section 5.

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. The cost causes the algorithm to fail or perform inferiorly in some scenarios, as next section will show.

Drake's algorithm tries to save time and space cost of the local filter of Elkan's algorithm, by maintaining the lower bounds for each point to its t cloest centers. However, this design still suffers from "big movers", as different points have different set of t cloest centers. Besides, to maintain the increasing order of these t bounds across iteration, they sacrifice additional tightness, which further reduces the efficency of their filters. As a result, their algorithm only yields better performance for a limited region of k and d, compared to Elkan's algorithm.

In comparison, Yinyang K-means overcomes these limitations through its space-conscious elasticity and more effective design of the filters. Moreover, Elkan's and Drake's algorithm is only for the *assignment* step, while Yinyang K-means also optimizes the center update step.

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 [6]), Drake [5] and standard K-means. All three algorithms are implemented in Graphlab [10] 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 [3], while the other four are commonly used image data sets[14, 15]. Their size and dimensionality 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 approach scales well with the size of the data (n), dimensionality 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 always able to fit in core memory.

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

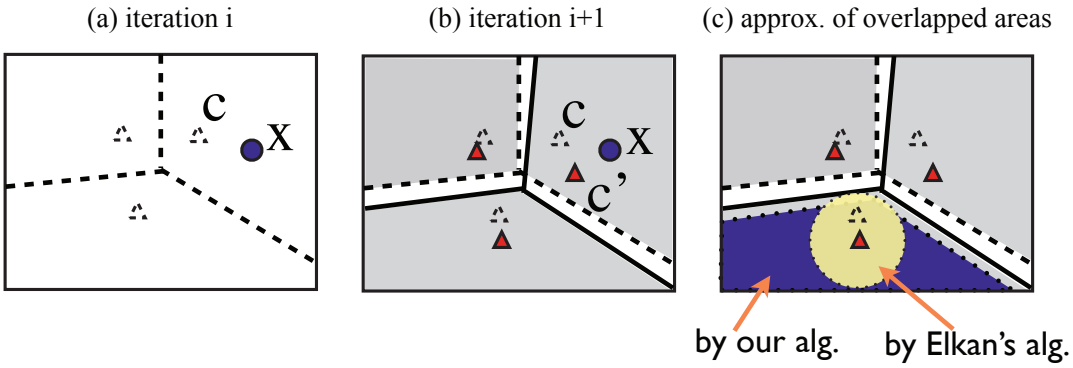

Figure 1. The Voronoi diagrams in two consecutive iterations of k-means, and the approximation of the overlapped areas. The shown approximation by our algorithm is when all centers are treated as in one group (i.e., t=1); when $t > 1$, more distance calculations are avoided. (Triangles for centers; circles for points; broken and solid lines for cluster boundaries, in the two iterations respectively.)

Table 1. Cost Comparison (α: the fraction of points passing the non-local filter, and Drake's algorithm does not have a local filter)

Algorithm	space cost	time cost		
		lower bounds maintainence	non-local filtering	local filtering
Yinyang K-means	$O(n*t)$	$O(n*t)$	$O(n)$	$O(n*\alpha*k)$
Elkan's [6]	$O(n*k)$	$O(n*k)$	$O(k^2*d+n)$	$O(n*\alpha*k)$
Drake's [5]	$O(n*t)$	$O(n*t*\alpha+n*k*\log t*(1-\alpha))$	$O(n*t)$	–

the first two data sets for their small sizes. The "elastic" column in Table 2 and Table 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, 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 results on data set VII because it cannot fit in memory even in the case of the standard K-means.

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 algorithm 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 comparible 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 9X speedups for that data set.

Drake's algorithm maintains one upper bounds 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, this design can beat Elkan's algorithm in the intermediate dimentionality $20 < d < 120$ 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: First is "big mover". As the k closest centers for different points are different, Drake's algorithm cannot afford to calculate the maximum movement of the $(k-t+1)$ farthest centers and instead uses the maximum movement of all centers for updating the last lower bound, which makes their technique suffer from "big movers". Second is the uniformity. The last lower bound in Drake's is used for too many clusters $(k-t+1)$. If this lower bound is smaller than the upper bound, then distance to all clusters are recalculated. In comparison, our technique provides a much more balanced scheme. Moreover, Drake's algorithm does not have local filtering. We discussed how lower bound for group of clusters can be used to generate per cluster lower bound, which further mitigates the problem of "big movers" and eliminate more redundant distance calculation.

Need to mention, we also check on other algorithm, that is also based on triangle inequality, Hamerly's algorithm [8]. It only maintains one lower bound and one upper bound across iteration, similar as the $t = 1$ case described in our paper. But they do not have any group filtering and local fiterling, or optimization for center update. So it is overall performance is even worse than our $t = 1$ case, let alone the elastic method. And due to the space limit, we did not include the exact performance for this method.

The carefully designed group filtering of Yinyang K-means contributes substantially to the speedup. As Table 4 shows, the filter helps avoid more than 69% of distance calculations for all the data sets, while the Elkan's non-local filter avoids less than 33% for three data sets and less than 10% for the other data sets. Drake's algorithm, as a comparison, gives a mediate filter power. On average, its non-local filter avoids 68% distance calculations.

We also inverstigate works in the area of approximation algorithm [4, 13]. Our work is orthogonal to those previous studies and can be used in a complementary maner. The clustering results from our algorithm are guaranteed to be the same as the results from the original Lloyd's algorithm. Maintaining the semantics of the original Lloyd's approach is powerful because the great number of practical uses of the Lloyd's algorithm can directly adopt our algorithm without worrying about any changes in the output. In terms of quantitative comparison, Sohler's work is a pure theoretical paper. Although it offers excellent theoretical results, it gives no

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 of Yinyang over		
					Standard time/iter (ms)	Speedup 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

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 filter of Elkan's algorithm and the group filter of Yinyang K-means ($k = 64$)

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

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

empirical evaluation. We compared our method with mini-bath K-means [13]. In general, to get the same accuracy level, mini-bath K-means takes around 10X or more iterations. In general, mini-batch k-means shows better performance for small k, and our method becomes better when k gets larger.

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 of faster than Elkan's and Drake's algorithm.

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 2 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 eliminates redundant distance calculations, it does so by using more space ($O(n*t)$), which at some point adversely impacts performance. We empirically observe $t = k/10$ across all our data sets is a good balance, hence 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.

Overall, the experiments confirm that Yinyang K-means is consistently faster than both the standard K-means, Elkan's and Drake's algorithms, regardless of the dimensionality 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 auotmatically strikes a good tradeoff between space cost and performance enhancement.

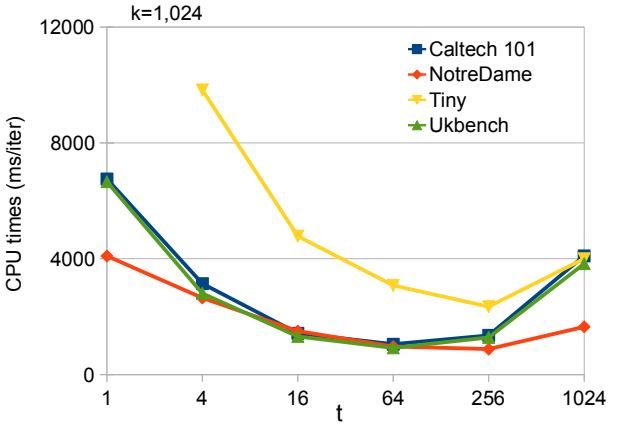

Figure 2. Averaged CPU times of assignment step for Yinyang K-means as a function of t.

6. Conclusion

This study demonstrates that Yinyang K-means gives consistent significant speedup compared to standard implementation and the prior known fastest alternative. 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.

References

[1] D. Aloise and P. Hansen. A branch-and-cut sdp-based algorithm for minimum sum-of-squares clustering. *Pesquisa Operacional*, 29(3): 503–516, 2009.

[2] D. Arthur, B. Manthey, and H. Roglin. k-means has polynomial smoothed complexity. In *Foundations of Computer Science, 2009. FOCS'09. 50th Annual IEEE Symposium on*, pages 405–414. IEEE, 2009.

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

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

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

[6] C. Elkan. Using the triangle inequality to accelerate k-means. In *ICML*, volume 3, pages 147–153, 2003.

[7] S. Guha, R. Rastogi, and K. Shim. Cure: an efficient clustering algorithm for large databases. In *ACM SIGMOD Record*, volume 27, pages 73–84. ACM, 1998.

[8] G. Hamerly. Making k-means even faster. In *SDM*, pages 130–140. SIAM, 2010.

[9] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu. An efficient k-means clustering algorithm: Analysis and implementation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(7):881–892, 2002.

[10] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein. Graphlab: A new framework for parallel machine learning. *arXiv preprint arXiv:1006.4990*, 2010.

[11] D. Pelleg and A. 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.

[12] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE, 2007.

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

[14] J. Wang, J. Wang, Q. Ke, G. Zeng, and S. Li. Fast approximate k-means via cluster closures. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3037–3044. IEEE, 2012.

[15] J. Wang, N. Wang, Y. Jia, J. Li, G. Zeng, H. Zha, and X. Hua. Trinary-projection trees for approximate nearest neighbor search. *Trans. Pattern Anal. Mach. Intell*, 2013.