New ideas based on the Hamerly's algorithm to accelerate K-means algorithm

Zezhong Wen
Department of Computer Science
George Washington University
Washington, DC 22202
wenzezhong@gwu.edu

Abstract

The k-means algorithm is widely used in data clustering applications. The popular Lloyd's algorithm does many unnecessary distance calculations. Several algorithms have been developed to accelerate Lloyd's algorithm such as Hamerly's algortihm. Hamerly's algorithm is based on the triangle inequality and uses a set of lower and upper bounds on point-centroid distances. In this paper a new update function is proposed based on Hamerly's algorithm to make upper/lower bounds tighter. It tracks the movement of centroids to avoids unnecessary bounds updates in Hamerly's algorithm. The experiment shows that the proposed algorithm always has better k-means objective and smaller number of distance calculations. For CPU time, it performs better than Hamerly's algorithm when the number of centroids is relatively large.

1 Introduction

Clustering is used frequently in machine learning and is often the significant part of learning systems. It has many applications, such as social network analysis and market study. So clustering algorithms should be fast. In this paper a new method is proposed based on prior algorithms [1] [2] [3]. The experiment shows that in some data sets the proposed algorithm performs better than Hamerly's algorithm.

clusters are sets of similar points according to specific measure. The measure is the squared distance between points and their assigned centroids. Point is assigned to its closest centroid in k-means algorithm.

$$J(c) = \sum_{i=1}^{n} ||x_i - c(x_i)||^2$$

This distortion function is to be minimized by assigning points to their closest centroids.

The popular Lloyd's algorithm [4] repeats a lot of unnecessary distance calculations across iterations. Several algorithms have been proposed to eliminate the redundancy. In this work we will focus on Hamerley's algorithm [3] to accelerate k-means algorithm. The paper will briefly introduce main features of some prior algorithms in Prior algorithms review.

These algorithms use triangle inequality [2] as well as lower/upper bounds to avoid unnecessary calculations. So these algorithms are faster than Lloyd's original algorithms. In this paper new ideas are proposed based on Hamerly's algorithm.

The new propose avoids some unnecessary lower/upper bounds update by tracking the movement of every center each iteration. Also it tracks the identity of the second closest centroid as well as

30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

the closest centroid of a point. The algorithm checks if the centroid moves away from/towards the point to determine whether there is need to update bounds. So the bounds are tighter than Hamerly's algorithm.

Experiments generally show that the proposed algorithm is faster than Hamerly's algorithm in some datasets. Also, it always has better k-means objective.

In Section 2 the paper briefly analyze prior algorithms related to this work. In Section 3 we present new methods and analyze the different features of the algorithm. In Section 4 the paper presents experiments results of the proposed algorithms and Section 5 is conclusion.

2 Prior algorithms review

2.1 Triangle Inequality Approaches

The triangle inequality [2] is a simple but useful method. It can be used in the k-means algorithm in multiple ways. Generally we use it to prove that some centroid is so far away from the point that we do not need to calculate the distance. Conversely, if a point is much closer to a centroid than to any other, there is no need to calculate the exact distance to determine the assignment. According to Elkan(2003), the two following two lemmas show how to use the triangle inequality to avoid distance calculations.

Lemma 2.1 *Let x be a point and let b and c be centers. If* $d(b,c) \geq 2d(x,b)$ *then* $d(x,c) \geq d(x,b)$

Lemma 2.2 *Let x be a point and let b and c be centers. Then* $d(b,c) \geq max\{0, d(x,b) - d(b,c)\}$

Lemma 1 is used as follows. Let x be a point and let c be the centroid of x, and let c' be any other centroid. The lemma says that if $\frac{1}{2}d(c,c') \geq d(x,c)$,then $d(x,c') \geq d(x,c)$. In this case, there is no need to calculate $d(x,c')$

We can use lemma 2 as follows. Let x be any data point, let b be any centroid, and let b' be the previous version of the same centroid. Suppose that in the previous iteration we knew a lower bound l such that $d(x,b') \geq l$. Then we can infer a lower bound for the current iteration: $d(x,b) \geq max\{0, d(x,b') - d(b,b')\} \geq max\{0, l - d(b,b')\}$

2.2 Hamerly's Algorithm

Hamerly's algorithm [3] (Algorithm 1) is based on triangle approach and it has only one lower bound per point. The lower bound means the distance between point x and its second closest centroid. It simplifies Elkan's algorithm in this way. If the lower bound is greater than the upper bound for some point x, we do not need to do the distance calculations. Between iterations the bounds are updated based on the triangle inequality. We assume the centroid moves toward point x considering lower bounds and we assume the assigned centroid moves away from x for upper bounds. We compare the lower and upper bounds to determine if we need to recalculate the distances.

3 Proposed method for speeding up k-means

In this section we will discuss new ideas based on Hamerly's algorithm. The paper analyzes the features of new algorithm and explains the difference between new algorithm and prior algorithms.

A new bounds update function

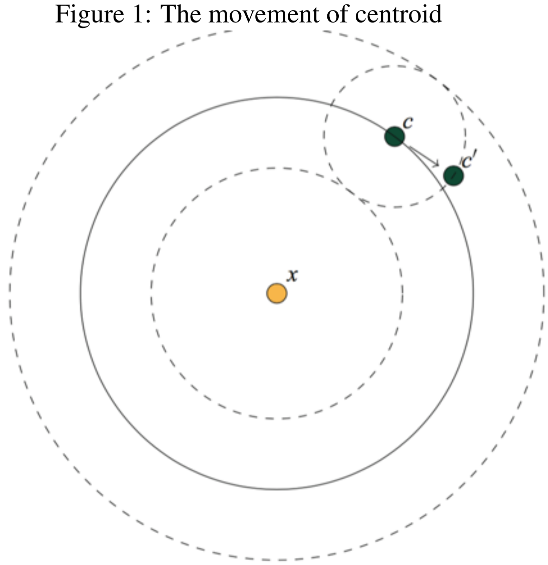
We assume that the centroid moves directly toward the point when we update the lower bounds using the triangle inequality. However, this is not always true considering centroid movement. For example, if some centroid c_j moves away from centroid c_i, the lower bound for x can even grow rather than shrink.

The paper proposes a new update function(Algorithm 2) to some unnecessary bounds updates in order to make bounds tighter.

begin
  $a(i) \leftarrow 1, u(i) \leftarrow 0, \forall i \in N$
  while *not converged* **do**
    compute $s(j) \leftarrow min_{i \neq j} ||c(j) - c(j')||/2, \forall j \in K$
    for *all* $i \in N$ **do**
      $z \leftarrow max(l(i), s(a(i)))$
      if $u(i) < z$ **then**
        continue with next i
      end
      $u(i) \leftarrow ||x(i) - c(a(i))||$ (Tighten the bound)
      if $u(i) < z$ **then**
        continue with next i
      end
      update $l(i), u(i), closest$
    end
    for *all* $i \in K$ **do**
      move $c(j)$ to new location
      $\delta(j) \leftarrow$ distance moved by $c(j)$
    end
    $\delta' \leftarrow max_{j \in K} \delta(j)$
    for *all* $i \in N$ **do**
      call update_bounds()
    end
  end
end

Algorithm 1: Hamerly's algorithm

Figure 1: The movement of centroid



The proposed algorithm (Algorithm 2) determines if lower bound or upper bound needs to be updated by checking the inner product of two vectors. One is the vector from centroid to the point x which is assigned to it and the other is the centroid movement vector.(the vector cx and cc' in the Figure 1) If the centroid moves away from the point, then the new algorithm will not decrease the lower bound and keep the previous value. It is similar considering upper bounds. In this way, some unnecessary updates of bounds will be avoided.

As is shown in Algorithm 2 and Algorithm 3, the new algorithm differs from Hamerly's algorithm in these aspects. First of all, the new algorithm rewrites the update_bounds function in Hamerly's

```
Function Hamerly:Update_bounds ()
    begin
        furthestMovingCenter ← 0;
        longest ← 0.0;
        secondlongest ← 0.0;
        for all j ∈ K do
            compare and update longest secondlongest furthestMovingCenter;
        end
        for all i ∈ N do
            u(i) += centerMovement(a(i));
            l[i] -= (a[i] == furthestMovingCenter)?secondLongest : longest;
        end
    end
a(i): the assigned centroid of point i
longest: the moving distance of the furthest moving center
second longest: the moving distance of the second furthest moving center
u(i): the upper bound of point i
l(i): the lower bound of point i
```

Algorithm 2: Update_bounds function in Hamerly algorithm

```
Function Hplus:Update_bounds ()
    begin
        for all i ∈ N do
            innerproductU ← vector(a(i), x(i)) · centerMovementVector(a(i));
            innerproductL ←
              vector(secondclosest(i), x(i)) · centerMovementVector(secondclosest(i));
            if innerproductU < 0.0 then
                u(i) += centerMovement(a(i));
            end
            if innerproductL > 0.0 then
                l[i] -= centerMovement((secondclosest(i)))
            end
        end
    end
a(i): the assigned centroid of point i
x(i): data point i
centerMovementVector: a global array keeping track of center movement
u(i): the upper bound of point i
l(i): the lower bound of point i
secondclosest: the global array keeping track of the secondclosest centroid of each point
```

Algorithm 3: Update_bounds function in Hplus algorithm

algorithm. The new algorithm does not do the for loop on k but instead add the inner product calculations of vectors. Second, the new algorithm keeps track of the second closest center in a global array. This is used for the vector calculations in the update_bounds function Third, the new algorithm keeps track of the center movement vectors in a global array in order for the calculations in the update_bounds function.

So the proposed algorithm decreases the number of distance calculations but adds inner product calculations instead.

4 Experiments and Discussion

The experiments are run on several types of datasets. For each experiment the CPU time needed to obtain the result, the memory usage, the k-means objective as well as the number of distance calculations are measured. The experiment mainly compares the proposed algorithm(Hplus) with Hamerly's algorithm.

All algorithms have taken as an input the same initialization produced by k-means++. All implementations were written in C++ with shared code. The platform is a 2-core 2.3GHz 64-bit Linux Intel machine with 8GB RAM. All implementations were single threaded. Tables and figures below show the results. The runtime results are averaged over 10 repeats of the algorithms.

Table 1 shows the synthetic and real-world datasets we used. Uniform-d, BIRCH [5] and MNIST-50 are the same datasets used in Hamerly's paper. To control the properties of clusterings and to apply the variate controlling approach, we create synthetic data generated from a mixture of Gaussians. we choose random vertices from multi-dimensional hypercube of side length 10. We then add gaussian random points (with variance 1) around each of these points.

Table 2 shows the comparison result of Hamerly's algorithm with the proposed algorithm in some datasets. The proposed algorithm shows better k-means objective in these datasets. Also, the number of distance calcualations is smaller using proposed datasets comparing to Hamerly's algorithm. For the CPU time, the proposed algorithm performs better on BIRCH and Uniform-2(about 3.5 times faster on BIRCH than Hamerly's algorithm) but the Hamerly's algorithm performs better on MNIST. (about 7.5 times faster than proposed algorithm)

To gain a clearer view of the two algorithms. We use variable controlling appraoch and synthetic data generated from a mixture of Gaussians. In our discussion, n denotes the number of points in the dataset, d denotes the dimension, and k denotes the number of clusters. The results are shown in Figure 1-4. Figure 1 shows as the k grows, the Hamerly's algorithm shows higher change rate in CPU time than the proposed algorithm. The proposed algorithm is faster when k is relatively large while the Hamerly's algorithm is faster when k is relatively small. Figure 2 shows that as the k grows, the difference between the two algorithms in k-means objective becomes bigger. Figure 3 shows that when d grows, the two algorithms show similar change rate in CPU time and the difference tends to be smaller. Figure 4 shows that as d grows, the difference between two algorithms in k-means objective tends to be smaller. In all experiments the proposed algorithm shows better k-means objective.

Table 1. Datasets in the experiments

Name	Description	Points N	Dimension D	k
Uniform-d	Synthetic data,uniform distribution	1000000	2/3/5/7	50
BIRCH [5]	Synthetic data,50 clusters of similar size	100000	2	100
MNIST-50	Random linear projection of MNIST-784	60000	50	10
Gaussian_d_k_n	Synthetic data,Gaussian distribution	10000	10-100	10-100

Table 2. Comparison of Hplus and Hamerly

Dataset	Algorithm	Cpu time	Memory	Distance counts	k-means objective
BIRCH	Hplus	0.72	7.01562	14197579	271448
	Hamerly	2.584	7.53125	96643580	377285
Uniform-2	Hplus	12.744	50.5938	80926502	7.52E+09
	Hamerly	25.852	60.3672	626755474	1.11E+10
MNIST-50	Hplus	6.136	27.1406	996264	5.80E+12
	Hamerly	0.824	27.8828	2818506	6.01E+12

Figure 4.1 cpu vs k;dimension=8;N=10000

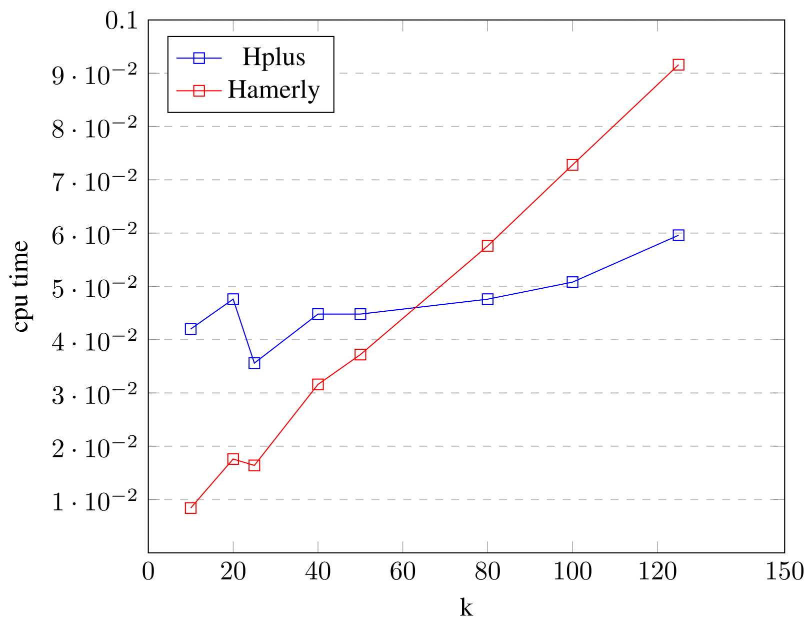


Figure 4.2. k-means objective vs k;dimension=8;N=10000

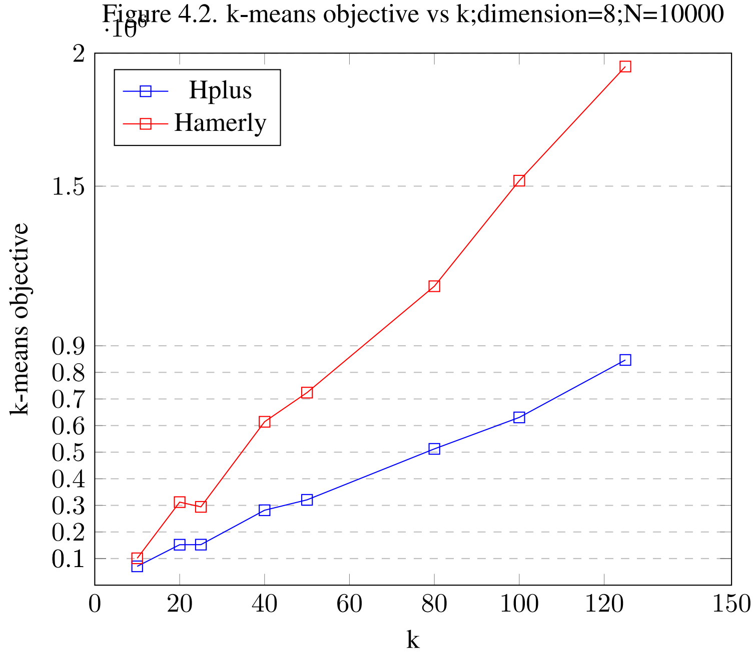


Figure 4.3. cpu vs d;k=100,N=10000

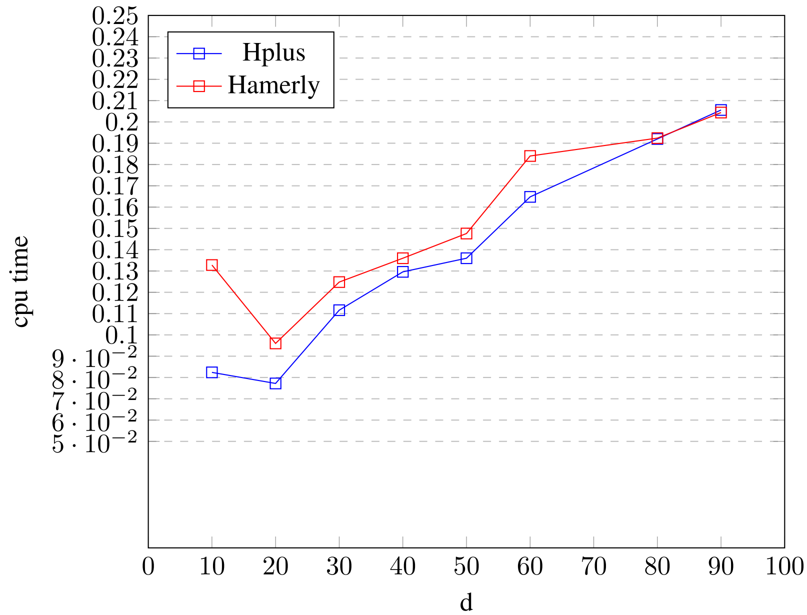
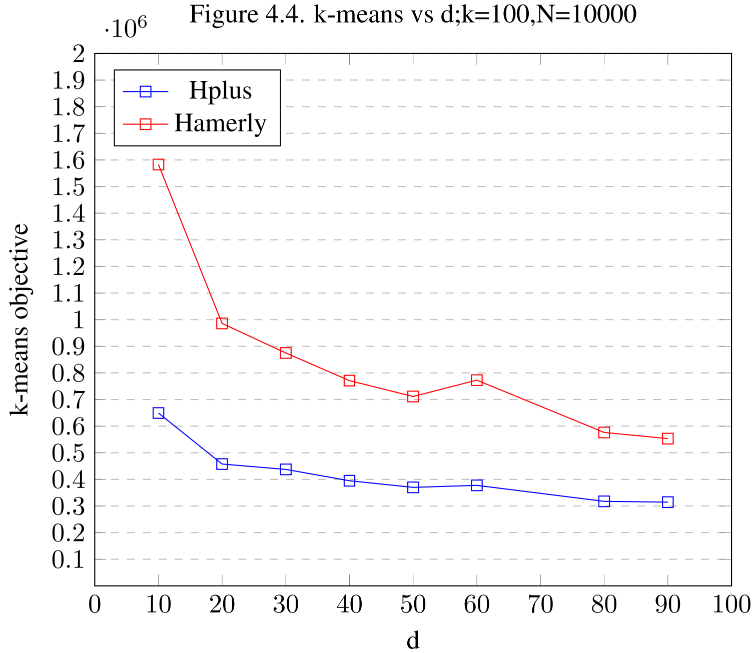


Figure 4.4. k-means vs d;k=100,N=10000



5 Conclusions

Clustering algorithms are so widely used in machine learning systems that they should be fast. According to Hamerly, "standard practice for finding good clusterings is to try multiple runs with multiple initializations." In this paper, a new update function is proposed based on Hamerly's algorithm to make the bounds tighter. Using tighter bound updates would allow the bounds to hold for more iterations and avoid more distance calculations. The experiment result shows that the proposed algorithm always has smaller k-means cost and smaller number of distance calculations. For CPU time, it performs better than Hamerly's algorithm when the number of centroids is relatively large.

References

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

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

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

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

[5] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: A new data clustering algorithm and its applications. *Data Mining and Knowledge Discovery*, 1(2):141–182, 1997.

6 Appendix

The raw data of the figures in the experiment section

Comparison of Hplus and Hamerly					
Dataset	Algorithm	Cpu time	Memory	Distance counts	k-means objective
g_d8_k10_10000	Hplus	0.042±0.025	3.685939±0.205	173982.72±82861.427	70766.5±30055.304
	Hamerly	0.0084±0.006	3.575391±0.09	174002.12±82890.736	100993.6±55786.767
g_d8_k20_10000	Hplus	0.0476±0.009	3.709766±0.224	221768.8±45630.928	152066±26515.367
	Hamerly	0.0176±0.006	3.64883±0.198	221774.3±45607.289	312198.7±117087.573
g_d8_k25_10000	Hplus	0.0356±0.014	3.751563±0.178	193395.8±37780.398	152324.9±46726.611
	Hamerly	0.0164±0.008	3.748829±0.178	193611.9±38100.056	294334.6±138302.778
g_d8_k40_10000	Hplus	0.0448±0.009	3.862111±0.131	225574.9±33274.316	281664.7±45312.257
	Hamerly	0.0316±0.011	3.905471±0.129	224680.3±33672.428	614223.9±209176.104
g_d8_k50_10000	Hplus	0.0448±0.012	3.740627±0.2	199028.7±28904.049	320737.4±77404.518
	Hamerly	0.0372±0.011	3.859767±0.149	198542.8±30169.175	723845.1±213359.496
g_d8_k80_10000	Hplus	0.0476±0.005	3.940624±0.102	210713.2±20103.393	512340.9±44446.587
	Hamerly	0.0576±0.008	3.935548±0.1	210079.7±19589.313	1123747.7±181030.271
g_d8_k100_10000	Hplus	0.0508±0.009	3.917189±0.094	210753.2±23052.039	630584.5±110036.823
	Hamerly	0.0728±0.02	3.918358±0.072	211792.8±22665.94	1520571.4±430074.249
g_d8_k125_10000	Hplus	0.0596±0.008	3.889453±0.103	220454.2±23811.945	846619.9±111250.011
	Hamerly	0.0916±0.015	3.909375±0.095	221668.4±24331.792	1949729.5±314341.808

Comparison of Hplus and Hamerly					
Dataset	Algorithm	Cpu time	Memory	Distance counts	k-means objective
g_d10_k100_10000	Hplus	0.0824±0.018	4.028906±0.123	221537.6±30679.043	649302.3±92323.093
	Hamerly	0.1328±0.028	3.922264±0.054	221002.4±31002.269	1582835.3±354482.81
g_d20_k100_10000	Hplus	0.0772±0.015	4.791016±0.066	506443.8±81405.118	457650.9±90206.845
	Hamerly	0.096±0.028	4.74531±0.131	506435±81432.265	985843±289204.976
g_d30_k100_10000	Hplus	0.1116±0.029	5.487502±0.1	819559.7±129507.774	437549.2±89628.479
	Hamerly	0.1248±0.041	5.563673±0.062	819559.3±129530.371	874897.6±287813.128
g_d40_k100_10000	Hplus	0.1296±0.031	6.341796±0.075	1146937.7±224598.496	394869±82945.593
	Hamerly	0.136±0.038	6.299217±0.095	1146265.7±223721.764	771008.1±216543.332
g_d50_k100_10000	Hplus	0.136±0.014	7.054297±0.079	1449180±243864.017	370002.1±56267.417
	Hamerly	0.1476±0.039	7.083595±0.082	1449188±243860.693	711412±194532.716
g_d60_k100_10000	Hplus	0.1648±0.033	7.889845±0.094	1915897±460413.294	377591.7±81857.301
	Hamerly	0.184±0.065	7.851172±0.066	1919394±461882.867	772861.2±277409.13
g_d70_k100_10000	Hplus	0.1984±0.061	8.605857±0.069	2132603±391151.801	379204.6±84190.092
	Hamerly	0.2288±0.083	8.622656±0.103	2132628±391132.923	757156.5±275037.38
g_d80_k100_10000	Hplus	0.192±0.048	9.403905±0.108	2279123±533832.572	317516.9±83846.493
	Hamerly	0.1924±0.084	9.382814±0.068	2279118±533823.287	576281.4±251335.168
g_d90_k100_10000	Hplus	0.2056±0.025	10.15743±0.088	2618365±413889.853	314542.7±45122.012
	Hamerly	0.2044±0.045	10.175388±0.128	2618365±413889.853	553201.6±121473.277