Summary of Paper: Accelerating Exact K-Means Clustering with Geometric Reasoning
Outline of their Thesis

• Note they use the term “point” to refer to a data point and also a point within a hyper-rectangle
• K-means clustering involves many repetitions of nearest neighbor assignment.
• Worst/best/expected complexity is $O(knmi)$ because at each iteration needs to reassign each and every data point.
• Idea
  – Pre-process the entire data set into a multi-resolution KD-Tree.
  – Each node in the tree represents a hyper-rectangle of the instance space.
  – Store sufficient statistics for each internal node
  – Do batch updating iff we know that all points in the node are “owned” by a single cluster centroid.
• Speed up (at least 100 times) – Too good to be true?
K-Means Algorithm

1. For each point \( x \), find the center in \( C^{(i)} \) which is closest to \( x \). Associate \( x \) with this center.

2. Compute \( C^{(i+1)} \) by taking, for each center, the center of mass of points associated with this center.

Minimizes vector quantization error

\[
\text{distortion}_\phi = \frac{1}{R} \cdot \sum_x d^2(x, \phi(x))
\]

Complexity is \( O(knmi) \)
Lots of “redundancy” in iterations w.r.t instance assignments

DM Spring 2005
Very Brief Introduction to KD-Trees as Used in this Work

- Binary
- Each node represents a hyper-rectangle of the instance space
  - Root node is the entire instance space
  - The “union” of the hyper-rectangles of the children form the parent’s hyper-rectangle.
- Each node contains:
  - $h_{max}$, $h_{min}$, number of points, centroid, sum of Euclidean Norms for all points falling within the hyper-rectangle
  - These are sufficient statistics for updating centroids!
- Internal nodes define a split (like a decision tree)
- Leaf nodes store actual points

DM Spring 2005
The Algorithm - 1

We define the distance \( d(x, h) \) between a point \( x \) and a hyper-rectangle \( h \) to be \( d(x, \text{closest}(x, h)) \). For a hyper-rectangle \( h \) we denote by width\((h)\) the vector \( h^{\text{max}} - h^{\text{min}} \).

- The basic idea: do batch updating. How???

**Definition 1** Given a set of centers \( C \) and a hyper-rectangle \( h \), we define by \( \text{owner}_C(h) \) a center \( c \in C \) such that any point in \( h \) is closer to \( c \) than to any other center in \( C \), if such a center exists.

**Theorem 2** Let \( C \) be a set of centers, and \( h \) a hyper-rectangle. Let \( c \in C \) be \( \text{owner}_C(h) \). Then:

\[
d(c, h) = \min_{c' \in C} d(c', h).
\]

Proof by contradiction

DM Spring 2005
The Algorithm - 2

• Ok. We have the definition of an owner and a state there can only be one…now what …

Definition 3  Given a hyper-rectangle $h$, and two centers $c^1$ and $c^2$ such that $d(c^1, h) < d(c^2, h)$, we say that $c^1$ dominates $c^2$ with respect to $h$ if every point in $h$ is closer to $c^1$ than it is to $c^2$.

Isn’t this a redundant definition of #1???

Lemma 4  Given two centers $c^1, c^2$, and a hyper-rectangle $h$ such that $d(c^1, h) < d(c^2, h)$, the decision problem “does $c^1$ dominate $c^2$ with respect to $h$?” can be answered in $O(M)$ time.

• Keep on comparing cluster centroids to determine if one centroid dominates all others for a hyper-rectangle

DM Spring 2005
Visualizing Centroid Domination

Figure 1: Domination with respect to a hyper-rectangle.

$L_{12}$ is the decision line between centers $c^1$ and $c^2$. Similarly, $L_{13}$ is the decision line between $c^1$ and $c^3$. $p_{12}$ is the extreme point in $h$ in the direction $c^2 - c^1$, and $p_{13}$ is the extreme point in $h$ in the direction $c^3 - c^1$. Since $p_{12}$ is on the same side of $L_{12}$ as $c^1$, $c^1$ dominates $c^2$ with respect to the hyper-rectangle $h$. Since $p_{13}$ is not on the same side of $L_{13}$ as $c^1$, $c^1$ does not dominate $c^3$. 
The Algorithm

Update($h, C$):

1. If $h$ is a leaf:
   
   (a) For each data point in $h$, find the closest center to it and update that center's counters.
   
   (b) Return.

2. Compute $d(c, h)$ for all centers $c$. If there exists one center $c$ with shortest distance:

   If for all other centers $c'$, $c$ dominates $c'$ with respect to $h$ (so we have established $c = \text{owner}(h)$):
   
   (a) Update $c$'s counters using the data in $h$.
   
   (b) Return.

3. Call Update($h_l, C$).

4. Call Update($h_r, C$).
How to do the Batch Update

• Just use hyper-rectangle centers and number of encompassed points
• Like a psuedo-point with non-unit weight
Open Issues

• Worst case analysis of improved algorithm is still $O(knmi)$! Can we determine a bound on the chance of success?

• ???

DM Spring 2005
When the Algorithm Will Work?

Figure 2: Visualization of the hyper-rectangles owned by centers. The entire two-dimensional dataset is drawn as points in the plane. All points that “belong” to a specific center are colored the same color (here, K=2). The rectangles for which it was possible to prove that belong to specific centers are also drawn. Points outside of rectangles had to be determined in the slow method (by scanning each center). Points within rectangles were not considered by the algorithm. Instead, their number and center of mass are stored together with the rectangle and are used to update the center coordinates.
Extensions to the Basic Algorithm

• Black-listing
  – If a centroid $a$ dominates centroid $b$ for $h$, then it will dominate for all descendents of $h$
  – Remove $b$ from the list of centroids to check when decomposing $h$
Experimental Results - 1

<table>
<thead>
<tr>
<th>points</th>
<th>blacklisting</th>
<th>naive</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>50000</td>
<td>2.02</td>
<td>52.22</td>
<td>25.9</td>
</tr>
<tr>
<td>100000</td>
<td>2.16</td>
<td>134.82</td>
<td>62.3</td>
</tr>
<tr>
<td>200000</td>
<td>2.97</td>
<td>223.84</td>
<td>75.3</td>
</tr>
<tr>
<td>300000</td>
<td>1.87</td>
<td>328.80</td>
<td>176.3</td>
</tr>
<tr>
<td>433208</td>
<td>3.41</td>
<td>465.24</td>
<td>136.6</td>
</tr>
</tbody>
</table>

Table 1: Comparative results on real data.
Run-times of the naive and blacklisting algorithm, in seconds per iteration. Run-times of the naive algorithms also shown as their ratio to the running time of the blacklisting algorithm, and as a function of number of points. Results were obtained on random samples from the 2-D “petro” file using 5000 centers.
Sensitivity to Different Data Sets

- How will the running time change for the basic (non-KD-Tree) algorithm as a function of: number of clusters, instances, attributes?

Figure 6: Effect of number of points on the blacklisting algorithm. Running time, in seconds per iteration, is shown as the number of points varies. Each line shows results for a different number of classes.

# points effect is linear  
DM Spring 2005
Results for Black-Listing Algorithm

Figure 4: Effect of dimensionality on the black-listing algorithm. Running time, in seconds per iteration, is shown as the number of dimensions varies. Each line shows results for a different number of classes (centers).

# dimensions effect is super-linear

Figure 5: Effect of number of centers on the black-listing algorithm. Running time, in seconds per iteration, is shown as the number of classes (centers) varies. Each line shows results for a different number of random points from the original file.

# number of centers effect is linear
Approximate Clustering – Not Everyone’s A Winner

Descend down the tree until the ratio of the hyper-rectangle volume to the instance space volume is “small”. Then evenly divide these points amongst all competitors (The ones that haven’t been ruled out by blacklisting).

Our pruning criterion is:

\[ n \cdot \sum_{j=1}^{M} \left( \frac{\text{width}(h)_j}{\text{width}(U)_j} \right)^2 \leq d^i \]

where \( n \) denotes the number of points in \( h \), \( U \) is the “universal” hyper-rectangle bounding all of the input points, \( i \) is the iteration number, and \( d \) is a constant, typically set to 0.8.

DM Spring 2005
Approximate Clustering - Results

Note, no longer equivalent to standard k-means
As expected, run time is better, but distortion is higher.

Figure 7: Runtime of approximate clustering. Running time, in seconds per iteration, is shown as the number of points varies. Each line stands for a different algorithm.

Figure 8: Distortion of approximate clustering. Distortion for approximate and exact clustering. Each line stands for a different algorithm.