Accelerated K-Means Clustering in Metric Spaces

Andrew Smellie*

ArQule Inc., 19 Presidential Way, Woburn, Massachusetts 01801

Received February 26, 2004

The K-means method is a popular technique for clustering data into k-partitions. In the adaptive form of the algorithm, Lloyds method, an iterative procedure alternately assigns cluster membership based on a set of centroids and then redefines the centroids based on the computed cluster membership. The most time-consuming part of this algorithm is the determination of which points being clustered belong to which cluster center. This paper discusses the use of the vantage-point tree as a method of more quickly assigning cluster membership when the points being clustered belong to intrinsically low- and medium-dimensional metric spaces. Results will be discussed from simulated data sets and real-world data in the clustering of molecular databases based upon physicochemical properties. Comparisons will be made to a highly optimized brute-force implementation of Lloyd's method and to other pruning strategies.

INTRODUCTION

Clustering is defined as "a group of the same or similar elements gathered or occurring closely together"[1] and is an essential part of understanding large collections of data. Clustering is often used as a method of data reduction, where a large data set can be reduced to smaller numbers of representatives drawn from clusters. It plays a role in many fields of research including, but not limited to, image processing,[2] statistics,[3] bioinformatics,[4] demographics,[5] and economics.[6]

There are many clustering techniques, each with their merits and drawbacks, but one of the most popular is the k-means clustering method.[7] This paper examines the k-means method in detail and proposes a metric pruning method to rapidly speed up the basic k-means algorithm. It is outside the scope of this paper to discuss the merits of the k-means algorithm itself; this has been discussed extensively elsewhere.[8−10] It will be pointed out however that the techniques described in this paper can be extended to incorporate some of the algorithmic improvements described in the literature to the basic k-means method.

In its most general form, the k-means method generates a set of k cluster *d*-dimensional centroids {C} for a set of points {P} in real *d*-dimensional space $\mathcal{R}^d$, for any integer k (k ≥ 2). The cluster centroids define a partition of the points in the space because a point P_i is assigned to a cluster center C_j if C_j is the closest centroid to P_i; that is, if the following condition is true

$$D(P_i,C_j) = \mathrm{MIN}_K\,[D(P_i,C_k)] \qquad \text{(Eq. I)}$$

where $D(P_i, C_j)$ = distance from point P_i to cluster centroid C_j.

The quality of the k-means clustering is often expressed as the sum of squares of distances of each point from its cluster center

$$E = \sum_{j=1}^{k} \sum_{P_i \in C_j} |P_i - C_j|^2 \qquad \text{(Eq. II)}$$

Generally speaking, the lower the value of E in Eq. II, the better the clustering. The term has the property that it penalizes large violations more than small ones due to the squared term. This value will be used in this paper only as part proof that the pruning techniques converge to the same solution as using the brute force method by converging to a solution with the same value of E (within numerical round off). Before discussing any algorithms in detail, some terms must be defined: N = the number of points being clustered, k = the number of clusters generated, d = the dimensionality of the space, {P} = the set of all points being clustered, P_i = the ith point being clustered, {C} = the set of all cluster centroids, C_j = the jth cluster centroid, $\{P\}_{Cj}$ = the set of all points closest to the cluster center C_j, $D(P_i, C_j)$ = the distance from the ith point to the jth cluster centroid, F = the final number of iterations to convergence, $\langle NC\rangle$ = the mean number of points per cluster, and E = the sum of squared distances of each point to its cluster center.

While there are many variants, the basic adaptive k-means method proceeds in four stages as described in Algorithm 1 below. This algorithm converges to a *local* minimum, as measured by the cost function E. The de facto method of addressing this problem is to use random restarts in choosing starting cluster centers in Step 1 of Algorithm 1. There are many variants of k-means clustering that attempt to get start points closer to optimal in Step 1,[11−13] but these are orthogonal and complementary to the pruning method described here. This paper uses the de facto k-means clustering method with 10 random restarts. It is more typical to use more restarts, but this will just magnify the effect of improving the run time when using just one iteration.

* Corresponding author phone: (781)994-0559; fax: (781)994-0679; e-mail: asmellie@arqule.com.

Algorithm 1. The Adaptive k-Means Clustering Algorithm

(1) Select a set of initial cluster centers {C}.

(2) Assign all points {P} to their respective closest cluster centroid C_j, such that Eq. I is true for all points P_i

(3) Recompute each cluster centroid j as the mean position of all points P_i currently assigned to that cluster

(4) If not converged, go to step 2

There are many suggested methods of implementing step 1, including pure random selection,[7] heuristic sampling and preclustering,[11] and bootstrap sampling.[13] This paper will focus on the optimization of step 2 (usually the overwhelmingly most expensive step, accounting for more than 90% of the total run time) and thus use the simplest method of initial cluster selection, namely random selection. It is pointed out that any heuristic improvement on the selection of starting cluster centroids with a view to improve the final quality is complementary to this work and can only improve on the results presented here.

The algorithmic complexity of step 2 is N.k. Because the overall algorithm is iterative, step 2 will be executed F times. Thus the total number of distance computations in step 2 is F.N.k. Any efficiency achieved here will greatly improve the efficiency of the overall algorithm.

The algorithmic complexity of step 3 is $\langle NC \rangle$.k. In this implementation, this is done brute-force with no attempt to increase the efficiency. However, it is generally step 2 that dominates so this is not an egregious factor.

There are many choices of convergence in step 4. One of the most common is a sufficiently small change in the value of the cost function E. In this work, a slightly different criterion is used, namely when the root mean-square difference in the centroid positions falls below a user-defined threshold between successive iterations. This is a preferable metric because (a) it removes the need to compute E until the very end, when a quality assessment of the clustering is required and (b) it is more physically interpretable and easier to supply meaningful convergence values. For simulated data points in a box with side lengths *l*, the convergence distance is set to 0.001 * *l*.

THE VANTAGE-POINT (VP) TREE

As already discussed, the most time-consuming part of Lloyd's algorithm[14] is the inner loop when a set of cluster centroids has been computed either from the initialization step or computed as the centroid of the current cluster members. Each point P_i in the space being clustered must be assigned to its closest cluster center. More formally, for each point P_i, find the cluster C_j for which the condition in Eq. I is true. Implemented naively, this step has complexity N.k. Intuitively an implementation of this step might involve taking every point P_i and searching for its closest cluster center. It has been shown in previous work[15] that significant cost savings can be made if the problem is inverted such that for each cluster center C_j, the set of points closest to it are found, $\{P\}_{Cj}$. Typically, the set of points being clustered is preprocessed into a data structure that facilitates rapid nearest-neighbor searching. Since there are more points being clustered than there are cluster centers (if this is not true, why do clustering?), this approach can greatly increase the overall clustering performance. The method described in ref 15 places the points being clustered into a k−d tree to accelerate the searching for nearest neighbors for each candidate cluster centers. This approach is sound, and the vantage point tree is an alternate method of speeding up the search for nearest neighbors. It has the extra metrit of being effective when the intrinsic dimensionality of the data is low, even if the explicit dimensionality is high.[16]

The vantage-point (VP) tree has previously been described as a data structure that partitions up points in a metric space based on their distances from some reference, or *vantage*, point. It has been shown to not be prone to inefficiencies in near-neighbor lookup for data that is *intrinsically* low-dimensional but *apparently* high-dimensional (*unlike* the kd tree). The method has been well described elsewhere[17,18] and only enough of the algorithm will be discussed here to sufficiently describe the pruning method of this paper. Briefly, the VP-tree is constructed as follows.

Algorithm 2. Construction of a Vantage-Point Tree. Given a set of points {P}

(1) Choose a random point P_v and remove it from {P}

(2) Compute the distances of all points {P} to P_v

(3) Sort the points {P} by their distances to P_v

(4) Choose the point P_w as the point from {P} with the median distance r to P_v

(5) Divide {P} into two sets $\{P\}_L$ and $\{P\}_R$ consisting of the set of points with distances to P_v that are lower or equal to and higher than the median distance, respectively.

(6) Make P_v a new node in the vantage point tree and place $\{P\}_L$ in the left branch of the tree and $\{P\}_R$ in the right branch of the tree. Store (a) the coordinates of P_v and (b) the median distance r

(7) *Also store on the new vantage point node, the upper* $U_{n(max)}$ *and lower* $L_{n(max)}$ *bounds on the distance of points in the left branch* $\{P\}_L$ *to* P_v

(8) If $\{P\}_L$ has too many points, set $\{P\} = \{P\}_L$, select a new point P_v, and go to 2

(9) If $\{P\}_R$ has too many points, set $\{P\} = \{P\}_R$, select a new point P_v, and go to 2

A simple example of vp-tree construction is shown in Figure 1 where a set of 12 points {P} is embedded in 2 dimensions. A point is selected at random (point #1) and becomes the *vantage point* P_v, which is subsequently removed from set {P}. The distance from the vantage point is computed to all the points in {P}, which is subsequently sorted by increasing distance from P_v, and the median distance from points in {P} to P_v is recorded (i.e. distance r_1). All points in {P} that have a distance $\leq r_1$ are placed in the left node of the vp-tree and all points with distance $> r_1$ are placed in the right node of the vp-tree. Figure 2 shows the structure of the vp-tree. The root node stores the coordinates of the vantage point P_v and the median distance used to split the data (i.e. r_1). Now the data have been divided into two nodes, the algorithm proceeds recursively in the left and right branches by, for each node, choosing a new vantage point and dividing the set of points again. Thus for a vp-tree of depth D_v, each leaf of the tree holds $|\{P\}|/2^{D_v}$ points. In the example of Figure 1, the final result is that there is a root split, then a subsequent split in the left and right branches resulting in the tree shown in Figure 2.

The steps shown in italics in Algorithm 2 are not strictly part of the vp-tree construction but are required for the pruning using the k-mean clustering.

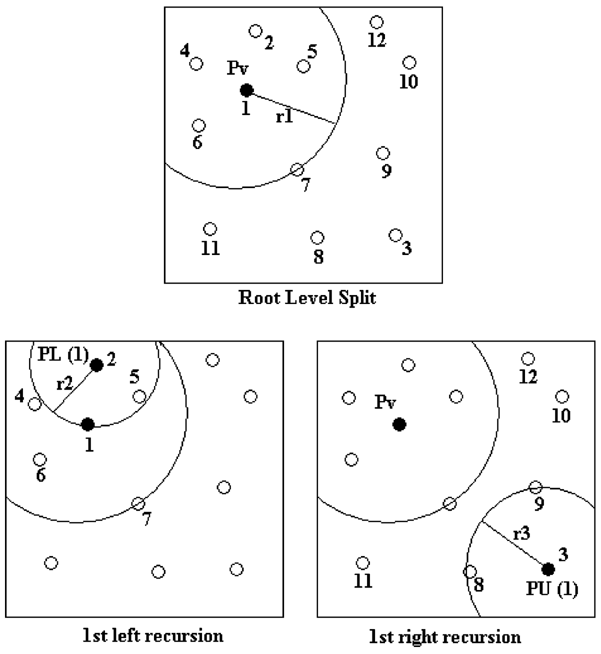

Figure 1. Constructing the vantage point tree.

From the construction of the vp-tree, the most important property of the tree that facilitates nearest neighbor searching, and the performance increase in clustering reported here, can now be clearly stated:

Property (i) For any node in the vp-tree, where the coordinates of its vantage point P_v and the median vantage distance r_v are stored, it is guaranteed that all points reached by traversing the left branch have a distance $r \le r_v$ to point P_v, and all points reached by traversing the right branch have a distance $r > r_v$ to point P_v.

PRUNING THE INNER K-MEAN LOOP USING VANTAGE-POINT (VP) TREES

Recalling step 2 of the k-means method (Algorithm 1), the need is to assign all points {P} to their respective closest cluster centroid C_j. If the problem is posed as a search for all points closest to a particular cluster center, using Property (i) of the vantage point tree, significant pruning can be achieved using the triangle inequality in metric spaces. A *metric space* is a set *X* together with a function *d* (called a *metric* or "*distance function*") which assigns a real number $d(x, y)$ to every pair $x, y \in X$ satisfying the properties (or *axioms*):

Property (ii) $d(x, y) \ge 0$ and $d(x, y) = 0 \Leftrightarrow x = y$

Property (iii) $d(x, y) = d(y, x)$

Property (iv) $d(x, y) + d(y, z) \ge d(x, z)$

First, some terms will be defined: $\{C\}_n$ = the set of candidate cluster centers viable at node n of the vp-tree, C_j = the jth cluster center, P_n = the vantage point stored at node n, r_n = median distance stored at node n after vp-tree construction, $\{L\}_n$ = the set of points stored in the left branch of node n, $\{R\}_n$ = the set of points stored in the right branch of node n, $D(P_n, C_j)$ = the measured distance from jth candidate cluster center to P_n, $L_{n(min)}$ = the minimum distance from P_n to any node in the left branch, $U_{n(max)}$ = the maximum distance from P_n to any node in the left branch, $LW(\{L\}, C_j)$ = the lower bound of distance from any point in $\{L\}_n$ to C_j, and $UP(\{L\}, C_j)$ = the upper bound of distance from any point in $\{L\}_n$ to C_j.

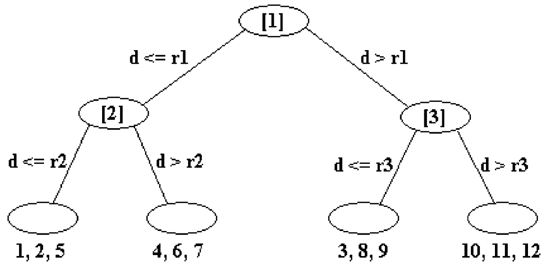

Figure 2. Traversing the vantage point tree.

The algorithm proceeds by first constructing a vantage point tree for all points {P} being clustering using Algorithm 2. In this implementation an initial vantage point is selected at random (step 1). Upon dividing the set of points by their median distance from the current vantage point (steps 2, 3, 4, and 5), a new vantage point is selected on the left and right subsets (steps 6 and 7) using a simple heuristic: a new vantage point is selected by randomly sampling points from the subset and choosing one that is furthest away from any vantage point chosen to date. This has the effect of ensuring that the vantage points are well separated as shown in Figure 1. Empirical testing on data sets from 1000 to 1000000 points has shown that the maximal pruning occurs when the tree is constructed to depths 12−17. In all examples shown here, the tree was constructed to a depth of 17.

At the beginning all cluster centers {C} are viable closest cluster centers for all points {P}. Figure 3 shows the pruning step used to definitively eliminate cluster centers as being candidates for the closest cluster center to the set of points in the left branch of the vp-tree. The distance from each candidate cluster center C_i is measured to the vantage point at the root of the vp-tree P_n, (i.e. all distances $D(P_n, C_i)$). Recall that in the construction of the vp-tree, the upper bound on the distance from any point in the left branch {L} to the vantage point is stored (i.e. $U_{n(max)}$). Thus candidate cluster centers can be definitively eliminated as being the closest cluster center to any point in the left branch by using the *triangle inequality*. From Figure 3 we can deduce the upper (UP) and lower bounds (LW) on the distance from a candidate cluster center to *any* point in the left branch as follows.

$$LW(\{L\}, C_j) = D(P_n, C_j) - U_{n(max)} \quad \text{(Eq. III)}$$

$$UP(\{L\}, C_i) = D(P_n, C_i) + U_{n(max)} \quad \text{(Eq. IV)}$$

A candidate cluster center j violates the triangle inequality if it can be shown for *any* pair of candidate cluster centers i and j that

$$LW(\{L\}, C_j) > UP(\{L\}, C_i) \text{ for any } i \ne j \quad \text{(Eq. V)}$$

To implement a test for Eq. V means that at each node of the vantage point tree the distance of each candidate cluster center must be measured to the vantage point and upper and lower bounds computed from Eq. III and Eq. IV respectively. To naively test for the condition of Eq. V would involve

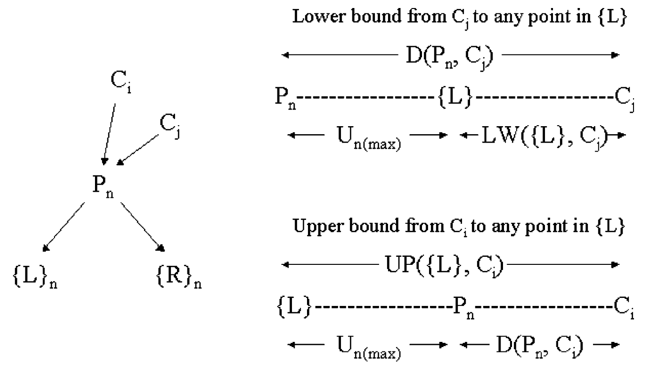
Lower bound from C_j to any point in {L}
$D(P_n, C_j)$
P_n-----------------{L}-----------------C_j
$U_{n(max)}$
$LW(\{L\}, C_j)$
C_i
C_j
P_n
$\{L\}_n$
$\{R\}_n$
Upper bound from C_i to any point in {L}
$UP(\{L\}, C_i)$
{L}-----------------P_n-----------------C_i
$U_{n(max)}$
$D(P_n, C_i)$

Figure 3. The pruning step using the vantage point tree.

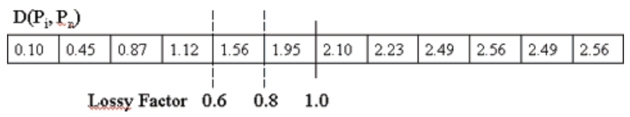

Figure 4. Approximations made during the construction of the VP tree.

comparing all pairs of distance bounds among the candidate cluster centers, an operation that is quadratic in the number of candidate centers. This can be converted to a linear time operation by first finding the smallest upper bound U of any candidate cluster center C_i and then finding all cluster centers C_j whose lower bound is greater than U. These cluster centers can be eliminated from further consideration.

This pruning operation proceeds recursively down the vantage point tree until a leaf is encountered. At this stage any remaining candidate cluster center must be compared against *all* points in the leaf brute-force fashion.

LOSSY CLUSTERING

A careful analysis of the construction of a vantage point tree points toward a method of "lossy" clustering in which approximations can be made in the construction of the tree which, while sacrificing rigorous proof of correctness, can offer performance increases of up to 50% for very little degradation in the final quality of the solution. Consider the situation in Figure 4, which summarizes steps (2) to (5) of Algorithm 2 when the VP tree is being constructed. The array in Figure 4 stores the distance of the ith point being clustered to its parent vantage point node during tree construction and has been sorted by increasing distance to the parent vp node. In the conventional construction of a vantage point tree, the set of points is divided at the median of the distances. In this example, all points with a median distance of less than or equal to 1.95 are allocated to the left child of the parent node and all points with a distance greater than 1.95 are allocated to the right child. The distance 1.95 becomes the $U_{(max)}$ in step 7 of Algorithm 2. Referring to Figure 1, $U_{(max)}$ is the radius of the sphere such that it is guaranteed that all points in the left branch of the VP tree are within distance $U_{(max)}$ of the parent vantage point node. It can be seen from Eq. III and Eq. IV that smaller values of $U_{(max)}$ will favor more aggressive pruning during the clustering. This concept is formalized by defining the *lossy factor* (*lf*) as $U_{(max)} = D(P_n, P_i)$, where P_n = the current vantage point, P_i = the ith point in current level during VP tree construction, i = *lf* * (N/2), and N = the number of nodes at current stage of VP tree construction.

Thus the value of $U_{(max)}$ stored at this point in the vantage point tree is the distance in array of Figure 4 taken from the median entry (when i = N/2) or from an entry with a smaller value with I = *lf* * (N/2). From the example in Figure 4, a lossy factor of 0.8 will produce a value of $U_{(max)}$ of 1.56 (corresponding to entry *i* = 0.8 * 12/2 = 4); a lossy factor of 0.6 will produce a value of 1.12 (*i* = 0.6 * 12/2 = 3). The effect of a lossy factor of less than 1.0 is to shrink the hypersphere around the defining vantage point. In the worst-case scenario, the test defined in Eq. V might erroneously eliminate a candidate cluster center from further consideration if the perceived $U_{(max)}$ is too small. Thus, the true closest cluster center might be prematurely eliminated. This risk is mitigated by initially setting the lossy factor to < 1.0 and allowing Algorithm 1 to converge, then setting it back to 1.0 (i.e. with no approximations) and repeating Algorithm 1.

CONSTRUCTION OF TEST DATA

To objectively test the effectiveness of the pruning requires simulated data whose properties are well understood. In this study, simulated data were generated in d-dimensional space for N points in k clusters as follows:

Algorithm 3 Generating Simulated Data

```
(1) $n = 0$
(2) Generate k random cluster centers in d-dimensional space in a box with sides ranging from $X_{min}$ and $X_{max}$.
(3) Select a random cluster center $k_j$
(4) Select a random distance r from a normal distribution of variance v
(5) Generate a random point $P_n$ on the locus of the point at distance r from $k_j$
(6) $n = n + 1$
(7) If $(n < N)$ go to 3
```

The parameters that affect the distribution of data are X_{min} = 0.0, X_{max}= 10.0, k= 50, 100, 250, 500, d = 2, 5, 10, N = 1000, 5000, 10000, 50000, 100000, 500000, 1000000, and v = 3.0, 8.0.

For molecular data**,** the DiverseSolutions[19] (DVS) package is a well-known tool for projecting molecules into low dimensional spaces. The VP method was tested with the set of compounds contained in the Asinex[20] catalog distributed with Pipeline Pilot.[21] BCUT descriptors were successfully computed for 135788 compounds (from a total of 137799). These compounds were projected into a 5-dimensional space (a DVS chemspace), and clustering was performed in this space. It is understood that the DVS chemspace tries to spread out the points as much as possible and so should present a challenging clustering problem.

RESULTS

The simulated data used to test the new pruning algorithm are shown in 2D in Figure 5 for cluster variances of 3.0 and 8.0 respectively (i.e. the variance υ used to generate the simulated data in Algorithm 3). It is expected that the simulated data of higher variance (8.0) would take longer to cluster because the points are more scattered. This is confirmed by the plot in Figure 6 where cluster run times are reported for simulated data at fixed dimension (5), number of points (1M), and number of generating clusters (500). It can be seen that higher variances generally mean

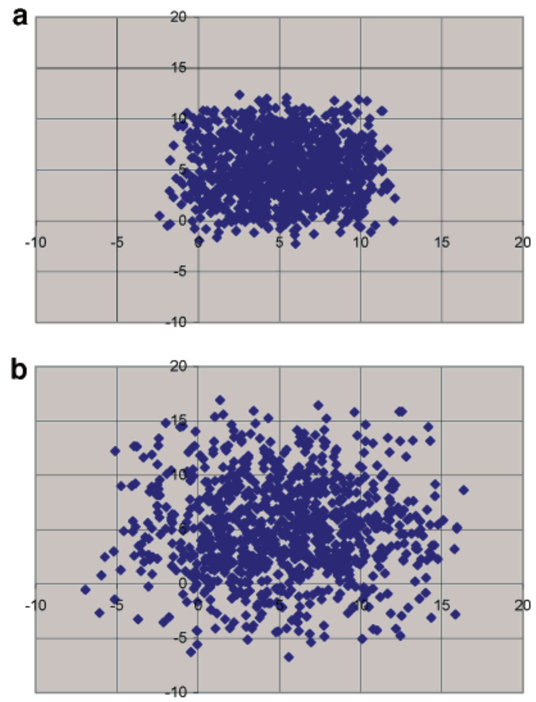

Figure 5. Examples of simulated cluster data. Cluster time vs variance used to generate simulated data. (a) Simulated cluster data of 1000 points in 2 dimensions with variance = 3.0 and 100 clusters. (b) Simulated cluster data of 1000 points in 2 dimensions with variance = 8.0 and 100 clusters.

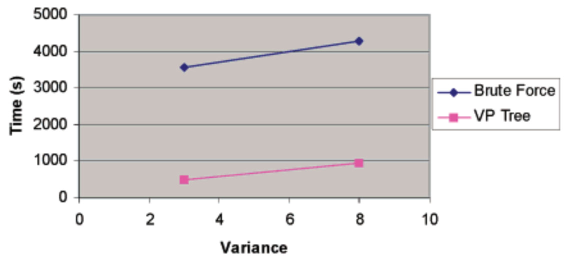

Figure 6. Cluster time vs variance used to generate simulated data. Point dimension = 5, number of generating clusters = 500.

longer clustering times. This can be understood from step 7 of Algorithm 2 where upper and lower bounds on a subspace division are constructed. More scattered points means the derived bounds would not be as tight as for more clustered points. In this study, the larger variance of 8.0 was used to give a more exacting test of the method.

The times required to complete the clustering are shown in Figure 7 (2D data), Figure 8 (5D data), and Figure 9 (10D data). In 2D and 5D, the vantage point clustering is always faster than the brute force method. For 2D, the difference in run times varies from a factor of 2 for 250 clusters and 1000 points, to a factor of 25 for 500 clusters and 1000000 points. More significantly, the difference in run times between the brute force method and the VP method increases as the product of the number of clusters and the number of points increases (indicated by the slope of the log(N)/log(T) plots). It is expected that this trend would continue for even larger numbers of points and/or clusters. In 10 dimensions, the difference in run times (i.e. by a factor > 2) only occurs when the number of points exceeds about 100000 points. This is consistent with previous experience with metric-pruning algorithms: the effectiveness decreases as the dimensionality increases. Despite this observation, it is clear

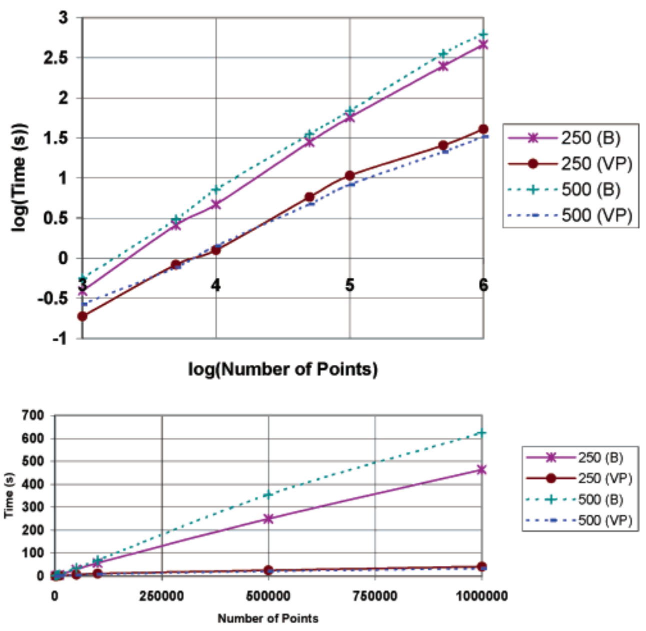

Figure 7. Plot of cluster time vs number of points in 2 dimensions. *n* (B) denotes the performance for generating *n* clusters by brute force. *n* (VP) denotes the performance for generating *n* clusters using the vp tree. The tree depth maximum searched was 17. Point dimension = 2, variance = 8.0.

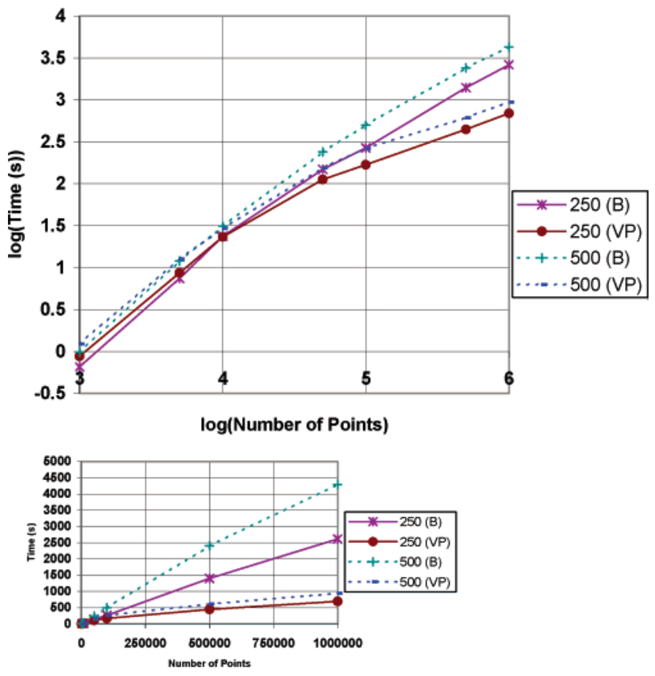

Figure 8. Plot of cluster time vs number of points in 5 dimensions. Point dimension = 5, cluster variance = 8.0.

from the plot in Figure 9 that for large numbers of points, the difference in run times is clearly divergent for large numbers of points: the VP method is superior to the brute force method.

Figure 10 shows the dependence of cluster run times with the VP method as a function of tree depths 14, 17, and 20. The data point at depth 0 reports the timings for the brute force method for comparison. There will be a natural "best" value for a given data set. The deeper the tree during the search, more tests will be required to eliminate candidates according to the test defined by Eq. V, but potentially fewer tests must be made at the leaves of the tree, where nearest

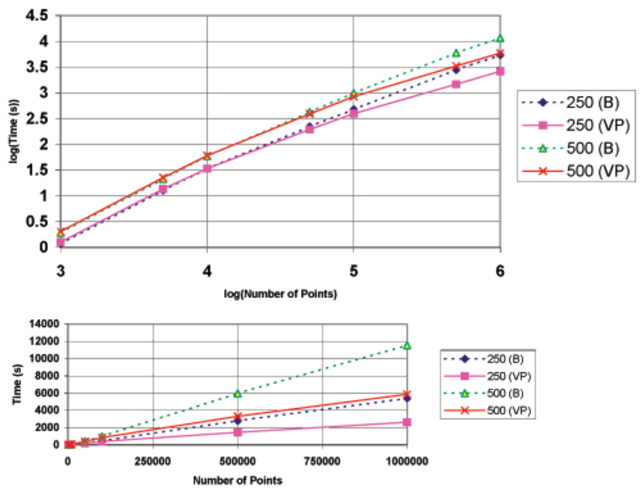

Figure 9. Plot of cluster time vs number of points in 10 dimensions. Point dimension = 10, cluster variance = 8.0.

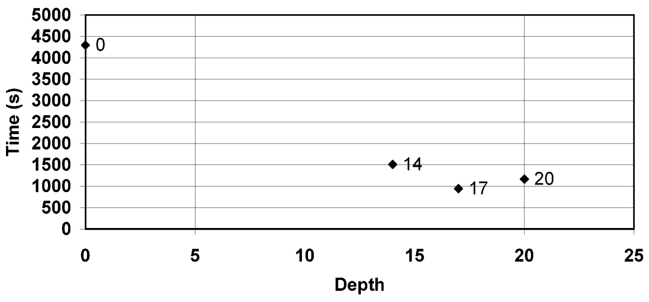

Figure 10. The effect of tree depth on cluster run times. Point dimension = 5, variance = 8.0; number of points = 1M, number of generating clusters = 500.

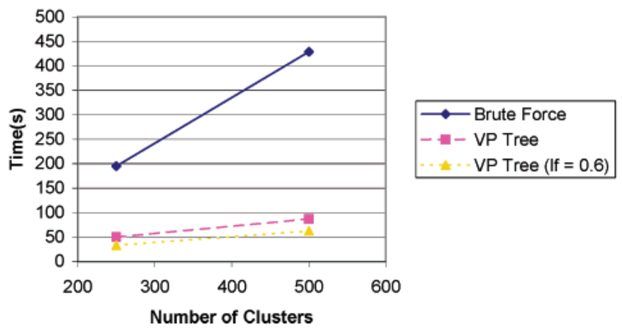

Figure 11. Clustering Asinex data in 5D BCUT coordinate space.

neighbors of all points in the leaf of the tree must be found from a set of surviving candidates. In this study, a depth of 17 generally performed the best, especially for large numbers of points. This corresponds to a VP tree with 2^{17} (~131K) nodes. For a million points, each leaf has an average of 8 points.

The results for clustering of the 135788 Asinex molecules in 5D BCUT space are shown in Figure 11. For 250 clusters, the VP method outperforms the brute force method by a factor of 4 (196 s vs 50 s respectively), and for 500 clusters the difference is a factor of 5 (440 s vs 85 s respectively). Running with a lossy factor of 0.6 resulted in a speed increase of factor of 8 (440 s vs 52 s). There was no appreciable degradation in the quality of the final answer as measured by Eq. II. The final value of the cost function when run with a lossy factor of 1.0 and 0.6 never differed by more than 2%.

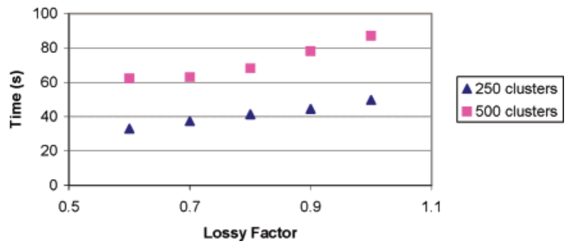

Figure 12. Lossy clustering performance for the Asinex data set.

The effect of changing the lossy factor is shown in Figure 12 for the Asinex data, where clustering into 250 and 500 clusters was performed using lossy factors of 1.0, 0.9, 0.8, 0.7, and 0.6. As expected we see a linear increase in performance since reducing the lossy factor means that fewer distance calculations are being performed. The final data point for 500 clusters and lossy factor 0.6 shows a slight increase in run time. Upon investigation the cause of this was the algorithm requiring more iterations to converge. All solutions found were with 2% of the brute force best solution. Based upon this and other results, a lossy factor of 0.7−0.8 is recommended as a tradeoff among stability, quality, and run time.

CONCLUSIONS

A new method for accelerating k-means clustering has been presented that demonstrates very good performance in low (2) (25x speed-up) to medium (5) (8x speed up) dimensional spaces. The method produced *identical* solutions for the same starting data when run in approximation-free mode, as measured by a standard cost function. An approximation was introduced that gave an incremental improvement in performance of up to 50%. The quality of solutions found differed from the “true” solutions by no more than 2%.

It is expected that the clustering methods presented here will be useful in the postprocessing of large quantities of molecular data, especially since projection methods have proved very popular and methods are available for representing very large collections of molecules in low dimensional spaces.[18]

The improvements outlined in this paper are complementary to other methods of increasing performance, such as a more careful choice of starting point, and should prove useful in rapidly generating clusters for large databases. In future work, extensions to the algorithm are being tested for higher dimensional spaces.

ACKNOWLEDGMENT

The author wishes to acknowledge David Hartsough for proofing the manuscript and Anton Filikov and Rocio Palma for many valuable discussions.

REFERENCES AND NOTES

(1) http://dictionary.reference.com/search?q=clustering.
(2) Estlick, M.; Leeser, M.; Theiler, J.; Szymanski, J. Algorithmic transformations in the implementation of K-means clustering on reconfigurable hardware, FPGA, 2001; pp 103−110.
(3) Hartigan, J.; Wong, M. *A K-means clustering algorithm*. *Appl. Statistics* **1979**, *28*, 100−108.
(4) Bickel, D. R. Robust Cluster Analysis of Microarray Gene Expression Data with the Number of Clusters Determined Biologically. *Bioinformatics* **2003**, *19*, 7, 818−824.

(5) Shepard, D. *How To Tackle A Cluster Analysis*, a report from Direct Marketing Business Intelligence, May 15th, 2003.
(6) Da, N. *Hedge Fund Classification using K-means Clustering Method, #284*, Computing in Economics and Finance 2003.
(7) MacQueen, J. B. Some Methods for classification and Analysis of Multivariate Observations, *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability*; Berkeley, University of California Press: 1967; Vol. 1, pp 281−297.
(8) Mashor, M. Y. *Improving the Performance of K-Means Clustering Algorithm to Position the Centers of RBF Network,* International Journal of the Computer, The Internet and Management, 6, 2, May-August 1998.
(9) Likas, A.; Vlassis, N.; Verbeek, J. J. The Global K-Means Clustering Algorithm. *Pattern Recognit.* **2003**, *36*(2), 451−461.
(10) Cheung, Y.-M. k*-means: A new generalized k-means clustering algorithm. *Pattern Recognit. Lett.* **2003**, *24*, 2883−2893.
(11) Bradley, P. S.; Fayadd, U. M. *Refining Initial Points for K-Means Clustering*, Proc. 15th International Conf. on Machine Learning, 1998.
(12) Fayyad, U. M.; Reina, C. A.; Bradley, P. S. Initialization of Iterative Refinement Clustering Algorithms. *Knowledge Discov. Data Mining* **1998**, 194−198.
(13) Davidson, I.; Satyanarayana, A. *Speeding up k-means Clustering by Bootstrap Averaging*, IEEE Data Mining Workshop on Clustering Large Data Sets, 2003.
(14) Lloyd, S. P. Least Squares Quantization in PCM. *IEEE Trans. Inf. Theory* **1982**, *28*, 129−137.
(15) Alsabti, K.; Ranka, S.; Singh, V. *An Efficient K-Means Clustering Algorithm*, IPPS: 11th International Parallel Processing Symposium, 1998.
(16) Xu, H.; Agrafiotis, D. K. Nearest neighbor search in general metric spaces using a tree data structure with a simple heuristic. *J. Chem. Inf. Comput. Sci.* **2003**, *43*, 1933−1941.
(17) Yianilos, P. N. Excluded Middle Vantage Point Forests for Nearest Neighbor Search, NEC Research Institute Report, Princeton, NJ, July 1998.
(18) Yianilos, P. N. *Locally Lifting the Curse of Dimensionality for Nearest Neighbor Search*, Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2000; 361−370.
(19) DiverseSolutions, Optive Research, Inc. 7000 North Mopac Expressway 2nd Floor, Austin TX 78731.
(20) Asinex Ltd., 6 Schukinskaya Street, Moscow 123182, Russia.
(21) Pipeline Pilot Server Version 3.0.5.0, Scitegic Corp., 9665 Chesapeake Drive, Suite 401, San Diego, CA 92123-1365.

CI0499222