k-Means Projective Clustering *

Pankaj K. Agarwal Nabil H. Mustafa

Department of Computer Science,
Duke University, Durham, NC 27708-0129, USA.
{pankaj, nabil}@cs.duke.edu

ABSTRACT

In many applications it is desirable to cluster high dimensional data along various subspaces, which we refer to as *projective clustering*. We propose a new objective function for projective clustering, taking into account the inherent trade-off between the dimension of a subspace and the induced clustering error. We then present an extension of the k-means clustering algorithm for projective clustering in arbitrary subspaces, and also propose techniques to avoid local minima. Unlike previous algorithms, ours can choose the dimension of each cluster independently and automatically. Furthermore, experimental results show that our algorithm is significantly more accurate than the previous approaches.

1. INTRODUCTION

Given a set of points in multidimensional space, the goal of clustering is to compute a partition of these points into sets called *clusters*, such that the points in the same cluster are more similar than points across different clusters. This general technique of clustering is very valuable in analyzing large data, and thus has found applications in many areas such as data mining, indexing, pattern recognition, and trend analysis and classification. For an overview to the topic, we refer the reader to [10, 15].

An important aspect of clustering is the notion of similarity, or "goodness" of a cluster. Similarity between two points is a function of the Euclidean distance between the points. Similarity between clusters is then defined as a function of the similarity of the points they contain, for example measuring inter-cluster distances (e.g. *single-linkage* (*complete-linkage*), defined as the distance of the closest (farthest) pair of points in the two clusters) or measuring distances using intra-cluster distances (e.g. k-median, k-means, min-sum clustering). A number of practical, efficient methods, such as CLARANS [18], BIRCH [20], CURE [9], OptiGrid [14], have been proposed for clustering based on Euclidean distances between points in the full dimensional space.

*Work has been supported by NSF under grants CCR-00-86013 EIA-98-70724, EIA-99-72879, EIA-01-31905, and CCR-02-04118 and by a grant from the U.S.-Israeli Binational Science Foundation.

PODS 2004 June 14-16, 2004, Paris, France.

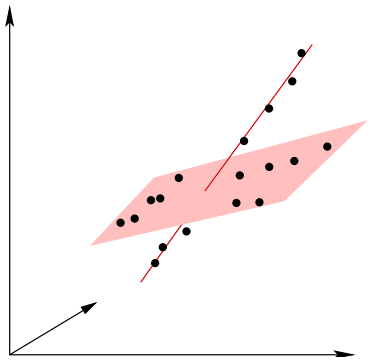


Figure 1: A set of points in $\mathbb{R}^2$ that cluster well in different dimensional non-orthogonal subspaces.

In clustering points lying in high dimensional spaces, formulating a desirable measure of "similarity" is more problematic. Simply looking at the Euclidean distance between two points is not good enough. Recent research shows that for high dimensional spaces, computing the distance by looking at all the d-dimensions is often useless, as the farthest neighbor of a point is expected to be almost as close as its nearest neighbor [13, 6]. This indicates a crucial difference in the input — data is typically quite sparse (as a function of the number of dimensions) in high dimensions, and the points cluster in much lower-dimensional *subspaces* (Figure 1). Therefore weighing all the dimensions equally while computing the distance gives an unreliable measure of similarity.

Two main approaches have been used to overcome this phenomenon. The idea behind early approaches was to first project all the points to a lower dimensional subspace so that the distortion between the distances is bounded by a pre-specified error. Then any of the previous clustering algorithms is used on the projected set. Many methods have been developed for computing the best projection subspace, including random projection-based methods (e.g. Johnson-Lindenstrauss Lemma [16]) and principal component analysis [17].

However, projecting all the points in a single lower subspace faces an obstacle: what if different clusters lie in different subspaces? See Figure 1 for an illustration of this problem. Therefore, to compute clusters in different lower dimensional subspaces, recent work has focused on *projective clustering*, defined as follows: given a set P of points in $\mathbb{R}^d$ and an integer k, partition P into k subsets that best classify P into lower dimensional subspaces according to some objective function. Instead of projecting all the points in the same subspace, this allows each cluster to have a different subspace

associated with it. It will be this problem that will be the focus of this paper.

In general, it is highly desirable that any technique for projective clustering satisfy the following two criteria:

- There should be a formal definition of what constitutes an optimal projective clustering. This definition must satisfy a number of objectives: it must be robust, geometrically meaningful and must capture the trade-off between dimensionality of the subspaces and the corresponding induced error.
- Furthermore, any algorithm must be able to (i) achieve good quality clustering, (ii) allow each cluster to have a different projective subspace of variable dimensionality (iii) automatically detecting the subspace dimensionality of each cluster, (iv) work well for detecting clusters of widely varying sizes, (v) work fast in practice.

Previous work. Previous work on projective clustering has addressed various aspects of the above list. Algorithms for computing projective subspaces have been proposed and used in indexing [7] and pattern discovery (PROCLUS [4], ORCLUS [3], DOC [19]). CLIQUE [5], a related method, enumerates all the dense subspace regions in the data and hierarchically combines them to output rectangular projections with overlapping points (with the execution time having an exponential dependence on the dimensionality of the points). We now explain two of these methods in more detail.

Aggarwal and Yu [3] use hierarchical clustering to compute projective clusters in different subspaces. Given the number of clusters k, and their dimension q, their algorithm initially computes a large number of clusters of dimension d, and then hierarchically merges the closest clusters (as defined by some criteria) while decreasing the dimensionality of the clusters by a constant factor. After a calculated number of such merges, the number of remaining clusters are k, and the dimensionality of each cluster has been reduced to the required dimensionality q. Though each cluster is associated with a different subspace, the algorithm requires the dimension of each cluster to be the same (i.e. q), and more importantly, must be specified by the user.

An attempt to overcome the shortcomings of the above approach, proposed by Procopiuc *et al.* [19], is to use an iterative method to extract the best projective cluster: the algorithm finds the best projective cluster from the remaining points by guessing points belonging to the optimal cluster (via random sampling), and then computes the best dimensions associated with the cluster. The algorithm has the advantage that it allows each cluster to have a different number of dimensions. Furthermore, the algorithm computes the best dimensionalities automatically. A major disadvantage of the algorithm is that it is restricted to finding only clusters in orthogonal subspaces.

The previous work is able to detect efficiently some cases of clusterings, and produce good-quality output. However, the general projective clustering solutions can still be improved in many ways:

- *Non-orthogonal clusters*. While computing only orthogonal projective clusters has practical applications, this constraint is not required in many applications. Furthermore, this constraint makes the problem rather non-robust: even if the optimal subspaces are orthogonal, a slight rotation of the point set would make such algorithms incapable of detecting the required clustering.
- *Non-balanced cluster sizes*. While sampling and other techniques yield fast reliable methods for the case in which clusters have similar sizes, they fail when the sizes of the clusters are different; in many cases simply missing out the small clusters. The disadvantages of this are two-fold: it yields more error in clustering, and it forces other clusters to have a higher dimensional subspace as they contain the missed clusters' points.
- *Cluster dimension as input*. In most applications, the dimensions of the clusters are not known. Hence it is a restrictive constraint to assume that the dimensions of the clusters will be given as input.

Our contributions. In this paper we propose a new objective function for projective clustering which takes into account the inherent trade-off between the dimensionality of the subspaces and the "goodness" of the clustering solution. We then propose an extension of the k-means algorithm for projective clustering that avoids local minima and works well for clusters of widely varying sizes and dimensionalities. We extend the algorithm to automatically detect the cluster dimensionalities. Finally, we present experimental results that illustrate the advantages of our algorithm over the previous algorithms.

2. PROJECTIVE CLUSTERING

In this section we describe the new objective function for projective clustering. Since this objective function is inspired by the k-means algorithm, we first give the definition of k-means clustering (Section 2.1) and then extend it to k-means projective clustering (Section 2.2). Finally, we describe how to compute the dimensions of the clusters automatically (Section 2.3).

Let $P = \{p_1, \ldots, p_n\}$ be a set of n points in $\mathbb{R}^d$. $\mathcal{P} = \{P_1, \ldots, P_k\}$ is called a k-partition of P if $P_1 \cup \ldots \cup P_k = P$ and $P_i \cap P_j = \emptyset$ for every i and j. A q-flat in $\mathbb{R}^d$ is defined as (a translation of) the subspace spanned by a set of q linearly independent vectors. For example, points, lines, and planes are 0-, 1- and 2-flats respectively. Given a q-flat ξ, define the distance of a point p to ξ as $d(p, \xi) = \min_{q \in \xi} d(p, q)$.

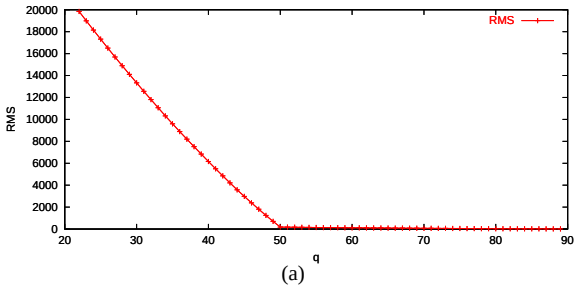
2.1 k-Means Clustering

Given a set P of points in $\mathbb{R}^d$, the *root-mean-square* (RMS) distance of a point $x \in \mathbb{R}^d$ to P is defined as the root-mean of the sum of squared distances of x to the points in P. Note that this distance is directly proportional to the sum of square distances, and from here onwards, we will present minimization for sum of squares, which yields the minimization for root-mean-square distances. Define the sum of square distances as:

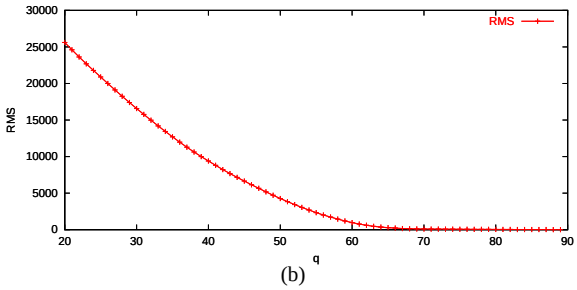
$$\mathrm{rms}(P, x) = \sum_{p \in P} d^2(p, x). \tag{1}$$

The notion of *root-mean-square* distance can be extended to a compact set C as the sum of squared distances of points in P to their nearest point in C:

$$\mathrm{rms}(P, C) = \sum_{p \in P} \min_{c \in C} d^2(p, c). \tag{2}$$



(a)



(b)

Figure 2: The graph of μ^* vs. q for (a) densely packed $2,000$ points in $\mathbb{R}^{100}$, (b) a set of sparsely scattered $2,000$ points.

The point c that minimizes $\mathrm{rms}(P, c)$ (i.e., $c = \arg\min_x \mathrm{rms}(P, x)$) is the *centroid* of P. In general, given an integer k, one can define the set of k points that realize minimum $\mathrm{rms}(P, C)$ over all sets C of size k:

$$\mu_k(P) = \arg \min_{|C|=k} \mathrm{rms}(P, C).$$

Note that $\mu_1(P)$ is simply the centroid of S. For brevity, we will use $\mu(P)$ to refer to $\mu_1(P)$. Also, we will use $\mu_k^*(P)$ to denote the minimum value $\mathrm{rms}(P, \mu_k(P))$.

The set of points $\mathcal{C} = \{c_1, \ldots, c_k\} = \mu_k(P)$ induce a k-partition (also called a *Voronoi Tessellation* [8]) $\{C_1, \ldots, C_k\}$ of P, where C_i is the subset of P with the point c_i as its closest neighbor. It is easy to see that the centroid of each set C_i is, in fact, exactly the point c_i; otherwise $\mathrm{rms}(P, \mathcal{C} \setminus \{c_i\} \cup \mu(C_i))$ is less than $\mu_k^*(P) = \mathrm{rms}(P, \mathcal{C})$, a contradiction.

2.2 k-Means Projective Clustering

It is natural to extend the notion of min-variance clustering to computing subspaces that minimize the RMS distance of the input points to their nearest subspace, allowing the dimension of the subspace to vary. However, we run into a problem: given a set S and an integer q, if we wish to compute a q-flat that minimizes the RMS distance between S and this flat, the RMS distance decreases monotonically with q (Figure 2). Hence if we simply ask for computing subspaces of varying dimensions such that the RMS distance between the input points and the subspaces is minimized, the algorithm will simply return the subspaces $\mathbb{R}^d$.

This problem can be addressed by defining projective clustering to take as input a set of k integers specifying the dimension of each of the required flats, and asking to minimize the cost function $\mathrm{rms}(\cdot)$ over all flats of these dimensions, i.e., generalize (2) to

$$\mathrm{rms}(P, F) = \sum_{p \in P} \min_{f \in F} d^2(p, f),$$

as the distance squared measure of partitioning P using flats in the set F. Given a set P, integer k, and a sequence Q of integers specifying the subspace dimensions, the projective clustering can be defined as:

$$\mu_k(P, Q) = \arg \min_{|F|=k} \mathrm{rms}(P, F),$$

where the minimum is taken over all sequences $F = \langle f_1, \ldots, f_k \rangle$ of k flats so that f_i has dimensionality q_i. As before, we will use $\mu(P, q)$ to refer to $\mu_1(P, q)$, and $\mu_k^*(P, Q)$ to denote the minimum value $\mathrm{rms}(P, \mu_k(P, Q))$.

While this extended definition allows flats to have different dimensions, it exacerbates the automation of clustering by requiring very specific input of flat dimensionalities Q, a requirement which can hardly be expected of most applications. We now present a general definition of optimal projective clustering.

First consider the case $k = 1$. We would like to choose a lower dimensional flat that represents P reasonably well. We have drawn the variation $\mu^*(P, q)$ as a function of q in Figure 2 for two cases – in the first case (Figure 2(a)) the "best" value of q is obvious (choose $q = 50$), while it is not at all obvious in the second case (Figure 2) because the value of $\mu^*(P, q)$ changes smoothly with q. Suppose we have a function $dim(\cdot)$, called the *dimension normalization function*, that returns the best value of q for a given point set. We can then define the optimal dimension-normalized subspace of P to be

$$\nu(P) = \mu(P, \dim(P)). \tag{3}$$

Finally, we define the *projective clustering problem*: Given P, k and the dimension normalization function $\dim(\cdot)$, partition P into k clusters that minimize the RMS distance of points in P to their optimal dimension-normalized subspace. More formally,

$$\nu_k(P) = \arg \min_{\text{k-Partition } \mathcal{P}} \sum_i \nu^*(P_i) = \mu^*(P_i, \dim(P_i)). \tag{4}$$

The computation of all the quantities in this section ultimately depends upon computing the q-flat, for a given q, that minimizes the RMS distance to P. Given P, the *singular value decomposition* [17] of an appropriately defined covariance matrix of P returns an orthonormal $d \times d$ matrix, where the subspace spanned by the first q columns of this matrix defines the q-flat that minimizes RMS distance to P. Note that by computing the decomposition only once, we are able to get the RMS distances (together with the subspaces) of the optimal q-flat over *all* q values. This fact will prove important for our algorithm.

2.3 Dimension Normalization Function

The dimension normalization function $\dim(\cdot)$ must capture a meaningful notion of the trade-off between $\mu^*(P, q)$ and q and return the smallest dimension q so that a q-flat approximates P well.

It can be shown that the graph of $\mu^*(P, q)$ as a function of q is *convex*, i.e., as the number of dimensions increases, the change in

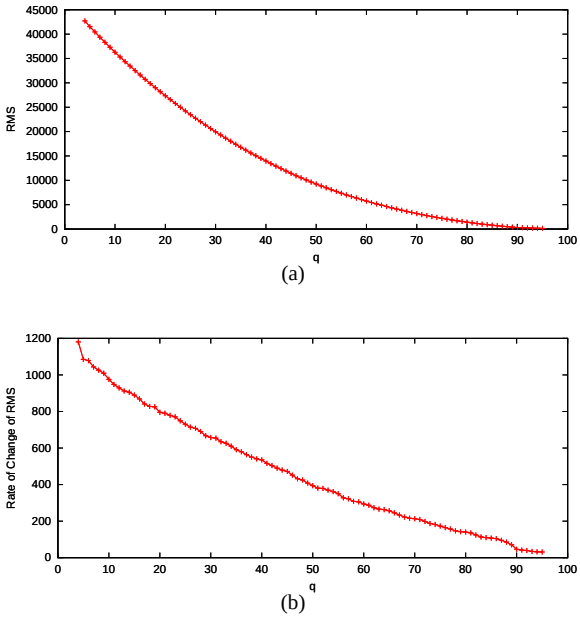


Figure 3: Cluster P consisting of $1,952$ points in $\mathbb{R}^{100}$ (a) The graph of $\mu^*(P,q)$ vs. q for P (b) Graph of the derivative of $\mu^*(P,q)$.

μ^*, from one dimensional subspace to the next, always decreases:

$$\mu^*(P,q-1)+\mu^*(P,q+1) \geq 2\cdot\mu^*(P,q). \quad (5)$$

See Figure 3(b) for an example of the derivative of $\mu^*(P,q)$ as a function of q.

Given the function $\mu^*(P,q)$ and a constant α, let s_α be the smallest dimension q' such that $\mu^*(P,q') \leq \alpha\mu^*(P,1)$. We describe two functions for $dim(\cdot)$; each of them uses s_α as a threshold and returns a value in the interval $[s_\alpha, d]$.

Density-based function. Let l be the line passing through the points $(s_\alpha, \mu^*(P,s_\alpha))$ and $(d,0)$. We compute the point $(q', \mu^*(P,q'))$ farthest from l, for $s_\alpha \leq q' \leq d$, and return q' as the best dimension.

Rate-of-change based function. A more local way of defining the dimension normalization function is by considering at the rate of change of μ^* as the value of q varies:

$$\dim(P) = \min_q \mu(P,q-1)+\mu(P,q+1)-2\mu(P,q). \quad (6)$$

However, this measure is local and non-robust, as local variations can distort the value. To overcome these robustness issues, we propose a scheme that avoids locality by extracting the important dimensions from the entire range $[s_\alpha \ldots d]$ (these dimensions are called *feature* dimensions), and using these extracted feature dimensions to evaluate the difference in the rate of change of $\mu^*(P,q)$.

Let $\pi(P)$ be the polygonal curve defined by the vertex sequence $\langle v_1,\ldots,v_d \mid v_i = (i,\mu(P,i))\rangle$. An ϵ-simplification of $\pi(P)$ (under Fréchet measure) is defined as a subset of vertices of $\pi(P)$ of smallest size whose Fréchet error with respect to $\pi(P)$ is at most ϵ (See [2] for a detailed definition). Intuitively, an ϵ-simplification of $\pi(P)$ is a wrinkle-free version of $\pi(P)$.

Algorithm 1 `Fixed Dimension Projective k-Means`
Input: A Point set $P = \{p_1,\ldots,p_n\} \subseteq \mathbb{R}^d$, $k \in \mathbb{Z}$, $Q = \langle q_1,\ldots,q_k\rangle \subseteq [0\cdot d]^k$
Output: A k-partition $\mathcal{P} = \{P_1,\ldots,P_k\}$ minimizing $\mu_k^*(P,Q)$

```
/* Start with any k-partition of P */
P ← {P_1, ..., P_k}
while Convergence Condition Unsatisfied do
  /* Computing optimal μ(P_i, q_i) for each cluster */
  for j = 1 to k do
    ξ_j = μ(P_j, q_j)
  /* Reassign Points to nearest flat */
  for i = 1 to n do
    Reassign p_i to Cluster arg min_j d(p, ξ_j)
Return P = {P_1, ..., P_k}
```

Given a sensitivity parameter ϵ, let $\pi(P,\epsilon)$ be an ϵ-simplification of P, and let $\pi(P,\epsilon) = \langle v_{i_1},\ldots,v_{i_l}\rangle$. For an integer q, define $\sigma(q)$ (respectively $\tau(q)$) as the smallest (respectively largest) index i_j larger (respectively smaller) than q. Now define the dimension normalization function as:

$$\dim(P) = \max_q \frac{\mu(P,\tau(q))-\mu(P,q)}{\tau(q)-q} \Big/ \frac{\mu(P,q)-\mu(P,\sigma(q))}{q-\sigma(q)}. \quad (7)$$

The user specified parameter ϵ defines the sensitivity of the measure to local changes. Setting $\epsilon = 0$ results in no simplification, and the measure becomes a completely local measure.

We can also use a function that combines density and rate-change based functions. For example, let q_1 and q_2 be the dimensions returned by the density and rate-change based functions respectively. For a fixed parameter β, define

$$\dim(P) = \begin{cases} q_1 & \text{if } |q_2-q_1| \geq \beta q_1, \\ q_2 & \text{Otherwise.} \end{cases} \quad (8)$$

3. ALGORITHM

We will extend the k-means algorithm [8] for computing projective clusterings. We first present the basic algorithm: given a point set P, the number of projective clusters k, and a sequence $Q = \langle q_1,\ldots,q_k\rangle$ of required flat dimensions, it tries to find a k-partition that attempts to minimize $\sum_i \mu^*(P_i,q_i)$ using local improvement steps. The algorithm consists of a number of iterations, improving the current k-partition in each iteration as follows. Start with an initial k-partition of P, chosen by assigning each point uniformly at random to one of the k clusters. Let the current clusters be $P_1,\ldots,P_k$. Compute the q_i-flat minimizing the sum of squared distances to P_i, $\mu(P,q_i)$, as described in Section 2. Given these k new flats $\xi_1,\ldots,\xi_k$, there could exist a point $p \in P_i$ which now has smaller distance to the subspace ξ_j of some other cluster P_j than to ξ_i. Reassign each such point to the nearest subspace, and reiterate. The algorithm is shown in Algorithm 1. Note that in both the steps, the total squared distance value $\sum_i \mu^*(P_i,q_i)$ decreases. Hence the algorithm will converge after a finite number of steps. We can also set a parameter $\mathcal{I}$, and terminate the algorithm after $\mathcal{I}$ iterations if it does not converge in less than $\mathcal{I}$ steps.

The results of running the algorithm on synthetic data (specified in more detail in Section 4) of $50,000$ points in $\mathbb{R}^{100}$ is illustrated in Figure 4. Note that the algorithm is able to converge quite rapidly to a local minimum, converging after only 9 iterations, although in

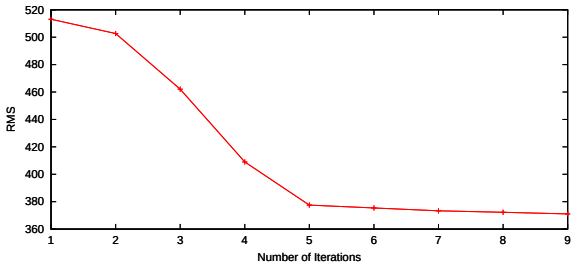


Figure 4: The k-means projective clustering algorithm running on $50,000$ points with five clusters in $\mathbb{R}^{100}$.

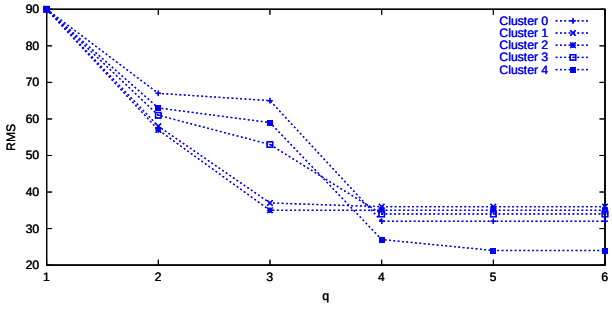


Figure 5: Algorithm 2 on synthetic data of $10,000$ points in $\mathbb{R}^{100}$ where the actual dimensionality of the clusters are $32, 34, 35, 36$ and 24.

some cases it may take many steps to converge.

Detecting subspace dimensionality. The projective k-means algorithm can be extended to minimize the dimension-normalized distance $\nu_k^*(P)$ instead of $\mu_k^*(P, Q)$, thereby eliminating the need for the user to specify the subspace dimensions. In each iteration of Algorithm 1, we now also compute the optimal subspace dimensions: for each current cluster P_i, compute the singular value decomposition of P_i, and using the eigenvalues thus computed, calculate $\dim(P_i)$, which is then set as the flat dimension of point set P_i. The complete procedure is shown in Algorithm 2.

Figure 5 gives an illustration of how the dimensions of various subspaces change with each iteration. Note the iterative nature of dimension computation: as the clusters get more refined, the normalization function $\dim(\cdot)$ returns lower dimensionalities, which in turn forces the reassignments to be accelerated, and so forth. Figure 6 illustrates the change of one curve over many iterations.

Handling local minima. The algorithm just described has two drawbacks: it may converge to a local minimum, and it may miss small clusters. We therefore augment the algorithm by adding further optimization steps that are effective in getting out of a local minimum and in detecting and computing small clusters.

We introduce a cluster splitting step that splits a cluster into two clusters as follows. Intuitively, the k-means algorithm is efficient in assigning an initially scattered set of points to clusters that are locally optimal. Therefore, we split a cluster S into two clusters, aiming to minimize the cost of only one cluster. The hope is that after splitting, the first cluster will contain some well-clustered points,

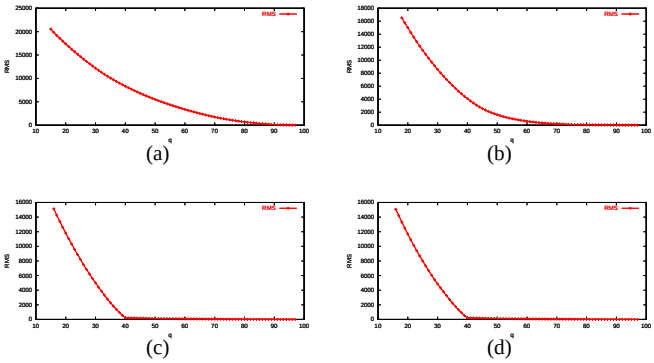


Figure 6: The change of the RMS-q curve for one cluster as the iterative algorithm adjusts the cluster dimensions over many iterations.

Algorithm 2 `Projective k-Means (PkM)`(P)
Input: A Point set $P = \{p_1, \ldots, p_n\} \subseteq \mathbb{R}^d, k \in \mathbb{Z}$
Output: A k-partition $\mathcal{P} = \{P_1, \ldots, P_k\}$

/* Start with any k-partition of P */
$\mathcal{P} \leftarrow \{P_1, \ldots, P_k\}$
$q_i = (d-1), i = 1, \ldots, k.$
while Convergence Condition Unsatisfied **do**
  /* Computing Optimal $\nu(P_i)$ */
  for $j = 1$ to k **do**
    $\xi_j = \mu(P_j, q_j)$
  /* Reassign Points to nearest flat */
  for $i = 1$ to n **do**
    Reassign p_i to Cluster $\arg\ \min_j d(p, \xi_j)$
  /* Recompute cluster dimensions */
  for $j = 1$ to k **do**
    Compute SVD (P_j)
    q_j = dim(P_j)
Return $\mathcal{P} = \{(P_1, q_1), \ldots, (P_k, q_k)\}$

while the second cluster's scattered points will be quickly reassigned during an iteration of the k-means algorithm.

Formally, define the problem *projective clustering with outliers:* Given a set of points S, compute a cluster $S_1 \subseteq S$ with $|S_1| \geq |S|/2$ such that the $\nu^*(S_1)$ is minimized over all subsets with size at least $|S|/2$. No polynomial-time algorithm for approximating this problem is known, therefore we present an iterative scheme to find a good solution.

Given S and a user-specified parameter γ, we iteratively discard $\gamma|S|$-th points that are farthest from the optimal flat $\xi = \nu(P)$, and recompute the optimal flat for the remaining points, and iterate. The procedure is given in Algorithm 4. Note that after $i = \log_\gamma(1/2)$ iterations, S_i contains $n/2$ points. Furthermore, the number of points thrown away decreases at each step as the subspaces gets refined.

For merging two clusters S_1 and S_2, we compute the cost of $S_1 \cup S_2$, $\nu^*(S_1 \cup S_2)$, and that measures the quality of merging the two clusters. The merging procedure is given in Algorithm 5.

The final algorithm, called *K-means with Splitting and Merging* (`KSM`) is shown in Algorithm 3. It consists of repeated calls to

Algorithm 3 `KSM` (P)
Input: A Point set $P = \{p_1, \ldots, p_n\} \subseteq \mathbb{R}^d$, $k \in \mathbb{Z}$, $Q = \langle q_1, \ldots, q_k \rangle \in [0 \cdot d]^k$
Output: A k-partition $\mathcal{P} = \{P_1, \ldots, P_k\}$, minimizing $\mu_k^*(P, Q)$

/* Start with any k-partition of P */
$\mathcal{P} \leftarrow \{P_1, \ldots, P_k\}$
while Convergence Condition Unsatisfied **do**
  `Split Clusters`($\mathcal{P}, k/2, \gamma$)
  `Projective k-Means`($\mathcal{P}$) for $\mathcal{J}$ iterations.
  `Merge Clusters`($\mathcal{P}, k/2$)
Return $\mathcal{P} = \{P_1, \ldots, P_k\}$

Algorithm 4 `Split Clusters`($\mathcal{P} = \{P_1, \ldots, P_l\}$)
Input: A k-partition $\mathcal{P} = \{P_1, \ldots, P_l\}$, $m \in \mathbb{Z}, \gamma \in \mathbb{R}$
Output: A k-partition $\mathcal{P} = \{P_1, \ldots, P_{l+m}\}$

Let $\mathcal{P}' = \{P_{i_1}, \ldots, P_{i_l}\}$ be sorted in *decreasing* order of $q_i \cdot |P_i|$
for $u = 1$ to m **do**
  /* Split cluster P_{i_u} */
  Initialization: $S_0 = P_{i_u}$
  for $j = 0$ to $\log_\gamma(1/2)$ **do**
    $\xi = \mu(S_j, q_{i_u})$
    Compute $d(p, \xi)$, for all $p \in S_j$
    Select point p with $(1-\gamma)|S_j|$-th largest distance
    Set $S_{j+1} = \emptyset$
    for $r \in S_j$ **do**
      if $d(r, \xi) \leq d(p, \xi)$ **then**
        $S_{j+1} = S_{j+1} \cup \{r\}$
  $P_{l+u} = S_j$, where $j = \log_\gamma(1/2)$
Return $\{P_1, \ldots, P_{l+m}\}$

Projective k-means, *Split Cluster* and *Merge Cluster*. Note that the calls to Projective k-means (`PkM`) must specify the number of iterations – typically we set this to a small constant. We also briefly look at the quality of the output on the synthetic data for which we ran k-means projective clustering (Figure 4). The results of using the improved algorithm are provided in Figure 8 — it can be verified that the algorithm actually reaches the global minimum with only five iterations. The improvement resulting from our augmenting split and merge steps combined with k-means is quite significant.

Running time. A single iteration of `PkM` performs k projection and SVD computations. Given a user-specified number of iterations $\mathcal{I}$ and splitting factor γ, Algorithm `Split Clusters` performs $\log_\gamma(1/2)$ projection and SVD computations. Algorithm `Merge Clusters` performs $O(k^2)$ SVD computations.

The bottleneck in the running time of the above algorithm is in computing the singular value decomposition of a point set. With very little loss of fidelity, we can instead approximate the optimal flat that minimizes the error to the point set P by randomly sampling a set S of points from P, and computing the optimal flat for S instead. The hope is that the optimal flat for a random sample S would be sufficiently close to the optimal flat for the set P. We repeat the above sampling and SVD computation procedure a number of times, and take the computed flat that best minimizes the RMS error to the point set.

Algorithm 5 `Merge Clusters`($\mathcal{P} = \{P_1, \ldots, P_l\}$)
Input: A k-partition $\mathcal{P} = \{P_1, \ldots, P_l\}$, Integer m
Output: A k-partition $\mathcal{P} = \{P_1, \ldots, P_{l-m}\}$

for $u = 0$ to m **do**
  $(i, j) = \arg\min_{i,j} \mu^*(P_i \cup P_j, \min\{q_i, q_j\})$
  /* Merging clusters P_i and P_j */
  $P_i = P_i \cup P_j$, Remove P_j
Return $\{P_1, \ldots, P_{l-m}\}$

Clustering along non-linear surfaces. Once we have an algorithm for projective clustering that clusters points along lower dimensional subspaces (of different dimensions), it becomes possible to cluster points along a variety of non-linear shapes. Using the so-called *linearization* technique, we can map a number of shapes (circles, cylinders, etc.) to a flat in a higher dimensional space. For example, the map $\varphi : \mathbb{R}^d \rightarrow \mathbb{R}^{d+1}$ defined as

$$\varphi(x_1, \ldots, x_d) = (x_1, \ldots, x_d, x_1^2 + \ldots, x_d^2),$$

maps a sphere σ in $\mathbb{R}^d$ to a hyperplane h_σ in $\mathbb{R}^{d+1}$ in the sense that a point $p \in \mathbb{R}^d$ lies on (resp. inside, outside) σ if and only if $\varphi(p)$ lies on (resp. below, above) h_σ. Therefore if we wish to cluster a set P of points along spheres, we use the projective clustering algorithm on $\varphi(P)$ and map points in each cluster back to the original point set. We refer the reader to [1, 11] for details on linearization.

4. EXPERIMENTS

We compare our two methods (`PkM` and `KSM`) with the algorithms `ORCLUS` [3] and `DOC` [19]. `ORCLUS` performs hierarchical clustering and `DOC` computes orthogonal projective clusters using an iterative approach. Both of them choose a random sample of the input for efficiency reasons. We study the performance of these four algorithms under two conditions: (i) whether clusters have the same dimension or different dimensions, and (ii) the clusters are balanced or unbalanced.

All experiments are performed on a Pentium IV 2.4GHz machine with 4 GB of main memory, and running Linux operating system. Unless stated otherwise, all synthetic data sets have $50,000$ points in $\mathbb{R}^{100}$.

Synthetic data generation. We use the data generator described in [19], but modify it to generate non-orthogonal clusters as follows [19] (a similar generator was used in [3]).

We fix a parameter k and choose the point set to be the union of k clusters $IC_1, \ldots, IC_k$. Each cluster IC_i is defined by a flat f_i of dimension q_i, and the points in IC_i are chosen in the neighborhood of f_i. We first choose f_i to be an orthogonal flat, i.e., it spans a subset of the coordinate axes. Then we rotate f_i and IC_i to an arbitrary orientation. The number of coordinate axes which f_i does not span is called the *bounded dimensions* of IC_i, and is denoted by d_i; clearly $d_i = d - q_i$. We now describe these steps in more detail.

We generate two types of data sets: *fixed dimensionality* input, in which all f_i's have the same dimension, i.e. $q_i = q$, for all $1 \leq i \leq k$; and *variable dimensionality* input, in which each cluster has a different dimension. In the latter case d_i (and thus q_i) is chosen as follows. We fix a parameter μ and choose d_i from a Poisson distribution with mean μ. Next, we choose a subset D_i of d_i coordinate axes, along which f_i does not span, as follows. We randomly choose $\min\{d_{i-1}, d_i/2\}$ axes from D_{i-1} and the remaining

ones are chosen randomly. This iterative technique allows different clusters to share subspaces.

After having chosen f_i, we choose the number of points in each IC_i. We generate k exponential random variables $x_0, \ldots, x_{k-1}$ with mean 1, and set the size of IC_i to $n \cdot (x_i / \sum x_j)$. To generate unbalanced cluster sizes, 80% of the points of the initial half of clusters are assigned to the corresponding second half of clusters, i.e. $x_i = 0.2x_i$ and $x_{i+\lfloor k/2 \rfloor} = x_{i+\lfloor k/2 \rfloor} + 0.8x_i$, for $0 \leq i < \lfloor k/2 \rfloor$.

Next we choose points in IC_i. Each point has coordinates in the range $[0 \ldots 100]$. The coordinates in the unbounded dimensions are chosen uniformly at random, while coordinates in the bounded dimension follow one of the following distributions:

1. *Uniform Distribution.* Coordinates chosen uniformly at random with width w. In our experiments, it is set to 15.
2. *Normal distribution.* For a cluster IC_i, determine the variance σ_{ij} of dimension j by randomly choosing the scale factor s_{ij} from $[1, 2]$, and setting $\sigma_{ij} = (2s_{ij})^2$. Then the jth coordinates of points are generated with mean 1 and variance σ_{ij}.
3. *MGq-clusters.* Coordinates follow a mixture of q Gaussian distributions.

Finally we generate random rotation matrices independently for each cluster as follows: generate a $d \times d$ matrix with entries following a normal distribution, and compute its QR decomposition. It is known that this method provides a random rotation matrix Q with the correct distribution [12].

Choice of parameters. We set the parameters as follows: $\alpha = 0.2$, $\gamma = 0.95$, number of iterations of PkM is set to 15, and algorithm KSM calls PkM with number of iterations $\mathcal{J} = 2$. We use the hybrid dimension-normalization function described at the end of Section 2.3 and set the parameter $\beta = 0.3$.

Measuring output quality. The quality of the output describes how well an algorithm is able to detect and compute all the clusters. We measure quality by the following two quantities. *Mismatch ratio* as used in Procopiuc *et al.* [19]: for an output cluster OC_i, let $\chi(i)$ denote the index of the input cluster IC_j whose points are the largest proportion in OC_i. The *mismatch ratio* is defined as:

$$\frac{1}{n} \sum_i \sum_{j \neq \chi(i)} |OC_i \cap IC_j|.$$

However, for unbalanced cluster sizes, this is an unreliable measure since missing small clusters only affects the mismatch ratio slightly, although some entire clusters could have been missed. Therefore we also use the *normalized mismatch ratio*: define $\chi(i)$ for each output cluster OC_i as before. Then the normalized mismatch ratio is:

$$\frac{1}{k} \sum_j \sum_{\chi(i) \neq j} |IC_j \cap OC_i| / |IC_j|.$$

Also, it is instructive to view the exact distributions of the points in output clusters with respect to all the input clusters. Therefore, we will also compute the *confusion matrix*: the entry (i, j) of the matrix contains $|OC_i \cap IC_j|$.

	Fixed Dimensions			Variable Dimensions		
q	DOC	PkM	KSM	DOC	PkM	KSM
15	0.00	0.20	0.00	0.00	0.20	0.00
20	0.00	0.00	0.00	0.00	0.20	0.00
25	0.00	0.00	0.00	0.00	0.00	0.00
30	0.00	0.05	0.00	0.00	0.20	0.00
35	0.00	0.00	0.00	0.00	0.26	0.00

Figure 7: Comparing DOC and KSM for orthogonal clusters. Each entry shows the normalized mismatch ratio.

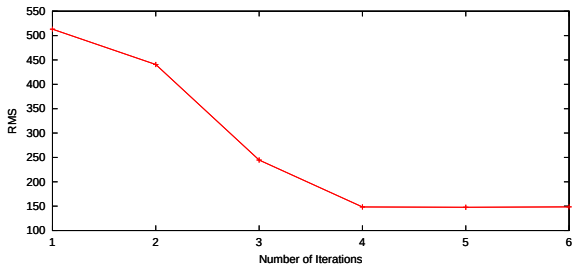


Figure 8: The improved k-means projective clustering algorithm running on $50,000$ points with five clusters in $\mathbb{R}^{100}$.

Orthogonal clusters. Since DOC only computes orthogonal clusters, we first compare our algorithm with only DOC for the special case of orthogonal clusters. We generate the point sets (as described above) with Normal distribution and unbalanced cluster sizes. We then observe the performance of the three algorithms as q varies by computing normalized mismatch ratio. The results are presented in Figure 7. Both DOC and KSM perform well.

For now on, we will run all the experiments on non-orthogonal clusters, generated as described earlier. We include DOC in our comparisons — its high mismatch ratios illustrates its severe limitation to orthogonal clusters only.

Fixed dimensionalities. We first compare the four methods when the clusters are non-orthogonal and the dimensionalities of all the flats are fixed and given as input to the methods. We vary this dimensionality q between 15 and 50.

Balanced point sets. We generate the point set P of size $50,000$ with 5 clusters using normal distribution. Figure 9 shows the qual-

q	DOC	ORCLUS	PkM	KSM
15	0	0.00	0.00	0.00
20	0.00	0.00	0.00	0.00
25	0.15	0.00	0.00	0.00
30	0.07	0.00	0.00	0.00
35	0.17	0.00	0.00	0.00
40	0.14	0.00	0.00	0.00
45	0.17	0.00	0.00	0.00
50	0.23	0.00	0.00	0.00

Figure 9: Mismatch ratio for the four algorithms when all clusters have the same dimension and are balanced.

q	DOC	ORCLUS	PkM	KSM
15	0.00	0.00	0.40	0.00
20	0.00	0.20	0.20	0.00
25	0.20	0.20	0.20	0.00
30	0.54	0.00	0.20	0.00
35	0.47	0.40	0.20	0.00
40	0.49	0.20	0.00	0.00
45	0.69	0.20	0.00	0.00
50	0.72	0.02	0.00	0.00

Figure 10: Normalized mismatch ratio when all clusters have the same dimensionality but are unbalanced.

ity of the clusterings by the various algorithms: q is the dimension of the subspace of the input clusters (same for all clusters) and each entry denotes mismatch ratio. In the experiments we performed, KSM returned the optimal clustering, with mismatch ratio of 0. Similarly, ORCLUS performs quite well, computing clusters with zero mismatch. DOC performs poorly, which is expected since the clusters are non-orthogonal.

Unbalanced point sets. Next, we generate unbalanced point sets, where the sizes of the input clusters can vary widely, i.e., the ratio of the largest cluster to the smallest cluster is high. This introduces an extra challenge for most algorithms, as it becomes more difficult to distinguish the small clusters, which are often lumped together with the larger clusters. The table comparing the qualities of output is shown in Figure 10. Each entry denotes the normalized mismatch ratio. The dimension of all the subspaces $f_1, \ldots, f_k$ is q.

KSM has a zero normalized mismatch ratio on all the inputs tried, as shown in Figure 10. This is one of the important strengths of the KSM: by continually splitting and merging clusters, interleaved with local optimizations performed by PkM, it nicely detects small clusters. ORCLUS *completely* misses the two small clusters, and they are lumped with other larger clusters in the output. Two out of the five clusters are completely missed, and the normalized mismatch ratios are quite large: mostly between $20\% - 40\%$ as seen in Figure 10.

To further illustrate this point, we can also look at particular confusion matrices for comparing these algorithms.

Figure 11 shows the confusion matrices for the data set used in Figure 10 with $q = 35$. Algorithm KSM, as witnessed from the mismatch ratio, is able to match output clusters to input clusters completely. The first two input clusters, IC_0 and IC_1, have small size, and in the output of DOC, and ORCLUS, they have been merged with the other larger clusters (DOC lumps IC_0 with IC_2, and IC_1 with IC_3 in the output clusters; ORCLUS lumps both IC_0 and IC_1 with IC_3 in the output clusters).

Variable dimensionalities. We now generate data sets in which the dimensions of the various clusters are generated with a Poisson distribution with mean q. We provide this mean value as input to the programs ORCLUS and KSM.

Balanced point sets. Figure 12 shows the mismatch ratios for balanced point sets. There are some mismatches for KSM on point sets with small subspace dimensionality: since the cluster dimensions were generated around a mean, there are cluster dimensions which are extremely low, i.e., 1 or 2. Detecting these very low dimensions is hard for the algorithm, and that consequently distorts the mismatch ratio. We also compute the confusion matrix and the corresponding dimensions in such a case. The confusion matrix for KSM on the data set used in Figure 12, with $q = 15$, is shown in Figure 14. Output cluster OC_0 is matched correctly, *with* the exactly correct dimension value (28) with cluster IC_2. Similarly, OC_2 is matched correctly with IC_1. However, the algorithm is unable to detect the dimension of cluster IC_0, and lumps it with cluster IC_4. Since input cluster IC_0 has very low dimension (2), output cluster OC_4, which contains exactly the points of IC_0 and IC_4, gets assigned the dimension 36. Finally, input cluster 3, of size $9,032$ is divided roughly equally between output clusters 1 and 3, and both of these clusters get (correctly) the dimension value of IC_3, i.e. 21.

Another surprising fact is that ORCLUS is able to cluster points well, despite the fact that the clusters lie in different dimensions. This is due to the fact that ORCLUS starts the hierarchical clustering with full dimensions, and slowly decreases the dimensions. Therefore initially points get assigned to their proper clusters, and although the inter-cluster distance (as defined by ORCLUS) becomes large later on (due to the fact that the dimensionality is fixed), the sparsity assures the points are not reassigned to some other cluster.

Algorithm DOC						
OC #	IC_0	IC_1	IC_2	IC_3	IC_4	Total
0	2014	0	4678	0	0	6692
1	0	1718	0	7804	0	9522
2	0	0	15603	914	0	16517
3	0	0	0	19097	3833	22930
4	0	0	0	0	11506	11506
Total	2014	1726	15612	19137	11511	
Algorithm ORCLUS						
OC #	IC_0	IC_1	IC_2	IC_3	IC_4	Total
0	471	0	0	0	11511	11982
1	18	0	0	7640	0	7658
2	1525	0	0	6650	0	8175
3	0	1726	0	4847	0	6573
4	0	0	15612	0	0	15612
Total	2014	1726	15612	19137	11511	
Algorithm KSM						
OC #	IC_0	IC_1	IC_2	IC_3	IC_4	Total
0	0	0	0	19137	0	19137
1	0	0	15612	0	0	15612
2	0	0	0	0	11511	11511
3	2014	0	0	0	0	2014
4	0	1726	0	0	0	1726
Total	2014	1726	15612	19137	11511	

Figure 11: Confusion matrix for an unbalanced cluster data set with cluster dimension $q = 35$. Each entry (i, j) contains the number of points common to both the output cluster OC_i and input cluster IC_j.

q	DOC	ORCLUS	PkM	KSM
15	0.08	0.18	0.00	0.18
20	0.30	0.16	0.00	0.37
25	0.10	0.00	0.18	0.21
30	0.20	0.00	0.00	0.00
35	0.27	0.00	0.00	0.00
40	0.15	0.00	0.00	0.00
45	0.18	0.00	0.00	0.00
50	0.50	0.00	0.19	0.00

Figure 12: A point set with $50,000$ points, and the individual cluster dimensions follow a Poisson distribution with mean q. The clusters are well-balanced.

q	DOC	ORCLUS	PkM	KSM
15	0.25	0.40	0.20	0.50
20	0.42	0.20	0.20	0.20
25	0.56	0.20	0.00	0.20
30	0.48	0.00	0.00	0.00
35	0.59	0.40	0.00	0.00
40	0.50	0.40	0.00	0.00
45	0.23	0.00	0.00	0.00
50	0.71	0.20	0.20	0.00

Figure 13: A point set with $50,000$ points. Each entry denotes the normalized mismatch ratio. The individual cluster dimensions follow a Poisson distribution with mean q. The sizes of the clusters can vary widely.

Algorithm KSM							
OC #	Dim	IC_0	IC_1	IC_2	IC_3	IC_4	Total
		2	28	28	21	33	
0	28	0	0	8710	0	0	8710
1	21	0	0	0	4821	0	4821
2	28	0	11613	0	0	0	11613
3	21	0	0	0	4211	0	4211
4	36	11290	0	0	0	9355	20645
Total		11290	11613	8710	9032	9355	

Figure 14: Confusion matrix for KSM on the data set in Figure 12 with $q = 15$.

Unbalanced point sets. The clusterings for unbalanced point sets are shown in Figure 13. A specific confusion matrix is shown in Figure 15, showing the output clusters computed, this time together with their dimensionalities. The input point set dimensionalities were generated with the mean value 35 (the actual dimensions are given in the table). Note that KSM not only clusters the points without mismatches, the dimension normalization function $\dim(\cdot)$ gradually sets the dimension values of each clusters *exactly* correctly. The same phenomenon was observed with the other data sets presented in Figure 13.

Non-linear shape clustering. We also tested our algorithm for clustering points along spheres using the linearization technique described in Section 3. We generate points distributed randomly (with random noise) on spheres using the software *rbox* (part of *qhull*). We then randomly scale and rotate the spheres. This gives the input set of spheres (with possibly different dimensionalities). We then compute the transformation and the projective clustering. The experimental results are shown in Figures 16.

Sensitivity analysis. We now briefly report some experimental results on varying a few other parameters.

Point distributions. We tested our algorithm on point sets generated from the two distributions mentioned earlier: Multiple Gaussians, where the coordinates follow a mixture of 4 distributions, and uniform, with points uniformly generated with width 15. Figure 17 presents the mismatch ratios when the dimension of each cluster is the same and varies from $20 \ldots 50$. The accuracy results are not much different from the ones for point sets generated with Normal distributions examined earlier.

Running time. The running time of KSM is linear in n, and cubic in d. Typically running $100,000$ points with $d = 50, m = 5$ takes around 10–15 seconds per iteration of algorithm KSM. So the overall running time for 15 iterations is roughly 3–5 minutes.

Algorithm DOC							
OC #	Dim	IC_0	IC_1	IC_2	IC_3	IC_4	Total
		28	35	30	46	42	
0	42	1883	0	1842	3447	0	717
1	45	0	2337	0	2938	109	5384
2	30	0	0	14021	1123	0	15144
3	32	0	0	0	19021	1608	20629
4	21	0	0	0	127	11631	11758
Total:		1883	2337	14027	19741	12013	

Algorithm ORCLUS							
OC #	Dim	IC_0	IC_1	IC_2	IC_3	IC_4	Total
		28	35	30	46	42	
0	35	70	2337	14027	0	0	16434
1	35	0	0	0	7124	0	7124
2	35	1515	0	0	5429	0	6944
3	35	298	0	0	0	12013	12311
4	35	0	0	0	7188	0	7188
Total		1883	2337	14027	19741	12013	

Algorithm KSM							
OC #	Dim	IC_0	IC_1	IC_2	IC_3	IC_4	Total
		28	35	30	46	42	
0	42	0	0	0	0	12013	12013
1	28	1883	0	0	0	0	1883
2	30	0	0	14027	0	0	14027
3	46	0	0	0	19741	0	19741
4	35	0	2337	0	0	0	2337
Total		1883	2337	14027	19741	12013	

Figure 15: Confusion matrix for unbalanced-cluster data set with cluster dimensions mean value $q = 35$. The input and output cluster dimensionalities computed by the algorithms are given as well.

	Fixed Dimensions			*Variable Dimensions*		
q	ORCLUS	PkM	KSM	ORCLUS	PkM	KSM
20	0.00	0.00	0.40	0.17	0.00	0.00
30	0.00	0.20	0.00	0.20	0.00	0.00
40	0.00	0.20	0.00	0.00	0.00	0.00
50	0.00	0.20	0.00	0.00	0.00	0.00
60	0.00	0.20	0.00	0.19	0.00	0.00

Figure 16: Clustering points distributed randomly along spheres. Each point set has $50,000$ points, with 5 clusters. Each entry denotes the normalized mismatch ratio.

Sampling accuracy. As mentioned in Section 3, we speed up the algorithm by using random sampling and computing optimal flats for those samples. All the results so far have been achieved by setting the sample size to $3 \cdot d$, where $d = 100$. As is clear, this is sufficient to achieve the high level of accuracy exhibited. The accuracy of KSM is largely unaffected by sampling changes even for large data sets with many clusters.

Clustering real data. In addition to our experiments on synthetic data, we tested our algorithm on two real data sets – gene expression data and computer vision data.

Gene data. Given a set of genes (viewed as points), and their expression values in a number of tissues (corresponding to dimensions), one would like to cluster the genes based on these expression values through different tissues. Two genes are "close" if their pattern of expression values are similar, even if the individual values are different. We cluster the Arabidopsis gene expression data, with $7,877$ points in $\mathbb{R}^{12}$. Figure 18 shows the genes (each curve) in a cluster together with their expression levels across varying tissues. Figure 19 shows the actual RMS clustering values of the three

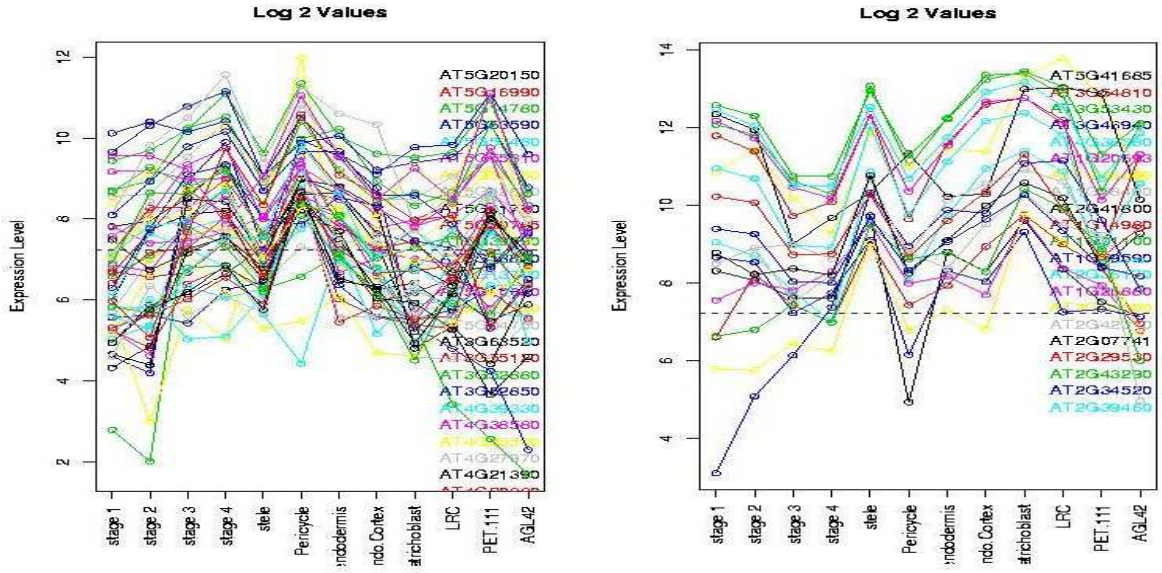


Figure 18: Set of genes from the same cluster. Each curve represents a gene, with the y-coordinate as the expression value for that particular tissue.

	Uniform				*Multiple Gaussian*			
q	DOC	ORCLUS	PkM	KSM	DOC	ORCLUS	PkM	KSM
20	0.00	0.04	0.04	0.00	0.00	0.00	0.09	0.00
30	0.00	0.00	0.03	0.00	0.03	0.00	0.00	0.00
40	0.08	0.03	0.00	0.00	0.21	0.00	0.00	0.00
50	0.12	0.09	0.04	0.00	0.18	0.04	0.04	0.00

Figure 17: Mismatch ratios on point sets with uniform and multiple Gaussian distributions and with unbalanced cluster sizes. For uniform distribution the coordinates are bounded with width 15, and we use a mixture of 4 Gaussian distributions.

k	ORCLUS	PkM	KSM
10	49.88	52.36	52.05
20	47.83	50.52	49.73
40	44.90	46.49	46.17
70	40.57	43.61	42.94
100	40.63	39.36	39.56
200	35.24	29.22	29.53

Figure 19: Projective clustering algorithms on gene expression data. The fht dimension is 5.

algorithms.

Vision data. One image-processing problem in computer vision is the recognition of letters from the corresponding hand signs. A hand gesture (stored as an image) is classified into 24 classes (each, except 2 of them, representing an alphabet), with each class further having a number of slight variational gestures. By adaptively sampling a number (300 in our case) of the same pixels from each image, the "closeness" of the two images can be computed by matching the intensity values (all images are gray-scale). We ran the algorithms on this $10,000$ point set data in $\mathbb{R}^{300}$. Figure 20 gives the RMS values for clustering this data set under the three algorithms.

k	q	ORCLUS	PkM	KSM
5	10	31.11	29.63	29.46
10	10	29.18	25.75	25.76
15	10	27.59	22.87	22.85
20	1	28.71	27.43	27.31
20	2	28.26	26.17	26.08

Figure 20: Projective clustering algorithms on vision data of $10,000$ points in $\mathbb{R}^{300}$.

5. CONCLUSION

In this paper we have proposed a new definition of projective clustering, together with local improvement algorithms and their implementations. We have tested and compared our algorithms with previous work, and illustrated their feasibility and practicality.

A number of interesting questions remain open. One shortcoming of our approach is that for clusters of widely varying dimensions, clusters of small dimensions are merged into various higher dimensional clusters. This seems like an inevitable drawback of any local improvement algorithm, and it would be interesting to design algorithms that work for such cases. Furthermore, our algorithms have cubic dependence on the dimension d, and therefore are infeasible for very high dimensional clusterings. So one direction would be the design of algorithms that exhibit near-linear running time d. Finally, it would be nice to have a worst-case theoretical analysis of these algorithms of their running time (i.e., number of iterations), and the quality (as measured by our objective function).

Acknowledgments

We would like to thank Magda Procopiuc for providing the DOC data generator, Arvind Sastry for providing the vision data, and David Orlando for the gene expression data.

References

[1] P. Agarwal and S. Har-Peled, Maintaining the approximate extent measures of moving points, *Proc. 12th ACM-SIAM Sympos. Discrete Algorithms*, 2001, pp. 148–157.

[2] P. K. Agarwal, S. Har-Peled, N. Mustafa, and Y. Wang, Near-linear time approximation algorithms for curve simplification in two and three dimensions, *Proc. of the 10th European Symposium on Algorithms*, 2002, pp. 544–555.

[3] C. Aggarwal and P. Yu, Finding generalized projected clusters in high dimensional spaces, *Proc. ACM-SIGMOD Intl. Conf. Management of Data*, 2000, pp. 70–81.

[4] C. C. Aggarwal, C. M. Procopiuc, J. L. Wolf, P. S. Yu, and J. S. Park, Fast algorithms for projected clustering, *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, 1999, pp. 61–72.

[5] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan, Automatic subspace clustering of high dimensional data for data mining applications, *Proc. ACM-SIGMOD Intl. Conf. Management of Data*, 1998, pp. 94–105.

[6] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, When is "nearest neighbor" meaningful?, *Lecture Notes in Computer Science*, 1540 (1999), 217–235.

[7] K. Chakrabarti and S. Mehrotra, Local dimensionality reduction: A new approach to indexing high dimensional spaces, *Proc. of 26th Intl. Conf. on Very Large Data Bases*, 2000, pp. 89–100.

[8] Q. Du, V. Faber, and M. Gunzburger, Centroidal voronoi tessellations: Applications and algorithms, *SIAM Review*, 41 (1999), 637–676.

[9] S. Guha, R. Rastogi, and K. Shim, CURE: an efficient clustering algorithm for large databases, *Proc. ACM-SIGMOD Intl. Conf. Management of Data*, 1998, pp. 73–84.

[10] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*, Morgan Kaufmann, 2001.

[11] S. Har-Peled and K. Varadarajan, Approximate shape fitting via linearization, *Proc. 42nd Annu. IEEE Sympos. Found. Comput. Sci.*, 2001, pp. 66–73.

[12] R. M. Heiberger, Algorithm AS 127: Generation of random orthogonal matrices, *Appl. Statist.*, 27 (1978), 199–206.

[13] A. Hinneburg, C. C. Aggarwal, and D. A. Keim, What is the nearest neighbor in high dimensional spaces?, *Proc. 26th Intl. Conf. Very Large DataBases*, 2000, pp. 506–515.

[14] A. Hinneburg and D. A. Keim, Optimal grid-clustering:towards breaking the curse of dimensionality in high-dimensional clustering, *Proc. of 25th Intl. Conf. Very Large DataBases*, 1999, pp. 506–517.

[15] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*, Prentice Hall, Englewood Cliffs, NJ, 1988.

[16] W. Johnson and J. Lindenstrauss, Extensions of Lipschitz maps into a Hilbert space, *Contemp. Math.*, 26 (1984), 189–206.

[17] T. T. Jolliffe, *Principal component analysis*, Springer-Verlag, New York, 2002.

[18] R. T. Ng and J. Han, Efficient and effective clustering methods for spatial data mining, *Intl. Conf. Very Large DataBases*, 1994, pp. 144–155.

[19] M. Procopiuc, M. Jones, P. K. Agarwal, and T. M. Murali, A monte carlo algorithm for fast projective clustering, *Proc. ACM-SIGMOD Intl. Conf. Management of Data*, 2002, pp. 418–427.

[20] T. Zhang, R. Ramakrishnan, and M. Livny, BIRCH: an efficient data clustering method for very large databases, *Proc. ACM-SIGMOD Intl. Conf. Management of Data*, 1996, pp. 103–114.