A Fast Algorithm for Finding Correlation Clusters in Noise Data

Jiuyong Li[1], Xiaodi Huang[2], Clinton Selke[3], and Jianming Yong[4]

[1] School of Computer and Information Science, University of South Australia, Mawson Lakes Adelaide, Australia, 5095 jiuyong.li@unisa.edu.au,
[2] Department of Mathematics, Statistics and Computer Science, The University of New England, Armidale, Australia, 2350,xhuang@turing.une.edu.au,
[3] Department of Mathematics and Computing, The University of Southern Queensland, Australia, clinux_rulz@hotmail.com,
[4] Department of Information Systems, University of Southern Queensland, Australia, Jianming.Yong@usq.edu.au

Abstract. Noise significantly affects cluster quality. Conventional clustering methods hardly detect clusters in a data set containing a large amount of noise. Projected clustering sheds light on identifying correlation clusters in such a data set. In order to exclude noise points which are usually scattered in a subspace, data points are projected to form dense areas in the subspace that are regarded as correlation clusters. However, we found that the existing methods for the projected clustering did not work very well with noise data, since they employ randomly generated seeds (micro clusters) to trade-off the clustering quality. In this paper, we propose a divisive method for the projected clustering that does not rely on random seeds. The proposed algorithm is capable of producing higher quality correlation clusters from noise data in a more efficient way than an agglomeration projected algorithm. We experimentally show that our algorithm captures correlation clusters in noise data better than a well-known projected clustering method.

Keywords: generalised projected clustering, SVD decomposition.

1 Introduction

Clustering is a classical technique in computing and statistics. Noise deteriorates cluster quality significantly and prevents finding meaningful clusters when the amount of noise is big. It is difficult to distinguish noise data objects from normal ones when we do not have prior knowledge about the data. However, clustering can serve as the first step to explore such a data set with noise, particularly when the prior knowledge about the data is unavailable.

Generalised projected clustering sheds light on solving this problem by finding correlated clusters. When data objects are projected to a data subspace using Singular Value Decomposition (SVD) or PCA, the correlation clusters are condensed to a small area

[1] This work was done when J Li was with Department of Mathematics and Computing of University of Southern Queensland. This work was partially supported by Australian Research Council Discovery Grant DP0559090.

whereas noise data objects scatter across the projected space. Therefore, it is possible to separate correlated data objects from noise ones.

Most existing projected clustering methods use agglomeration methods to find correlation clusters. A big data set is randomly partitioned into a large number of micro clusters, and then an agglomeration approach is used to group correlation clusters. When correlated data are split into a number of micro clusters, they themselves become noise too. This process of randomly generated seeds affects the quality of found clusters. When a process for noise elimination is employed, many data objects in the correlation clusters are removed before they are grouped into clusters. Some well known examples of generalized projected clustering are PROCLUS [2], ORCLUS [1], $4C$ [4], CURLER algorithm [5] and HARP [7]. We do not consider axis-parallel projection methods, also called subspace clustering, such as CLIQUE [3], and EPCH [6].

Instead of agglomerating randomly generated micro clusters into final clusters, we partition a data set into clusters in a top-down manner. The key idea for such a divisive method is to find a suitable criterion for data partition. We capture the direction of the largest variance of data using the corresponding principle vector, thereby taking small risk of partition correlation clusters into separate clusters. We employ grouping technique used in agglomeration methods to group correlation clusters after partitions. The proposed divisive projected clustering method preserves the essence of projected clustering, overcoming the drawbacks of existing projected clustering methods. In addition, the proposed algorithm is significantly more efficient than most agglomeration algorithms.

2 Problem definitions

Projected clustering searches for hidden subspaces together with a set of data objects such that data objects are closed with each other in the lower dimensional subspaces. The hidden data spaces are found by using SVD decomposition. Eigenvectors corresponding to eigenvalues with low spreads forms a subspace. The intuitive explanation for this is as follows (see more justifications in Section 4). When the covariance matrix of a set of correlated points is decomposed by SVD, some eigenvalues should be zero or close to zero. All the points are projected along a line in the subspace spanned by eigenvectors corresponding to these zero eigenvalues. In other words, the tightness of objects in the subspace defined by eigenvectors associated with the lowest eigenvalues is an alternative to measuring the correlation level of data objects.

Formally, let D be a dataset of m data objects (row vectors) being treated as d-dimensional feature (column) vectors. $\mathbf{o}_i \in D$ stands for the i-th object in D where $\mathbf{o}_i = (o_{i1}, o_{i2}, \ldots, o_{id})$. Simply, we have $D = [o_{ij}]$, $1 \leq i \leq m$, and $1 \leq j \leq d$.

Definition 1. *Generalised projected clustering*
*Given the user-specified l and k, a data set D is partitioned into k disjoint subsets D_1, D_2, ..., D_k horizontally, such that, for all $1 \leq p \leq k$, S_p contains l close to zero eigenvalues, where $U_p S_p V_p^T = \mathrm{cov}(D_p)$ (*cov *is the covariance matrix of D_p, the SVD decomposition of which results in U_p, S_p, and V_p^T).*

Data points are clustered based on their closeness in some projected subspaces instead of the original space. This clustering captures correlations among data points.

To measure the closeness of data points in a subspace, the projected distance is defined as following.

Definition 2. *Projected distance*
Let D_p be a subset of a data D. $U_p S_p V_p^T = \mathrm{cov}(D_p)$ ($\mathrm{cov}(D_p)$ is the $d \times d$ covariance matrix of D_p, and U_p, S_p, and V_p^T are results of SVD decomposition). Let E be the set of eigenvectors corresponding to l smallest eigenvalues. A data object $\mathbf{p} \in D$ is projected to $E = (\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_l)$ space as $(\mathbf{p} \cdot \mathbf{e}_1, \mathbf{p} \cdot \mathbf{e}_2, \ldots, \mathbf{p} \cdot \mathbf{e}_l)$. The projected distance of objects $\mathbf{p}$ and $\mathbf{q}$, denoted by $\mathrm{Pdist}(\mathbf{p}, \mathbf{q}, E)$,is their Euclidean distance in projected space E.

The projected distance between two data points is the Euclidean distance between their projected images in a subspace. This distance varies in different subspaces.

To measure the projected distance variation over a group of data points in a subspace, the projected energy is defined as the following.

Definition 3. *Projected energy*
The projected energy of data set D_p is defined as $\mathrm{Energy}(D_p, E) = \sum_{i=1}^{i=N} \mathrm{Pdist}(\mathbf{o_i}, \mathbf{c}, E)/N$, where N is the number of objects in D_p, $\mathbf{c}$ the centroid of all objects in D_p, and E an associated subspace.

The smaller the projected energy, the denser the data point in the subspace. In clustering, low projected energy is preferred.

In projected clustering, the traditional distances between data objects are replaced by the projected distances in subspaces. However, there are no uniform and invariant distances in projected clustering since each tentative cluster has its own subspace. In other words, the distance between two objects varies in different subspaces.

Projected distances have been studied in statistics. The Mahalanobis distance [8] measures distance between two objects by using a set of reference data. But the Mahalanobis distance is the projected distance in the entire space. The generalised projected distance is defined in a subspace, and is a generalised Mahalanobis distance.

As mentioned before, the ORCLUS algorithm [1] presents a variant agglomerative method to find k projected clusters. First, D are randomly partitioned into k_0 initial data subsets D_1, D_2, ..., D_{k_0}, where $k_0 >> k$. If each data object is considered as an initial micro cluster, then the computational cost will be too expensive. The smaller k_0, the faster the ORCLUS. However, the high quality of clusters is sacrificed if k_0 is small.

Second, ORCLUS performs the following two iterations:

1. Merge pairs of clusters with the smallest, combined project energy until the number of clusters is down to k_p (determined by a parameter for the step size α).
2. Redistribute all data objects to the k_p clusters according to their respective, projected distance to each cluster center. An object is assigned to the cluster with the smallest, projected distance.

The above procedure terminates until $k_p = k$ with a parameter α to control the step size. If the step parameter is big, then a lot of merge occur in one iteration and the quality of final clusters is not guaranteed. If the step parameter is small, the execution time is increased.

A significant computational cost of the algorithm is from the decomposition of a data subset D_i. The complexity of such a decomposition is determined by the the number of dimension (attributes). Specifically, it costs $O(d^3)$. Moreover, the computation has to be done in each merger of two data subsets. The complexity of the ORCLUS algorithm is therefore as high as $k_0^3 + k_0 N d + k_0^2 d_0^2$.

A heuristic way of speeding up the ORCLUS algorithm is to make k_0 small and to conduct more merges in each step. However, the quality of clustering has been traded off. This problem is caused by the fact that ORCLUS is an agglomerative algorithm and too many merges are required to form a small number of clusters. In contrast, the divisive method needs much less steps to form clusters.

3 Divisive Projected clustering (DPCLUS)

Large computational costs of projected clustering lie in computing covariance matrixes and SVD (or PCA) decomposition. The computational costs of covariance matrixes and SVD (or PCA) decomposition is largely determined by the number of attributes, d, rather than the number of objects in a data set N_i.

In most applications, we have $k << m$ where m is the number of objects in the data set, and k is the number of clusters. Therefore, a top-down method (divisive method) needs significantly less number of computations of covariance matrixes and SVD (or PCA) decomposition than a bottom-up one (agglomerative method).

A key question is how to partition the data. Given a dataset, the projected cluster problem can be regarded as the one of partition of the dataset into k clusters such that the sum of the projected distance of each data object to its cluster centroid is minimized. Compared to the clustering in full-dimensional space, the projected clustering makes use of the projected distance instead of the full-dimensional distance. Recall that we need to find a best subspace and their associated subsets of data objects such that the sum of the projected distances of data objects to their centroids is minimized. The number of the projected clusters in our algorithm is given. So it is to determine only the directions of spanning vectors. Within one cluster the optimal direction of the vector to which its associated data objects are projected should reflect the minimal variance of these data. An eigenvalue is numerically related to the variance it captures. The higher the value, the more variance it has captured. The principal vector defines a projection that encapsulates the maximum amount of variation in a dataset. This principal vector is in fact the eigenvector with the highest corresponding eigenvalue.

We make use of the principle vector of eigenvectors. All data objects D are projected to the principle vector as discussed in the previous section, and the centroid separates data into two groups: D_1 and D_2. D_1 contains data objects whose projected values are greater than or equal to the means, and D_2 contains the rest.

The pseudo code of the algorithm is listed below.

DPCLUS algorithm (Divisive Projected clustering)
$Input$: data set D, cluster number k, subspace dimension l, the minimum object number $\min_N$, and the minimum distance for excluding outliers δ
$Output$: $\geq k$ projected clusters

initialise an empty tree T;
let the root of T store D;
where (the number of leaves of $T < k$)
foreach D_i stored in a leaf of the newest layer
Partition (D_i);
Redistribute (all data sets stored in the leaves of the newest layer);
output data sets stored in all leaves of T;

Function Partition(D_i)
if D_i satisfies Definition 1 or $|D_i| < \min_N$
then terminate the leaf storing D_i and *return*;
split D_i into D_{i1} and D_{i2} by the centroid in the principal vector;
insert two son leaves of the node storing D_i to store D_{i1} and D_{i2};

Function Redistribution(all data sets stored in the newest layer)
foreach data object **p** in all data sets stored in the newest layer
foreach data set D_j stored in the newest layer
compute the projected distance between **p** and the center of $\mathbf{D}_j$;
if projected distances of **p** to all data sets $> \delta$
then exclude **p** from future clustering;
else assign **p** to the data set with the smallest projected distance;

The DPCLUS algorithm partitions a data set into clusters in a top- down manner. The splitting point is the centroid of data objects projected to the principal vector. This saves a lot of computation for covariance matrixes and SVD decompositions as done in the ORCLUS algorithm. The sole dependence on the principal vector to separate data is rough and does not produce quality clusters. We design the $Redistribution$ function to minimise the projected energy of each clusters after clusters are formed by partitions.

The number of final clusters can be greater than k because the number of leaves is not tested until all data sets stored in the newest tree layer are split and redistributed.

Outliers affect the quality of final clusters very much since they change the orientations of data objects greatly. Some data objects may not belong to any cluster and are considered outliers. To deal with this problem, we set an outlier threshold in Redistribution step, say δ. When the projected distance of a data object to any cluster is greater than δ, then the data object is considered as an outlier and is excluded from the subsequent clustering.

We discuss the complexity of the algorithm in the following.

It is assumed that k denotes the number of final classes, N the number of data objects, d the dimension of the data set, and l the dimension of subspace.

The cost for partition is $kd^3/2$. d^3 comes from computing covariance matrices and SVD decomposition for a cluster. Since the partition is conducted in a binary way, the number of total partitions is k. Each partition requires a SVD decomposition to determine the subspace.

After the partition, each cluster has to be decomposed again to determine whether or not it satisfies the projected clustering requirement. If no, it will participate in redistribution. The number of such decomposition is $2k$, and hence the costs for the decompositions is $2kd^3$. All clusters are projected to l dimensional subspaces, and each data object has to be checked against each cluster. Therefore, the total costs for distribution is kNl.

In sum, the computational complexity for the DPCLUS algorithm is $O(3kd^3 + kNd)$. Note that we could not do much for term d^3 since it is for computing a covariance matrix and a SVD decomposition. However, the proposed algorithm has reduced the number of such computations significantly.

Our DPCLUS algorithm is faster than most exiting generalised projected clustering algorithms. We compare the time complexities in the table below.

Algorithms	Time complexities
ORCLUS [1]	$O(k_0^3 + k_0Nd + k_0^2d_0^2)$
$4C$ [4]	$O(d^2NlogN + d^3N)$(with data index)
	$O(d^2N^2 + d^3)$(without data index)
CURLER [5]	$O(k_0Nd^2 + k_0d^3) + O(Nl^2 + k_0^2), where k_0 > k$
HARP [7]	$O(d^2N^2 + N^2log^2N)$
DPCLUS	$O(3kd^3 + kNd)$

It should be noted that in order to speed up, some algorithms make use of techniques such as heuristics, small number of micro-clusters, and random samples. However, these techniques come with a price; that is, some of the clustering quality must be sacrificed.

4 Experimental evaluation

4.1 Efficiency comparison to ORCLUS

We use synthetic data sets for this experiment. More details about how these data sets were generated will be given in the following subsection.

For the test of scalability with the size of data sets, the data sets each contain 10 attributes and up to 300,000 objects. For the test of scalability with the number of attributes, the data sets each contain 100,000 objects and up to 50 attributes. The number of embedded clusters is fixed to 20 for the above two tests.

l is set to 10 for both methods. k_0 is set to $15 \times k$ to make ORCLUS efficient. δ for both methods is set as 0.01. $\min_N$ varies for different data sets, but is set as the same for both methods. A value less than 0.0001 is consider as 0 in the experiments to test the satisfaction of Definition 1.

Figure 1 shows that DPCLUS is more efficient than ORCLUS in large data sets as well as in high dimensional data sets. Consider that most computational time for projected clustering is spent on data decomposition, whose time complexity is cubic to the dimension and independent of the data set size. DPCLUST outperforms ORCLUS significantly in high dimensional data sets since it reduces the number of data decompositions significantly.

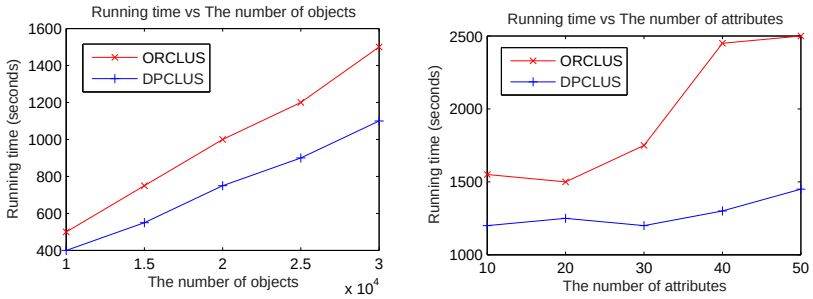

Fig. 1. The scalability of DPCLUS in comparison to that of ORCLUS

4.2 Clustering quality

To demonstrate the clustering quality of DPCLUS, we compare it to three clustering methods on a synthetic data set. We embedded 20 clusters that are correlated in some subspaces over a set of random data objects. The data set contains 10,000 data objects, with each object having 20 attributes. Each embedded cluster contains 250 data objects, which have 10% variations from the original pattern. Other 5,000 data objects are random data objects generated by the uniform distribution.

We set the parameters of DPCLUS as $l = 10$, $k = 20$, $\min_N = 50$, and $\delta = 0.01$. The results from DPCLUS are shown in Figure 2. DPCLUS is able to find all embedded clusters correctly. Although k is set as 20 in the experiment, the number of final clusters can be any integer number between 20 to 32, because the number of clusters is not tested until all data sets stored in the newest tree layer are split and redistributed. The number of the found clusters are greater than 20, since some clusters are split into two. For example, clusters at row 2: 1 and 2 are from the same cluster. DPCLUS has successfully identified cluster patterns from random data.

We set the parameters of ORCLUS as $k = 20$, $l = 10$, $k_0 = 350$, and $\delta = 0.01$. Figure 2 shows a good result. ORCLUS identified fewer than a half of embedded clusters with high quality. Since initial micro-clusters in ORCLUS is randomly chosen, the final clusters vary in different executions.

We further show that both *k*-means and hierarchical clustering methods failed to find quality clusters in such noise data in Figure 3. Data is sampled for hierarchical clustering method because of efficiency constraint.

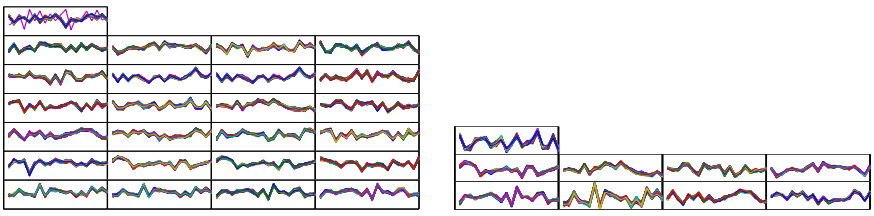

Fig. 2. Left: clusters found by DPCLUS, Right: clusters found by ORCLUS.

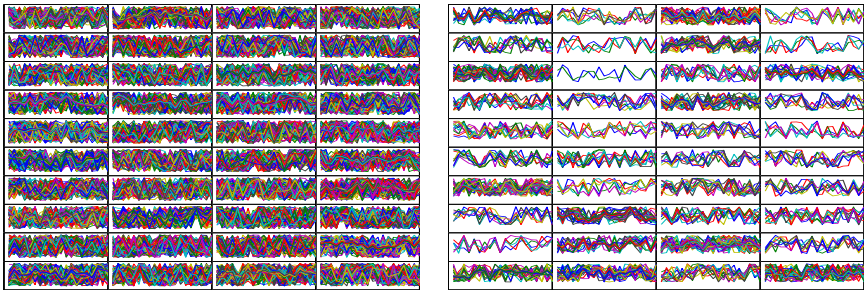

Fig. 3. Left: clusters found by kmeans. Right: Clusters found by hierarchical clustering.

5 Conclusions

We have presented a divisive, projected clustering algorithm for detecting correlation clusters in highly noised data. The distinction of noise points from correlated data points in a projected space offers benefits for projected clustering algorithms to discover clusters in noise data. The proposed algorithm mainly explores this potential. Further, the proposed algorithm is faster than most existing general projected clustering algorithms, which are agglomerative clustering ones. Unlike those agglomerative algorithms, the produced clusters by the proposed algorithm do not rely on the choice of randomly generated initial seeds, and are completely determined by the data distribution. We experimentally show that the proposed algorithm is faster and more scalable than than ORCLUS, a well-known agglomerative projected clustering, and that the proposed algorithm detects correlation clusters in noise data better than ORCLUS.

References

1. C. C. Aggarwal and P. S. Yu. Finding generalized projected clusters in high dimensional spaces. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data (SIGMOD)*, pages 70–81, 2000.
2. C. C. Aggarwal, C. Procopiuc, J. L. Wolf, P. S. Yu, and J. S. Park. Fast algorithms for projected clustering. In *Proceedings of the ACM SIGMOD Conference*, pages 61-72, 1999.
3. R. J. Aggrawal, J. Gehrke, D.Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proceedings of the ACM SIGMOD Conference*, pages 94-105, 1998, Seattle, WA.
4. C. Bhm, K. Kailing, P. Krger, and A. Zimek. Computing Clusters of Correlation Connected objects. In *Proceedings of the ACM SIGMOD international conference on Management of data*, June 13-18,2004, Paris, France.
5. A. K. H. Tung, X. Xu, and B. C. Ooi. CURLER: finding and visualizing nonlinear correlation clusters. In *Proceedings of the ACM SIGMOD international conference on Management of data*,pages 467-478, 2005.
6. E. Ng, A. Fu, and R. Wong. Projective Clustering by Histograms. *IEEE Transactions on Knowledge and Data Engineering*, Vol. 17, No. 3, pages 369-383, March 2005.
7. K. Y. Yip, D. W. Cheung, and M. K. Ng. HARP: A Practical Projected Clustering Algorithm. *IEEE Transactions on Knowledge and Data Engineering* , Vol. 16, No. 11, pp. 1387-1397, Nov. 2004.
8. G. Taguchi, R. J. Taguchi, and R. Jugulum. *The Mahalanobis-Taguchi Strategy: A Pattern Technology System*. John Wiley & Sons, 2002.