Clustering Gene Expression Data in SQL Using Locally Adaptive Metrics

Dimitris Papadopoulos
UC Riverside
dimitris@cs.ucr.edu

Carlotta Domeniconi
George Mason University
carlotta@ise.gmu.edu

Dimitrios Gunopulos*
UC Riverside
dg@cs.ucr.edu

Sheng Ma
IBM T.J. Watson Research Center
shengma@us.ibm.com

ABSTRACT

The clustering problem concerns the discovery of homogeneous groups of data according to a certain similarity measure. Clustering suffers from the curse of dimensionality. It is not meaningful to look for clusters in high dimensional spaces as the average density of points anywhere in input space is likely to be low. As a consequence, distance functions that equally use all input features may be ineffective. We introduce an algorithm that discovers clusters in subspaces spanned by different combinations of dimensions via local weightings of features. This approach avoids the risk of loss of information encountered in global dimensionality reduction techniques. Our method associates to each cluster a weight vector, whose values capture the relevance of features within the corresponding cluster. In this paper we present an efficient SQL implementation of our algorithm, that enables the discovery of clusters on data residing inside a relational DBMS.

1. INTRODUCTION

The clustering problem concerns the discovery of homogeneous groups of data according to a certain similarity measure. It has been studied extensively in statistics [2], machine learning [5; 13], and database communities [14; 10; 20]. Clustering suffers from the curse of dimensionality problem in high dimensional spaces. In high dimensional spaces, it is highly likely that, for any given pair of points within the same cluster, there exist at least a few dimensions on which the points are far apart from each other. It is not meaningful to look for clusters in such a high dimensional space as the average density of points anywhere in input space is likely to be low. As a consequence, distance functions that equally use all input features may be ineffective.

The problem of high dimensionality can be addressed by requiring the user to specify a subspace (i.e., subset of dimensions) for cluster analysis. However, the identification of subspaces by the user is an error-prone process. More importantly, correlations that identify clusters in the data are likely not to be known by the user. Indeed, we desire such correlations, and induced subspaces, to be part of the findings of the clustering process itself.

In order to capture the local correlations of data, a proper feature selection procedure should operate locally in input space. In this paper we propose a *soft* feature selection procedure that assigns (local) weights to features according to the local correlations of data along each dimension. Dimensions along which data are loosely correlated receive a small weight, that has the effect of elongating distances along that dimension. Features along which data are strongly correlated receive a large weight, that has the effect of constricting distances along that dimension. Figures 1 give a simple example. The left plot depicts two clusters of data elongated along the x and y dimensions. The right plot shows the same clusters, where within-cluster distances between points are computed using the respective local weights generated by our algorithm. The weight values reflect local correlations of data, and reshape each cluster as a *dense spherical cloud*. This directional local reshaping of distances better separates clusters, and allows for the discovery of different patterns in different subspaces of the original input space.

The data-mining task of clustering ideally can be performed inside a relational DBMS. By relying on a DBMS to manage the data, the clustering application is freed from this task. Also, implementing a clustering algorithm in SQL, makes the programming task easier. Finally, since SQL is an industry standard and available on all major DBMS's, the implementation of the clustering algorithm is portable.

In this paper we formalize the problem of finding different clusters in different subspaces. Our algorithm (Locally Adaptive Clustering, or LAC) discovers clusters in subspaces spanned by different combinations of dimensions via local weightings of features. This approach avoids the risk of loss of information encountered in global dimensionality reduction techniques. We give a concrete SQL implementation of our algorithm, that exhibits good performance and scalability. These are the key considerations for deploying a clustering algorithm inside a DBMS, especially when dealing with very large datasets that must reside in disk space, or if we want to take advantage of computer clusters.

2. RELATED WORK

Local dimensionality reduction approaches for the purpose of efficiently indexing high dimensional spaces have been recently discussed in the database literature [12; 4; 17]. In general, the efficacy of these methods depends on how the clustering problem is addressed in the first place in the original feature space. A potential serious problem with such techniques is the lack of data to locally perform PCA on each cluster to derive the principal components. Moreover, for clustering purposes, the new dimensions may be diffi-

---

*Supported by NSF CAREER Award 9984729,NSF IIS-9907477, and the DoD.

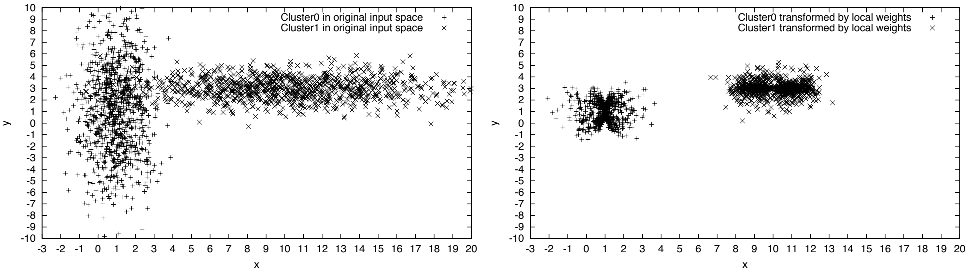

Figure 1: (Left)Clusters in original input space. (Right) Clusters transformed by local weights.

cult to interpret, making it hard to understand clusters in relation to the original space.

The problem of finding different clusters in different subspaces of the original input space has been addressed in [16]. While the work in [16] successfully introduces a methodology for looking at different subspaces for different clusters, it does not compute a partitioning of the data into disjoint groups. The reported dense regions largely overlap, since for a given dense region all its projections on lower dimensionality subspaces are also dense, and they all get reported. On the other hand, for many applications such as customer segmentation and trend analysis, a partition of the data is desirable since it provides a clear interpretability of the results.

Recently [15], another density-based projective clustering algorithm (DOC) has been proposed. This approach pursues an optimality criterion defined in terms of density of each cluster in its corresponding subspace. A Monte Carlo procedure is then developed to approximate with high probability an optimal projective cluster.

[9] also addresses the problem of feature selection to find clusters hidden in high dimensional data. The authors search through feature subset spaces, evaluating each subset by first clustering in the corresponding subspace, and then evaluating the resulting clusters and feature subset using the chosen feature selection criterion. We observe that dimensionality reduction is performed globally in this case. Therefore, the technique in [9] is expected to be effective when a data set contains some relevant features and some irrelevant (noisy) ones, across all clusters.

The problem of finding different clusters in different subspaces is also addressed in [1]. The proposed algorithm (PROjected CLUStering) seeks subsets of dimensions such that the points are closely clustered in the corresponding spanned subspaces. Both the number of clusters and the average number of dimensions per cluster are user-defined parameters. PROCLUS starts with choosing a random set of medoids, and then progressively improves the quality of medoids by performing an iterative hill climbing procedure that discards the 'bad' medoids from the current set. In order to find the set of dimensions that matter the most for each cluster, the algorithm selects the dimensions along which the points have the smallest average distance from the current medoid. The authors do not prove whether or not the algorithm converges to the optimality criterion they choose.

In contrast to the PROCLUS algorithm, our method does not require to specify the average number of dimensions to be kept per cluster. For each cluster, in fact, *all* features are taken into consideration, but properly weighted. The PROCLUS algorithm is more prone to loss of information if the number of dimensions is not properly chosen. For example, if data of two clusters in two dimensions are distributed as in Figure 1 (Left), PROCLUS may find that feature x is the most important for cluster 0, and feature y is the most important for cluster 1. But projecting cluster 1 along the y dimension doesn't allow to properly separate points of the two clusters. We avoid this problem by keeping both dimensions for both clusters, and properly weighting distances along each feature within each cluster. In addition, while the authors in [1] do not prove whether their algorithm converges to the chosen optimality criterion, we can formally show that our algorithm converges to a local minimum of the associated error function.

Generative approaches have also been developed for local dimensionality reduction and clustering. The approach in [11] makes use of maximum likelihood factor analysis to model local correlations between features.

[18] extends the single PCA model to a mixture of local linear sub-models to capture nonlinear structure in the data. A mixture of principal component analyzers model is derived as a solution to a maximum-likelihood problem. An EM algorithm is formulated to estimate the parameters.

3. BICLUSTERING OF GENE EXPRESSION DATA

Microarray technology is one of the latest breakthroughs in experimental molecular biology. Gene expression data are generated by DNA chips and other microarray techniques, and they are often presented as matrices of expression levels of genes under different conditions (e.g., environment, individuals, tissues). Each row corresponds to a gene, and each column represents a condition under which the gene is developed.

Biologists are interested in finding set of genes showing strikingly similar up-regulation and down-regulation under a set of conditions. To this extent, recently the concept of *bicluster* has been introduced [6]. A bicluster is a subset of genes and a subset of conditions with a high similarity score. A particular score that applies to expression data is the *mean squared residue score* [6]. Let I and J be subsets of genes and experiments respectively. The pair (I, J) specifies a submatrix A_{IJ} with a mean squared residue score defined as follows:

$$H(I,J) = \frac{1}{|I||J|} \sum_{i\in I, j\in J} (a_{ij} - a_{iJ} - a_{Ji} + a_{IJ})^2, \qquad (1)$$

where $a_{iJ} = \frac{1}{|J|}\sum_{j\in J} a_{ij}$, $a_{Ij} = \frac{1}{|I|}\sum_{i\in I} a_{ij}$, and $a_{IJ} =$

$\frac{1}{|I||J|}\sum_{i\in I,j\in J}a_{ij}$. They represent the row and column means, and the mean of the submatrix, respectively. The lowest score $H(I,J)=0$ indicates that the gene expression levels fluctuate in unison. The aim is then to find biclusters with low mean squared residue score (below a certain threshold). We observe that the mean squared residue score is minimized when subsets of genes and experiments (or dimensions) are chosen so that the gene vectors (i.e., rows of the resulting bicluster) are close to each other with respect to the Euclidean distance. As a result, the LAC algorithm, and other subspace clustering algorithms, are well suited to perform simultaneous clustering of both genes and conditions in a microarray data matrix.

[19] introduces an algorithm (pCluster) for clustering similar patterns, that has been applied to DNA microarray data of a type of yeast. The pCluster model optimizes a criterion that is different from the mean squared residue score, as it looks for coherent patterns on a subset of dimensions (e.g., in an identified subspace, objects reveal larger values for the second dimension than for the first).

4. PROBLEM STATEMENT

We define what we call *weighted cluster*. Consider a set of points in some space of dimensionality N. A *weighted cluster* C is a subset of data points, together with a vector of weights $\mathbf{w}=(w_1,\ldots,w_N)$, such that the points in C are closely clustered according to the L_2 norm distance weighted using $\mathbf{w}$. The component w_j measures the degree of correlation of points in C along feature j. The problem becomes now how to estimate the weight vector $\mathbf{w}$ for each cluster in the data set.

In this setting, the concept of *cluster* is not based only on points, but also involves a weighted distance metric, i.e., clusters are discovered in spaces transformed by $\mathbf{w}$. Each cluster is associated with its own $\mathbf{w}$, that reflects the correlation of points in the cluster itself. The effect of $\mathbf{w}$ is to transform distances so that the associated cluster is reshaped into a dense hyper-sphere of points separated from other data.

In traditional clustering, the partition of a set of points is induced by a set of *representative* vectors, also called *centroids* or *centers*. The partition induced by discovering weighted clusters is formally defined as follows.

Definition: Given a set S of D points $\mathbf{x}$ in the N-dimensional Euclidean space, a set of k centers $\{\mathbf{c}_1,\ldots,\mathbf{c}_k\}$, $\mathbf{c}_j\in\Re^N$, $j=1,\ldots,k$, coupled with a set of corresponding weight vectors $\{\mathbf{w}_1,\ldots,\mathbf{w}_k\}$, $\mathbf{w}_j\in\Re^N$, $j=1,\ldots,k$, partition S into k sets $\{S_1,\ldots,S_k\}$:

$$S_j=\{\mathbf{x}|(\sum_{i=1}^{N}w_{ji}(x_i-c_{ji})^2)^{1/2}<(\sum_{i=1}^{N}w_{li}(x_i-c_{li})^2)^{1/2},\forall l\neq j\},\tag{2}$$

where w_{ji} and c_{ji} represent the ith components of vectors $\mathbf{w}_j$ and $\mathbf{c}_j$ respectively (ties are broken randomly).

The set of centers and weights is *optimal* with respect to the Euclidean norm, if they minimize the error measure:

$$E_1(C,W)=\sum_{j=1}^{k}\sum_{i=1}^{N}w_{ji}e^{-X_{ji}}\tag{3}$$

subject to the constraints $\sum_{i=1}^{N}w_{ji}^2=1\ \forall j$. C and W are $(N\times k)$ matrices whose column vectors are $\mathbf{c}_j$ and $\mathbf{w}_j$ respectively, i.e. $C=[\mathbf{c}_1\ldots\mathbf{c}_k]$ and $W=[\mathbf{w}_1\ldots\mathbf{w}_k]$. X_{ji} represents the average distance from the centroid $\mathbf{c}_j$ of points in cluster j along dimension i, and is defined as follows:

$$X_{ji}=\frac{1}{|S_j|}\sum_{\mathbf{x}\in S_j}(c_{ji}-x_i)^2.$$

The exponential function in (3) has the effect of making the weights w_{ji} more sensitive to changes in X_{ji}, and therefore to changes in local feature relevance. This allows larger error improvements as we adapt the values of weights and centers, and therefore a faster computation to achieve spherically shaped clusters (separated from each other) in the space transformed by optimal weights (see Figure 1).

In the following we present an algorithm that finds a solution (set of centers and weights) that is a local minimum of the error function (3).

5. LOCALLY ADAPTIVE CLUSTERING ALGORITHM

This section describes our feature relevance estimation procedure that assigns local weights to features according to the local correlation of data along each dimension. Our technique progressively improves the quality of initial centroids and weights, by investigating the space near the centers to estimate the dimensions that matter the most, i.e. the dimensions along which local data are mostly correlated. Specifically, we proceed as follows.

We start with *well-scattered* points in D as the k centroids: we choose the first centroid at random, and select the others so that they are far from one another, and from the first chosen center. We initially set the values of weights to 1. Given the initial centroids $\mathbf{c}_j$, for $j=1,\ldots,k$, we compute the corresponding sets S_j as defined in (2), where $w_{ji}=1\ \forall j$ and $\forall i$. We then compute the average distance along each dimension from the points in S_j to $\mathbf{c}_j$. Let X_{ji} denote this average distance along dimension i. The smaller X_{ji} is, the larger is the correlation of points along dimension i. We use the value X_{ji} in an exponential weighting scheme to credit weights to features (and to clusters):

$$w_{ji}=exp(-h\times X_{ji})/(\sum_{l=1}^{N}(exp(-h\times 2\times X_{jl})))^{1/2}\tag{4}$$

where h is a parameter that can be chosen to maximize (minimize) the influence of X_{ji} on w_{ji}. When $h=0$ we have $w_{ji}=1/N$, thereby ignoring any difference between the X_{ji}. On the other hand, when h is large a change in X_{ji} will be exponentially reflected in w_{ji}. We empirically determine the value of h through cross-validation in our experiments with simulated data. We set the value of h to 9 in the experiments with real data. The exponential weighting is more sensitive to changes in local feature relevance [3] and gives rise to better performance improvement. In fact, it is more stable because it prevents distances from extending infinitely in any direction, i.e., zero weight. This, however, can occur when either linear or quadratic weighting is used.

The weights w_{ji} enable to elongate distances along less important dimensions, i.e. dimensions along which points are loosely correlated, and, at the same time, to constrict distances along the most influential ones, i.e. features along which points are strongly correlated. Note that the technique is centroid-based because weightings depend on the centroid.

The computed weights are used to update the sets S_j, and therefore the centroids' coordinates. The procedure is iterated until convergence is reached, i.e. no change in centers' coordinates is observed.

The resulting algorithm, that we call LAC (Locally Adaptive Clustering), is summarized in the following.

Input: Set D of points $\mathbf{x} \in R^N$, and the number of clusters k.

1. Start with k initial centroids $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_k$;
2. Set $w_{ji} = 1$, for each centroid $\mathbf{c}_j$, $j = 1, \ldots, k$ and each feature $i = 1, \ldots, N$;
3. For each centroid $\mathbf{c}_j$, $j = 1, \ldots, k$, and for each data point $\mathbf{x}$:
    - Set $S_j = \{\mathbf{x} | j = arg\min_l WDist(\mathbf{c}_l, \mathbf{x})\}$, where $WDist(\mathbf{c}_l, \mathbf{x}) = (\sum_{i=1}^N w_{li}(c_{li} - x_i)^2)^{1/2}$;
4. **Compute new weights.** For each centroid $\mathbf{c}_j$, $j = 1, \ldots, k$, and for each feature i:
    - Set $X_{ji} = \sum_{\mathbf{x} \in S_j} (c_{ji} - x_i)^2 / |S_j|$, where $|S_j|$ is the cardinality of set S_j (X_{ji} represents the average distance of points in S_j from $\mathbf{c}_j$ along feature i);
    - Set $w_{ji} = exp(-h \times X_{ji}) / (\sum_{l=1}^N exp(-h \times 2 \times X_{jl}))^{1/2}$;
5. For each centroid $\mathbf{c}_j$, $j = 1, \ldots, k$, and for each data point $\mathbf{x}$:
    - Recompute $S_j = \{\mathbf{x} | j = arg\min_l WDist(\mathbf{c}_l, \mathbf{x})\}$;
6. **Compute new centroids.** Set $\mathbf{c}_j = \frac{\sum_{\mathbf{x}} \mathbf{x} 1_{S_j}(\mathbf{x})}{\sum_{\mathbf{x}} 1_{S_j}(\mathbf{x})}$, for each $j = 1, \ldots, k$, where $1_S(.)$ is the indicator function of set S;
7. Iterate 3,4,5,6 until convergence (i.e., no change in centroids' coordinates)

The sequential structure of the LAC algorithm is analogous to the mathematics of the *EM* algorithm [7]. The hidden variables are the assignments of the points to the centroids. Step 3 constitutes the *E* step of the *EM* algorithm: it finds the values of the hidden variables S_j given the previous values of the parameters w_{ji} and c_{ji}. The following step (*M* step) consists in finding new matrices of weights and centroids that minimize the error function with respect to the current estimation of hidden variables.

We can also prove the following [8]:

Theorem. The LAC algorithm converges to a local minimum of the error function (3).

6. IMPLEMENTING LAC IN SQL

In the implementation of LAC in SQL we used the tables shown in Table 1. Next we describe in detail each step of our algorithm presented in Section 5, expressed in SQL.

```
INSERT INTO S
WITH T1 (id, cid, wd) AS (
SELECT DATA.id,
C.cid,
SQRT(W.x1*(C.x1-DATA.x1)**2+..+W.xn*(C.xn-DATA.xn)**2)
FROM  DATA, C, W
WHERE C.cid = W.cid
),
T2 AS (
SELECT T1.id,  MIN(wd) AS mwd
FROM T1
GROUP BY T1.id
)
SELECT T2.id,  T1.cid
FROM T1, T2
WHERE T2.mwd = T1.wd AND T2.id = T1.id ;
```

Figure 2: SQL: Assign points (DATA) to clusters (S), calculating the weighted distances on the fly

```
INSERT INTO CARD_S
SELECT CID, COUNT(*)
FROM S
GROUP BY CID ;
```

Figure 3: SQL: Calculate the cardinalities of sets S_j.

In Figure 2 we give the query that populates table $S(id, cid)$, where *id* is the point id, and *cid* is the cluster id. This query implements steps 3 and 5 of the LAC algorithm. The query joins the dataset table $DATA$, with the centroids' table C and weights' table W, in order to calculate the weighted distances of each point to each cluster centroid. The result of the join goes to the temporary table $T1$. Then, in the second table $T2$, for each point, we calculate the minimum distance of the point at hand from all clusters. The final result (i.e. table S) is given through a join of $T1$ and $T2$. Table $T1$ has size of kD, while $T2$ has a size of D (and so does table S), where k is number of clusters and D is the number of points. Note that we can reduce the space used by the temporary tables by opening a cursor on table $DATA$, and for each tuple, calculate the k distances, and store the point, along with the cluster id corresponding to the minimum distance, directly into table S. Thus, the space requirement is the actual size of table S, i.e. D.

Figure 3 shows the query that computes the cardinalities of sets $S_j, j = 1 \ldots k$. The sets are stored in table $S(id, cid)$. The computation is straightforward: it is a count of rows in S, grouped by the cluster id. The result is stored into table $CARD\_S$, and the values are used in steps 4 and 6 of LAC.

In Figure 4 we give the query that computes the average distance of points in S_j from c_j. This query implements the first part of step 4, where the values $X_{ji} = \sum_{\mathbf{x} \in S_j} (c_{ji} - x_i)^2 / |S_j|$ are computed. The result set of this query is kept into table $X(cid, d_1, \ldots, d_n, sum\_exp)$. Column sum_exp is calculated as a function of the n preceding columns, and it is used in second part of step 4, where the weights are updated. We chose to actually store this sum, in order to avoid having long expressions in the computation of the weights. The query joins tables C, $DATA$ and S on the cluster id and the point id, and presents the result by grouping the resulting rows on the cluster id.

The second part of step 4 of LAC is presented in Figure 5. The query uses the results stored in table X. Note that

Table	PK	Columns	Cardinality	Contents
$DATA$	id	$x1,\ldots,xn$	D	data points
W	cid	$x1,\ldots,xn$	k	weights
C	cid	$x1,\ldots,xn$	k	centroids
S	id, cid		D	sets S_j
X	cid	$d1,\ldots,dn, sum\_exp$	k	avg distances
$CARD\_S$	cid	$card$	k	card. of S_j

Table 1: Tables' descriptions

```
INSERT INTO X
WITH T1(cid, s1,..,sn) AS
(
SELECT S.cid ,
SUM((C.x1-DATA.x1)**2),...,SUM((C.xn-DATA.xn)**2)
FROM C, DATA, S
WHERE C.cid = S.cid
AND S.id = DATA.id
GROUP by S.cid
)
SELECT T1.cid ,
s1/CARD_S.card,
...
sn/CARD_s.card, 0
FROM T1, CARD_S
WHERE T1.cid = CARD_S.cid ;

UPDATE X
SET sum_exp = EXP(-h * X.d1) + ... + EXP( -h * X.dn) ;
```

Figure 4: SQL: Compute the distances X_{ji}

```
INSERT INTO W
SELECT X.cid,
EXP(-h * X.d1) / SQRT(X.sum_exp),
...
EXP( -h * X.dn) / SQRT(X.sum_exp)
FROM X;
```

Figure 5: SQL: Compute the weights (W)

parameter h is actually substituted by a scalar value, during the dynamic invocation of the queries by our control program.

In Figure 6 the query that computes the centroids is presented. This is step 6 of the LAC algorithm. The query first joins tables $DATA$, S and C, in order to compute the sums of the attributes of points that belong to the same cluster. This intermediate results goes to temporary table $SUMS$, and has size k. Then, tables $SUMS$ and $CARD\_S$ are joined, in order to compute the final result, i.e. the coordinates of the new centroids. Note that we cannot avoid the second join (which is performed on small tables, anyway), because all the free columns that appear in a SELECT clause and are not involved in an aggregation function (SUM in our case) have to appear in the GROUP BY clause. In our case, we perform a GROUP BY on column $S.cid$ only, so we cannot use column $CARD\_S.card$ in the SELECT clause that creates table $SUMS$. Hence the need for the second join between tables SUM and $CARD\_S$.

```
INSERT INTO C
WITH SUMS(cid, s1,..,sn) AS
(
SELECT C.cid, SUM(DATA.x1),...,SUM(data.xn)
FROM DATA, C, S
WHERE C.cid = S.cid
AND S.id = DATA.id
GROUP BY C.cid
)
SELECT  SUMS.cid ,
s1/CARD_S.card AS new_x1,
....
sn /CARD_S.card AS new_xn
FROM SUMS,  CARD_S
WHERE CARD_S.cid = SUMS.cid ;
```

Figure 6: SQL: Compute new centroids (C)

7. EXPERIMENTAL EVALUATION

In this section we present an experimental evaluation of the SQL implementation of LAC. The experiments were run on a desktop PC having a P4 processor running at 1.6GH and 1G of RAM. The hosting operating system was Linux Mandrake 8.1 (kernel ver. 2.4.17) and we we used DB2 v8.1.

We created synthetic datasets having dimensionality $N = 5$ and we embedded $k = 5$ clusters. The points' coordinates spanned within $[0, 100]$ on each dimension. The cardinalities of the datasets D were $100k$, $500K$ and $1M$ tuples. Figure 7 shows the execution times per iteration. The algorithm converges, on average, after 5 to 7 iterations. We can see that the execution times scale linearly with the size of the database. The algorithm's error (fraction of misclassified objects over total number of objects) for the three datasets (having $100k$, $500k$ and $1M$ tuples) was 6.16%, 6.09% and 6.5%, respectively.

In order to evaluate the performance of LAC with respect to the dimensionality of the data, we generated a second set of datasets having dimensionality N equal to 10, 20 and 30. We fixed the cardinality D to $100k$ tuples and we embedded $k = 5$ clusters. The execution times per iteration, for these datasets, is show in Figure 8. The algorithm's error for the three datasets was 8.4%, 4.16% and 4.06%, respectively.

Furthermore, in order to check the performance while varying the number of embedded clusters, datasets with k equal to 5 , 10 and 20 were used. The cardinality D was fixed to $100k$ tuples, while the dimensionality N of the data was set to 5. Figure 9 depicts the execution time per iteration for these sets of data. The algorithm's error for the three datasets was 6.16%, 4.26% and 3.77%, respectively.

The basic characteristic of microarray DNA expression data is the large number of conditions, under which the probes are done. In the form of a relation table, the genes correspond to rows and the conditions to columns (attributes). In order to examine the scalability of the SQL implementation

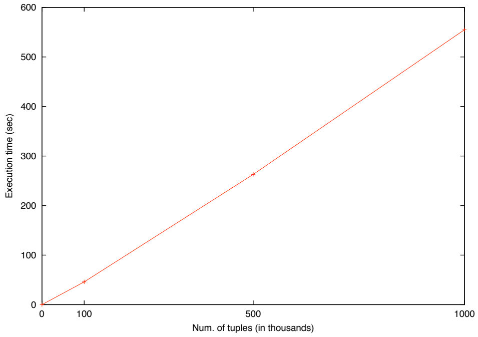

Figure 7: Synthetic datasets: time per iteration with varying no. of tuples, N=5, k=5

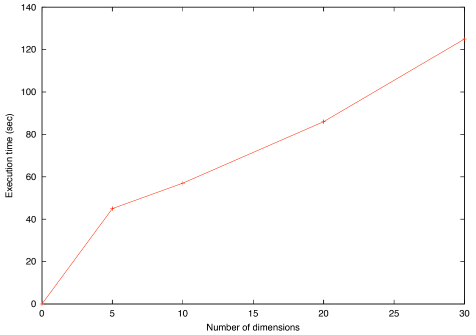

Figure 8: Synthetic datasets: time per iteration with varying no. of dimensions, D=100k, k=5

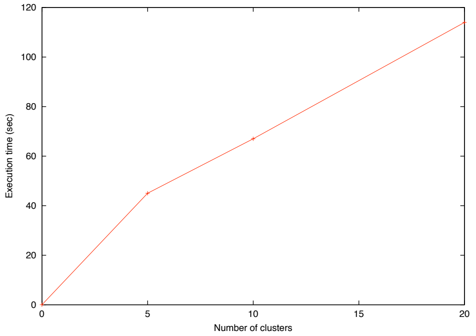

Figure 9: Synthetic datasets: time per iteration with varying no. of clusters, D=100k, N=5

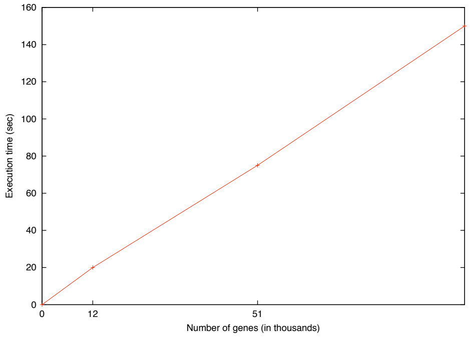

Figure 10: Biological datasets: time per iteration with varying no. of genes, N=22

of LAC, we did the following: we used a DNA microarray of gene expression profiles in hereditary breast cancer[1], which has 3226 genes and 22 conditions, as a basis, and we replicated it 4, 16 and 31 times, in order to get larger sets, having approximately 12K, 51K and 100K tuples, respectively. We ran LAC with k=4. The results show that within two biclusters five conditions (having id's 7,8,9,19,22 in the first one, and 7,8,9,13,22 in the second one) receive considerably larger weight than the others. The remaining biclusters contain fewer genes, and all conditions receive equal weights. This indicates that no correlation was found among those conditions. We observe that different combinations of conditions are selected for different biclusters, as also expected from a biological perspective. Figure 10 presents the execution times per iteration for this set of experiments. We observe that the algorithm scales up linearly with respect to database size, i.e. number of genes.

8. CONCLUSIONS

We presented an algorithm that discovers clusters in subspaces by different combinations of dimensions via local weightings of features. This method avoids the risk of loss of information encountered in global dimensionality reduction techniques. We implemented our algorithm using SQL in order to facilitate the discovery of clusters on datasets stored in a DBMS. The experimental evaluation shows the scalability of our algorithm.

9. REFERENCES

[1] C. Aggarwal, C. Procopiuc, J. L. Wolf, P. S. Yu, and J. S. Park. Fast Algorithms for Projected Clustering. In *Proc. ACM SIGMOD*, 1999.

[2] P. Arabie and L. Hubert. An overview of combinatorial data analysis. Clustering and Classification. *World Scientific Pub.*, pages 5–63, 1996.

[3] L. bottou and V. Vapnik. Local Learning algorithms. *Neural Computation*, 4(6):888–900, 1992.

[4] K. Chakrabarti and S. Mehrotra. Local dimensionality reduction: A new approach to indexing high dimensional spaces. *In Proc. VLDB*, 2000.

---

[1] http://research.nhgri.nih.gov/microarray/NEJM_Supplement

[5] P. Cheeseman and J. Stutz. *Bayessian Classification (autoclass): Theory and Results*, chapter 6. AAAI/MIT Press, 1996.

[6] Y. Cheng and G. M. Church. Biclustering of expression data. In *Proc. ISMB*, pages 93–103, 2000.

[7] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society*, 39(1):1–38, 1997.

[8] C. Domeniconi. *Locally Adaptive Techniques for Patern Classification*. PhD thesis, UC Riverside, Aug 2002.

[9] J. G. Dy and C. E. Brodley. Feature Subset Selection and Order Identification for Unsupervised Learning. In *Proc. ICML*, 2000.

[10] M. Ester, H. P. Kriegel, and X. Xu. A database interface for clustering in large spatial databases. In *Proc. KDD*, 1995.

[11] Z. Ghahramani and G. E. Hinton. The EM Algorithm for Mixtures of Factor Analyzers. Technical Report CRG-TR-96-1, Dept. of Computer Science, Univ. of Toronto, 1996.

[12] E. Keogh, K. Chakrabarti, S. Mehrotra, and M. Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. In *Proc. of ACM SIGMOD*, 2001.

[13] R. Michalski and R. Stepp. *Machine Learning: An Artificial Intelligence Approach*, chapter 'Learning from observation: Conceptual Clustering'. IOGA Publishing Co., 1983.

[14] R. T. Ng and J. Han. Efficient and Effective Clustering Methods for Spatial Data Mining. In *Proc. of VLDB*, 1994.

[15] C. M. Procopiuc, M. Jones, P. K. Agarwal, and T. M. Murali. A Monte Carlo algorithm for fast projective clustering. In *Proc. ACM SIGMOD*, 2002.

[16] D. G. R. Agrawal, J. Gehrke and P. Raghavan. Automatic Subspace Clustering of High Dimensional Data For Data Mining Applications. In *Proceedings of ACM SIGMOD*, pages 94–105, June 1998.

[17] A. Thomasian, V. Castello, and C. S. Li. Clustering and singular value decomposition for approximate indexing in high dimensional spaces. In *Proc. CIKM*, 1998.

[18] M. E. Tipping and C. M. Bishop. Mixtures of Principal Component Analyzers. *Neural Computation*, 1(2), 1999.

[19] H. Wang, W. Wang, J. Yang, and P. S. Yu. Clustering by pattern similarity in large data sets. In *Proc. ACM SIGMOD*, 2002.

[20] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An Efficient Data Clustering Method for Very Large Databases. In *Proc. ACM SIGMOD*, 1996.