Proceedings of the 2007 IEEE Symposium on
Computational Intelligence and Data Mining (CIDM 2007)

From similarity retrieval to cluster analysis: The case of R*-trees

Jiaxiong Pi, Yong Shi, *Senior member, IEEE* and Zhengxin Chen, *Member, IEEE*

Abstract— Data mining is concerned with important aspects related to both database techniques and AI/machine learning mechanisms, and provides an excellent opportunity for exploring the interesting relationship between retrieval and inference/reasoning, a fundamental issue concerning the nature of data mining. In the data mining context, this relationship can be restated as connection and differences between data retrieval and data mining. In this paper we explore this relationship by examining time series data indexed through R*-trees, and study the issues of (1) retrieval of data similar to a given query (which is a plain data retrieval task), and (2) clustering of the data based on similarity (which is a data mining task). Along the way of examination of our central theme, we also report new algorithms and new results related to these two issues. We have developed a software package consisting of a similarity analysis tool and two implemented clustering algorithms: KMeans-R and Hierarchy-R. A sketch of experimental results is also provided.

I. Introduction

The issue of using R*-trees for similarity analysis and cluster analysis is motivated from the following two observations:
(1) In order to make data mining manageable, data mining has to be database-centered. Yet, data mining goes beyond traditional realm of database techniques; in particular, reasoning methods developed from machine learning techniques and other fields in artificial intelligence (AI) have made important contributions in data mining. For example, since data mining is intended to discover hidden knowledge patterns, many data mining methods are based on inductive inference. Therefore, data mining offers an excellent opportunity to explore the interesting fundamental issue of the relationship between data/knowledge retrieval and inference/reasoning. Decades ago researchers made an important remark stating that since knowledge retrieval must respect the semantics of the representation language and therefore knowledge retrieval is a limited form of inference operating on the stored facts [5]. In addition, the *inverse* side of this statement has also been explored, which views inference as an extension of retrieval. For example, [3] described a computer model which is able to generate suggestions through document structure mapping based on the notion of reasoning as extended knowledge retrieval; the model was implemented using a relational approach. In the context of data mining, the relationship between retrieval and inference can be stated as an examination of connection and differences between data retrieval and data mining. However, although the issue of foundations of data mining has attracted much attention among data mining researchers [10], little work has been done in this respect. A possible reason of lacking such kind of research is the difficulty of identifying an appropriate common ground which can be used to examine both data retrieval and data mining.

(2) As our second observation, we point out that an important approach to achieve efficient data mining is by exploiting important features of database primitives. For example, as a multidimensional index structure for spatial data, R* tree [2] is a powerful database primitive. A rich literature exists in regard to the application of R*-trees for data mining (e.g., [8]). Since R*-tree was originally developed for spatial data retrieval, such kind of development reveals that R* tree structure can serve as a common ground to explore the relationship between retrieval and mining as discussed above.

In this paper, we take our first step to explore this interesting issue. We examine time series data indexed through R*-trees, and study the issues of (1) retrieval of data similar to a given query (which is a plain data retrieval task), and (2) clustering of the data based on similarity (which is a data mining task). Along the way of examination of our central theme, we also report new algorithms and new results related to these two issues. We have developed a software package consisting of components to handle these two tasks. We describe both parts of our work, with an emphasis on dealing with the challenges of moving from retrieving individual queries similar to a given query, to clustering the entire data set based on similarity.

II. Background

Just like a B-Tree, an R-Tree [6] relies on a balanced hierarchical structure, in which each tree node is mapped to a disk page. However, whereas B-Trees are built on single-value keys and rely on a total order on these keys, R-Trees

J. Pi is with University of Nebraska at Omaha, Omaha, NE 68182 (email: jpi@mail.unomaha.edu).
Y. Shi is with Data Technology and Knowledge Economy Center of Chinese Academy of Sciences, Graduate University of Chinese Academy of Science,Beijing 100080, China (email: yshi@gucas.ac.cn) and University of Nebraska at Omaha, Omaha, NE 68182 (email: yshi@mail.unomaha.edu).
Z. Chen is with University of Nebraska at Omaha, Omaha, NE 68182 (email: zchen@mail.unomaha.edu).

organize rectangles according to a containment relationship. Each object to be indexed will be represented by Minimum Bounding Box (MBB) in the index structure except point for which an MBB simply degrades to a point. All indexed objects will eventually be put in leaf nodes. A leaf node contains an array of leaf entries. A leaf entry is a pair (*mbb*, *oid*), where *mbb* is the Minimum Bounding Box (MBB) and *oid* is the object ID. Each internal node is associated with a rectangle, referred to as the directory rectangle (*dr*), which is the minimal bounding box of the rectangle of its child nodes. The structure of R-Tree satisfies the following properties:

- For all nodes in the tree (except for the root), the number of entries is between *m* and *M*, where $0 \leq m \leq M/2$.
- For each entry (*dr*, *node-id*) in a non-leaf node *N*, *dr* is the directory rectangle of a child node of *N*, whose page address is *node-id*.
- For each leaf entry (*mbb*, *oid*), *mbb* is the minimal bounding box of spatial component of the object stored at address oid.
- The root has at least two entries (unless it is a leaf).
- All leaves are at the same level.

R*-Tree [2,7] is a variant of the R-Tree that provides several improvements to the insertion algorithm. Among other things, R* tree reinserts entries upon overflow, rather than splitting.

III. R*-TREES FOR SIMILARITY ANALYSIS

As shown in [1], when R*-tree is used for time series data indexing, each time series of length *n* is mapped to a point in *n*-dimension space. Thus a similarity query problem can be converted to finding those points close to a given point. The whole dataset is indexed through an R*-Tree, and similarity query is then carried out on the R*-Tree. Since R*-Tree indexes spatial objects according to spatial proximity and close points tend to be put in the same leaf node, small amount leaf nodes will be traversed before similar points are found. As a result, fast similarity analysis can be achieved. However, due to the so-called “dimensionality curse,” R*-Tree’s performance degrades rapidly when dimension is larger than 10. Provided that some dataset/database is made up of time series with large length (>>10), and for R*-Tree to work efficiently, a dimensionality reduction technique is necessary. Over the years various dimensionality reduction techniques have been proposed; for example, [1] adopted Discrete Fourier Transform (DFT) as a dimensionality reduction method., and [8] proposed a simple data transformation technique, Piecewise Aggregation Approximation (PAA). However, although principal component analysis (PCA) is a well-known method for dimensionality reduction, its application for time series analysis has not been reported.

We have explored using PCA for dimensionality reduction for time series data and developed a tool for similarity analysis using PCA and other methods. The following are general steps for performing PCA:

Step 1: Construct covariance matrix of column vectors;

Step 2: Calculate eigenvectors;

Step 3: Determine principle components and perform dimensionality reduction.

Compared with DFT and PAA, PCA has following virtues:

- PCA is an orthogonal transformation and can guarantee distance conversation if all eigenvectors are used.
- PCA operates on the whole dataset and can capture the primary features such as data distribution and variation of the dataset. After dimensionality reduction, those primary features can be maintained.

We have developed a similarity analysis tool which is made up of three modules, namely R*-Tree module, PCA module and B-Tree module. In this similarity analysis tool, starting from a dataset and a query and ending with similarity results found, data go through phases as shown in Figure 1. According to this data flow diagram, the similarity analysis tool proceeds as follows. First dimensionality reduction is performed on a given dataset, then the reduced dataset is indexed using R*-Tree. When a query for similarity is issued, the dimensionality of query data is reduced using the same method. The modified query is then performed on R*-Tree and IDs of candidate points are returned. Based on those IDs of candidate time series, their original time series are retrieved through B-Tree. The distance between those original time series and original query time series are calculated. A candidate time series is a final result only if the distance is less than a pre-defined threshold. In case of a reduced distance after transformation, R*-Tree tends to find more points for a given query, a refinement procedure or post-processing is thus needed to remove those induced false points. During post-processing stage, we need access original data. For fast access of those data, we index them through a B-Tree on the identifiers of the data points.

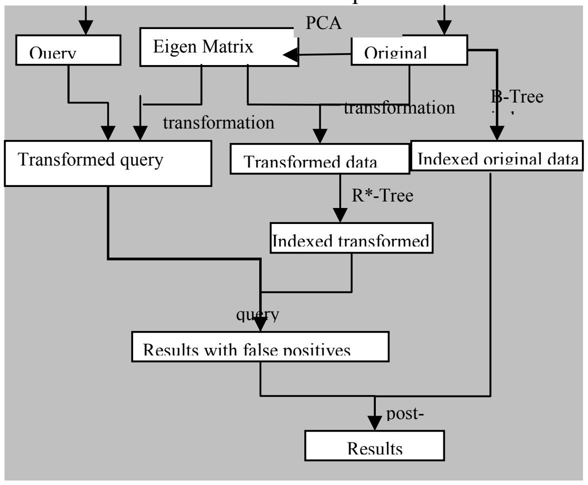


Figure 1 Data flow diagram in the similarity analysis tool when PCA module is used

We have also conducted experiments on various data set, and compared PCA as a dimensionality reduction method for time series data with other methods such as DFT and PAA in terms of efficiency, using the similarity tool described above. The quantities used in the comparison are measured query time, post-processing time and returned false positives. Different datasets were used in this study. For a given query, the performance of each method is evaluated by query time, false positives returned, and post-processing time.

Due to space limitation, only experiments related to a meteorological time series dataset are reported here. First we conducted distance conservation experiment. Results show that PCA can achieve better distance conservation than PAA and DFT, particularly for the reduced dimension larger than 1/3 of the original one. We then conducted experiments on query time efficiency. For this purpose, we constructed two types of query, namely exact query and similar query. The exact query (or Type I query, which is included in our study to compare with similar query) is to retrieve a time series present in the dataset while similar query (or Type II query, which is our focus) is to retrieve similar counterparts in the time series dataset.

To construct a Type II query, we used the 50 queries of Type I as base and fluctuate them with amplitude of 1. The plot of query result is shown in Figure 2, which shows that PCA has slight advantage over PAA, and their query time is not sensitive to the number of dimensions. DFT, however, is 5 to 10 times slower than PCA and PAA at high dimension and 2 to 3 times slower at low dimension. The general trend of query time of DFT is declined with the reduction of dimension. The result can be interpreted by the fact that even for Type II query, the PCA has the best distance conservation and thus candidate points of a query tend to be in the same leaf node of the R* -Tree.

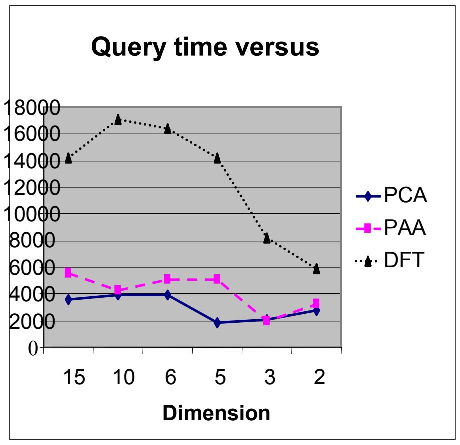


Figure 2 The query time of Type II query for meteorological dataset

IV. R*-TREES FOR CLUSTER ANALYSIS

Similarity querying on R*-trees as discussed above takes advantage of R*-tree's feature of grouping objects together based on spatial proximity. A natural extension of this study would be to push the task of retrieval a step further: to the task of clustering, where the entire dataset is clustered into groups based on similarity so that the commonality of the data in the same cluster can be revealed. However, using R*-tree for cluster analysis should be done cautiously, as researchers already issued the following warning a decade ago [4]:

- IF R*-tree is used for clustering, then all clusters (i.e. the directory rectangles) have a rectangular shape and these rectangles have to be parallel to the axes of the coordinate system. This restriction does not comply with the objective of cluster analysis.
- An R*-Tree is a balanced tree. For all nodes in the tree (except for the root), the number of entries is between m and M, where M is node capacity, m is set to a constant in [0, M/2]. Assuming that each node is a cluster, the number of members in every cluster is then between m and M. In reality, however, no bound is set for the number of cluster members.
- The R*-Tree structure does not allow users to specify the number of clusters, it derives the number of cluster, k, indirectly from n and from the capacity of a page. This k may be inappropriate for a given application and may yield clusterings with a high total distance.

Although there is a mismatch of objectives and behavior between R*-tree structure and cluster analysis, the attractive indexing characteristics of R*-trees still lends itself for great potential of contribution to clustering tasks. Below we discuss issues related to this aspect by examining how to take advantage of R*-tree's indexing feature to get around of the problems mentioned above, and present two improved cluster analysis algorithms by incorporating R*-tree features: KMeans-R (a revised K-means algorithm) and Hierarchy-R (a revised hierarchical clustering algorithm). These algorithms form another part of the software package we have developed.

(1) K-Means extended with R*-Tree: the KMeans-R algorithm

Among the partitioning algorithms, K-Means is a popular and widely studied clustering method for points in Euclidean space. The algorithm of this method is presented below, where *IC* is the collection of initial centroids of *k* clusters. *S* is the collection of centroids of leaf nodes. KMeans(*IC*, *Data*, *k*) refers to the algorithm of clustering *Data* into *k* Clusters through K-means with initial centroids of *IC*, and *k* clusters returned as the result.

K-Means(*IC*, *Data*, *k*)

```
1. $IC = \Phi$
2. Pick randomly from $Data$ $k$ data items, $c_i$ $i=1$ ,…, $k$,
   $IC = IC \cup c_i$
```

```
3. Initialize $k$ clusters, Cluster(i), i=1, …, $k$, with centroids of $c_i$ respectively.
4. For each data item $d_i$, determine its nearest centriods $c_j$, then set $Cluster(j) = Cluster(j) \cup c_j$
5. Set $IC = \Phi$
6. Calculate the new centriods, $nc_i$ , i=1,…, $k$, of each cluster. Set $IC = IC \cup nc_i$
7. Go to 4 until no variation in clustering results.
```

The K-means algorithm is simple and fast, and can handle high-dimensionality relatively well. However, K-means and its variations have a number of limitations. For example, it has difficulty detecting the "natural" clusters; it suffers from the problem of local minima; it requires provision of total number of cluster, *k,* from the user; its behavior is significantly affected by the initial selection of centroids; and it has problem when outliers exist; etc. The issue of initial selection of centroids is partially addressed by a variant (referred to as KMeans-S here) where sampling techniques are used to determine optimized centroids in which sampled points are selected and clustered first to determine initial centroids for clustering the whole dataset (see below).

KMeans-S(*Data*, *k*)

```
1. Sample in Data and generate a subset of Data, S
2. $IC = \Phi$
3. Apply K-Means(IC, S, k) and obtain new IC
4. Apply K-Means(IC, Data, k)
```

Using R*-Trees cannot take care all of the shortcomings of KMeans or KMeans-S algorithms. However, it is reasonable to expect at least R* Trees can be used to assist better sampling. Note that when KMeans-S is used, the sampled points are not guaranteed to represent the whole dataset well. In our view, when data points are indexed through an R*-Tree, if we select one point from each leaf node, these selected data points collectively should represent the data in a way better than random sampling. As a result, the clustering result could be improved. Pushing this observation a step further, we take centroids of data in leaf nodes as a sample instead of using random points, one from each leaf node. Those sampled centroids are then used as initial centroids for clustering. Note that in KMeans-S some sampled points may not represent well the distribution of data points; but in our proposed approach, the sampled points should be relatively good representatives of the dataset, because the collection of leaf nodes is a partition of a dataset and can reflect the distribution of the dataset. Since this proposed approach incorporates R*-Tree into K-Means, it will be referred to as KMeans-R. The algorithm is shown below.

Algorithm KMeans-R(*Data*, *k*)

```
1. $IC = \Phi$, $S = \Phi$
2. Index Data by R*-Tree
3. For each leaf Ni of R*-Tree, obtain the centroid of its data elements, $c_i$ , then set $S = S \cup c_i$
4. Apply K-Means(IC, S, k), obtain new centroids ICN of newly formed clusters, then set IC=ICN
5. Apply K-Means(IC, Data, k) return the clustering results.
```

We have the following observation involving these three clustering algorithms. KMeans-S uses the centroids of sampled dataset as initial centroids instead of randomly picked centroids as done in K-Means. Therefore in KMeans-S, centroids should better capture the spatial distribution of data. KMeans-R further improves initial centroids by using the centroids of leaf nodes. Since the quality of K-Means clustering can be affected by the selection of initial centroids, KMeans-R should perform better than KMeans-S and K-Means. Of course, this observation is only a heuristic; the algorithm developed based on this heuristic is to be confirmed through experimental studies.

(2) Hierarchical clustering extended with R*-Tree: the Hierarchy-R algorithm

Unlike the K-Means method for which a user needs to specify the number of clusters beforehand, hierarchical clustering gives a series of clustering results at each level through merging process.

For comparison purpose, the basic algorithm of the hierarchical clustering is shown below, where *Data* is the input data with size of *n*. Note that *distMatrix* is a matrix of distance between any two clusters. In our implementation, it is initially set to an $m \times m$ lower triangle matrix with elements of 0s in main diagonals where *distMatrix*(i, j) is the element of i^{th} row and j^{th} column of the matrix. We use *merge*(*Cluster*(i), *Cluster*(j)) as the method to merge two clusters. The nearest pair of clusters in a collection of clusters is the pair of clusters which have shortest all pair distance between. The pair can be easily identified as the two clusters corresponding to the row and the column of the minimum element blow the diagonal of *distMatrix*. Note that merging process stops when the desired number of clusters reaches by setting loop times. Clustering results are stored in remaining clusters.

Algorithm Hierarchy(*Data*)

```
1. Assign each points to a cluster, and generate n clusters, say, Cluster(1), Cluster(2), …, Cluster(n).
2. Start off the merging process as follows.
   2.1. Calculate and form initial an $n \times n$ all-pair distance matrix, distMatrix(n, n)
   2.2. Based on distance matrix, identify a pair of nearest clusters, say, Cluster(i) and Cluster(j), then merge them. Set Cluster(i) = merge(Cluster(i), Cluster(j)).
3. Recalculate distance matrix
   3.1. Assume Cluster(s) is the last one in the sequence of clusters, set Cluster(j) = Cluster(s) and then delete Cluster(s).
```

3.2. Recalculate the distance of *Cluster*(i) and *Cluster*(j) to other remaining clusters respectively

3.3. Based on above calculation, form an (s-1)×(s-1) distance matrix *distMatrix*(s-1,s-1)

4. Go to 2.2 until number of clusters is reduced to 1

If clustering starts off from individual points as done in original hierarchical clustering method, the number of start-up clusters will be large and thus clustering will be temporal and spatial expensive (O (m^2), where m is total number of objects). R*-trees can come for help. Although a leaf node in R* tree does not necessarily represent a cluster (as explained in Section 1), it is reasonable to hypothesize that when the node capacity is low for leaf nodes, the points in an R*-Tree's leaf node are likely belonging to same cluster. Therefore rather than start clustering process from individual points, we can first index those points and then cluster those minimal bounding boxes of leaf nodes. The corresponding algorithm is presented below. We would like to point out that the value of *m* cannot be big, otherwise the points in a leaf node could belong to two or more clusters. The value of *m* cannot be too small either, otherwise Hierarchy-R degrades to hierarchical clustering (m=1).

Algorithm Hierarchy-R(*Data*)

```
1. Index $Data$ through R*-Tree and generate $m$ leaf nodes.
2. Assign the points in each leaf node to a cluster, and generate $m$ initial clusters,
    $Cluster(1), Cluster(2), \ldots, Cluster(m)$.
3. Merge clusters as done in hierarchical clustering
```

V. Experimental evaluation

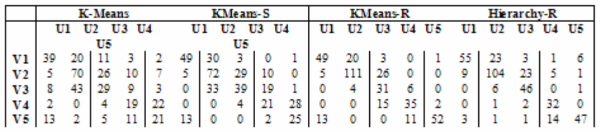
We have conducted experiments on time series data analysis using the proposed algorithms. Due to space limitation, here we only report one of the experiments on yeast cell cycle dataset [9], which shows the fluctuation of expression levels of approximately 6000 genes over two cell cycles (17 time points). It has been identified 420 genes which peak at different time points and categorized them into five phases of cell cycle. Out of the 420 genes they classified, 384 genes were classified into only one phase (some genes peak at more than one phase of cell cycle). For this dataset, we have prior knowledge that 5 classes exist (see Table 1). The clustering results in Table 1 indicate that none of four clustering methods produces a result same as our prior knowledge. The original clusters (prior knowledge) is split in our newly formed clusters. In KMeans-R and Hierarchy-R, original clusters are less spread which can be seen from the dominance of the numbers in the diagonal. On the other hand, in K-Means, original clusters spread out much in newly formed clusters. KMeans-S is a little better than K-Means, but worse than KMeans-R and Hierarchy-R.

To evaluate the clustering results accurately, we resort to the Rand Index (RI), Adjusted Rand Index (ARI) and information gain (IG). For the clustering results in Table 2, the calculated RI and ARI are shown in Table 3.

For the clustering output, one can see the calculated ARIs are much less than RIs. The ratio of RI between KMeans-R and KMeans is $\frac{0.816}{0.711} = 1.148$, while the ratio of ARI is $\frac{0.486}{0.173} = 2.809$ .This means ARI is more sensitive to clustering method than RI, and thus more suitable for clustering quality evaluation.

Clusters	**U1**	**U2**	**U3**	**U4**	**U5**
Number	67	135	75	52	55

Table 1. Known clusters and the number of elements in the yeast cell cycle dataset



	K-Means					KMeans-S					KMeans-R					Hierarchy-R				
	U1	U2	U3	U4	U5	U1	U2	U3	U4	U5	U1	U2	U3	U4	U5	U1	U2	U3	U4	U5
V1	39	20	11	3	2	49	30	3	0	1	49	20	3	0	1	55	23	3	1	6
V2	5	70	26	10	7	5	72	29	10	0	5	111	26	0	0	9	104	23	5	1
V3	8	43	29	9	3	0	33	39	19	1	0	4	31	6	0	0	6	46	0	1
V4	2	0	4	19	22	0	0	4	21	28	0	0	15	35	2	0	1	2	32	0
V5	13	2	5	11	21	13	0	0	2	25	13	0	0	11	52	3	1	1	14	47

Table 2. The generated contingency table

	K-Means	K-Means-S	K-Means-R	Hierarchy-R
RI	0.711	0.739	0.816	0.802
ARI	0.173	0.255	0.486	0.453

Table 3. Calculated RI and ARI based on Table 2

Both RI and ARI are the highest in KMeans-R and lowest in K-Means. The RI and ARI in Hierarchy-R are a little smaller than those in KMeans-R, but they are comparable. The two indexes in KMeans-S are slightly greater than in KMeans. Therefore in terms of clustering quality, KMeans-R and Hierarchy-R outperform both KMeans and KMeans-S. The result of using IG for evaluation is similar.

VI. Conclusion

Data mining is concerned with important aspects related to both database techniques and AI/machine learning mechanisms, and provides an excellent opportunity for exploring the interesting relationship between retrieval and inference.

This paper reports and important step toward this direction of study from a data mining-related perspective. R*-tree indexing techniques have been used to deal with similarity retrieval and clustering. As our work shows, extending similarity retrieval to clustering is not a straightforward process. The general lesson we learned from this study is that since this relationship is a complex issue, it should be studied on a case-by-case base; for example, in our study, we have exploited features of R*-trees. However, this is not to say that there is no commonality on different cases of relationship between retrieval and reasoning. Rather, finding such a commonality is a challenging task which deserves much effort.

Future trends in regard to the research work related to this paper can be divided into two categories. On the one hand, as an attractive database primitive, R* trees will continue to play an important role in similarity retrieval and cluster analysis. On the other hand, R* trees are only one of many spatial data structures. Many other spatial data structures, such as Octrees and KD-trees, have been proposed. Spatial data structures typically address the important issue of dealing with high dimensionality, a common concern behind index structures such as X-tree, TV-tree, SR-tree, among others. However, studies of similarity retrieval and cluster analysis regarding to these data structures are less matured than the case of R*-trees, and more research work is still ahead.

REFERENCES

[1] R. Agrawal, R., C. Faloutsos, and A. Swami, Efficient similarity search in sequence databases, *Proc. of the 4th Conference on Foundations of Data Organization and Algorithms*, 1993.

[2] N. Beckmann, H. P. Kriegel, R. Schneider and B. Seeger, The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles, *Proc. SIGMOD Conference,* 322-331, 1990.

[3] Z. Chen, Generating suggestions through document structure mapping, *Decision Support Systems,* 16(4), 297-314, 1996.

[4] M. Ester, H. P. Kriegel, and X. Xu, Knowledge discovery in large spatial databases: Focusing techniques for efficient class identification, *Proceedings of 4th International Symposium on Large Spatial Databases (SSD'95)*, Portland, ME, LNCS, Springer, pp 67-82, 1995.

[5] A. M. Frisch and J. F. Allen, Knowledge retrieval as limited inference, *Proceedings of the 6th Conference on Automated Deduction* (Lecture Notes in Computer Science), Loveland, D. (ed.), 274-291, 1982.

[6] A. Guttman, R-trees: A Dynamic Index Structure for Spatial Searching, *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pp. 47-54, 1984.

[7] V. Gaede and O. Günther**,** Multidimensional Access Methods*, ACM Computing Surveys*, 30(2):170-231, June 1998.

[8] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra, Dimensionality reduction for fast similarity search in large time series databases, *Knowledge and information Systems 3(3):263-286*, 2000.

[9] K. Y. Yeung and W. L. Ruzzo, Principle component analysis for clustering gene expression data, *Bioinformatics*, 17(9):763-774, 2001.

[10] Foundation of Data Mining Workshop Call for Papers, http://biomig.csie.cgu.edu.tw/meeting/2004.07.25.htm