Non-Hierarchical Clustering with Rival Penalized Competitive Learning for Information Retrieval*

Irwin King and Tak-Kan Lau

Department of Computer Science & Engineering
The Chinese University of Hong Kong
Shatin, New Territories, Hong Kong
king@cse.cuhk.edu.hk, http://www.cse.cuhk.edu.hk/~king

Abstract. In large content-based image database applications, efficient information retrieval depends heavily on good indexing structures of the extracted features. While indexing techniques for text retrieval are well understood, efficient and robust indexing methodology for image retrieval is still in its infancy. In this paper, we present a non-hierarchical clustering scheme for index generation using the *Rival Penalized Competitive Learning* (RPCL) algorithm. RPCL is a stochastic heuristic clustering method which provides good cluster center approximation and is computationally efficient. Using synthetic data as well as real data, we demonstrate the *recall* and *precision* performance measurement of nearest-neighbor feature retrieval based on the indexing structure generated by RPCL.

1 Introduction

One of the key issues in information retrieval of data in large and voluminous database is the design and implementation of an efficient and effective indexing structure for the data objects in the database.[1] Without a properly designed indexing structure, the retrieval of information may be reduced to a linear exhaustive search. On the other hand, a good indexing structure will make the retrieval accurate and computationally efficient.

The following paragraphs outline the basic feature vector model for nearest-neighbor search. In our framework, we let $DB = \{I_i\}_{i=1}^{N}$ be a set of image objects. Without loss of generality, a feature extraction function $f : I \times \boldsymbol{\theta} \to \mathcal{R}^d$ extracts from an image I, with a set of parameters $\boldsymbol{\theta} = \{\theta_1, \theta_2, \cdots, \theta_m\}$, a real-valued d-dimensional vector. Hence, we may view the extracted feature vector as a point in a d-dimensional vector space. Furthermore, we may use a random variable X to denote the feature vector extracted from the image set DB and

* This work is supported in part by the RGC Grant #CUHK4176/97E. Portions of this manuscript have been presented in [8].

[1] In this paper, data objects and feature vectors are interchangeable unless stated otherwise.

$\boldsymbol{x}_i, i = 1, 2, \cdots, N$ to denote the instance of the feature vector extracted from DB.

Once the feature vectors have been obtained. Similar feature (content-based) search can be performed as a nearest-neighbor search by using a distance function. A typical distance function D is defined as $D : F \times F \rightarrow \mathcal{R}$ satisfying: (1) $D(x, y) \geq 0$, (2) $D(x, y) = D(y, x)$, (3) $D(x, y) = 0$ iff $x = y$, and (4) $D(x, y) + D(y, z) \geq D(x, z)$ where x, y, and $z \in F$ and F is a feature vector set. Here, L_2-norm (Euclidean distance) is one of the common distance functions and it is defined as: D as: $D(x, y) = ||x - y|| = (\sum_{i=1}^{d}(x_i - y_i)^2)^{1/2}$.

There are two typical types of query involved in image databases: (1) Range Search and (2) k Nearest-Neighbor Search. Given a set of N features $X = \{x_i\}_{i=1}^{N}$, a *Range Query*, $\hat{x}$, returns the set, P, of features as $P = \{x | x \in X \text{ and } 0 \leq D(x, \hat{x}) \leq \epsilon\}$, where ϵ is a pre-defined positive real number and D is a distance function. As in the k *Nearest-Neighbor Search* case, given a set of N features $X = \{x_i\}_{i=1}^{N}$, a k *Nearest-Neighbor Query*, $\hat{x}$, returns the set $P \subseteq X$ satisfying: (1) $|P| = k$ for $1 \leq k \leq N$ and (2) $D(\hat{x}, x) \leq D(\hat{x}, y)$ for $y \in X - P$ where D is a distance function. In this paper, we will focus on the latter type of query.

Once the features have been extracted and the query model has been defined. The crucial link between the features and user query is the indexing structure (organization). A well-organized indexing structure of the underlying feature vectors support an efficient and effective retrieval of user queries.

Recently, researchers have developed many new indexing methods for content-based retrieval in multimedia databases. For example, rectangle-based indexing as in R-Tree [6], R+-Tree [11], R*-Tree [1], SR-Tree [7]. Partition-based Indexing as in Quad-tree [5], k-d Tree [2], VP-Tree [4,13], and MVP-tree [3].

However, one major problem of these indexing techniques has been that these methods fail to utilize the underlying data distribution to their advantage in their indexing structure. This results in what is known as the *boundary query problem* where the retrieval *Precision* will degrade when a query is near the boundary of a partition in the indexing structure due to the systematic yet unfavorable partitioning of some indexing techniques. To overcome this, we will present a non-hierarchical clustering algorithm based on Rival Penalized Competitive Learning (RPCL) heuristic and demonstrate its effectiveness in generating indexing structure for large image databases.

In Section 2 we will present more details on RPCL and the associated non-hierarchical indexing. Experimental results of the proposed method are presented in Section 3. We will discuss some issues associated with the proposed method and conclude in Section 4.

2 Non-Hierarchical Clustering with RPCL

There are two main goals in our proposed solution: (1) find a quick way to partition the input feature set into partitions and (2) impose an indexing structure over these partitions so that the nearest-neighbor information retrieval can be made effectively.

Rival Penalized Competitive Learning (RPCL) clustering [12] can be regarded as an unsupervised extension of Kohonen's supervised learning vector quantization algorithm LVQ2 [9]. It can also be regarded as a variation to the more typical Competitive Learning (CL) algorithms [10]. RPCL is a stochastic clustering algorithm that is able to perform adaptive clustering efficiently and quickly leading to an approximation of clusters that are statistically adequate.

The proposed solution is to use RPCL to find hierarchical clusters such that the indexing structure can be created based on a natural partition of the feature vector distribution. Although this may not result in a balanced tree structure, this indexing structure will be able to answer nearest-neighbor queries more effectively.

The main advantages of RPCL are: (1) the heuristic is computationally efficient, (3) it is no worse than other methods when high dimensional features, and (3) RPCL can be implemented in a distributed environment achieving even greater speed-up in generating indexing structure of feature vectors.

2.1 The RPCL Algorithm

Step 0: **Initialization** Randomly pick $\boldsymbol{c}_i, i = 1, 2, \cdots, k$ as the initial cluster centers.

Step 1: **Winner-Take-All Rule** Randomly take a feature vector $\boldsymbol{x}$ from the feature sample set X, and for $i = 1, 2, \cdot, k$, we let

$$u_i = \begin{cases} 1, & \text{if } i = w \text{ such that} \\ & \quad \gamma_w \|\boldsymbol{x} - \boldsymbol{c}_w\|^2 = \min_j \gamma_j \|\boldsymbol{x} - \boldsymbol{c}_j\|^2, \\ -1, & \text{if } i = r \text{ such that} \\ & \quad \gamma_r \|\boldsymbol{x} - \boldsymbol{c}_r\|^2 = \min_j \gamma_j \|\boldsymbol{x} - \boldsymbol{c}_j\|^2, \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

where w is the winner index, r is the second winner (rival) index, $\gamma_j = n_j / \sum_{i=1}^{k} n_i$ and n_i is the cumulative number of the occurrences of $u_i = 1$. This term is added to ensure that every cluster center will eventually become the winner during the updating process.

Step 2: **Updating Cluster Centers** Update the cluster center vector $\boldsymbol{c}_i$ by

$$\Delta \boldsymbol{c}_i = \begin{cases} \alpha_w (\boldsymbol{x} - \boldsymbol{c}_i), & \text{if } u_i = 1, \\ -\alpha_r (\boldsymbol{x} - \boldsymbol{c}_i), & \text{if } u_i = -1, \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

where $0 \leq \alpha_w, \alpha_r \leq 1$ are the learning rates for the winner and rival unit, respectively.

Assuming there are k cluster centers, the basic idea behind RPCL is that in each iteration, the cluster center for the winner's unit is accentuated where as the weight for the second winner, or the rival, is attenuated. The remaining $k-2$ centers are unaffected. The winner is defined as the cluster center that is closest to the randomly selected feature vector. Instead of k can be of any value, in our application we use the special version of the RPCL clustering algorithm when $k = 2^i, i = 1, 2, 3, \cdots$ so that a systematic index tree can be formed.

Let k, $\boldsymbol{c}_w$ and $\boldsymbol{c}_r$ to denote the number of clusters, cluster center points for the winner and rival clusters respectively. The algorithm is illustrated in the boxed region above.

Step 1 and 2 are iterated until one of the following criteria is satisfied: (1) the iteration converges, (2) $\alpha_w \to \epsilon$ for a time decaying learning rate of α_w with a pre-specified threshold of ϵ, or (3) the number of iterations reaches a pre-specified value.

2.2 Non-Hierarchical RPCL Indexing

There are two ways to perform the clustering. One is the non-hierarchical approach and the other is the hierarchical approach. In this paper, we will only focus on the former approach.

The non-hierarchical indexing approach considers the whole feature vector space each time for clustering by RPCL. We use an example here to explain its basic idea. Given a set of feature vectors, our method clusters the set into 2 clusters at the first time (see Fig. 1 (a)). If four partitions are required at the next time, our method will consider the whole space again and clusters the set into 4 clusters (see Fig. 1 (b)). In the non-hierarchical clustering, clusters at a subsequent stage may not be nested into clusters from a previous stage, but this method ensures to obtain the correct natural clusters at each stage.

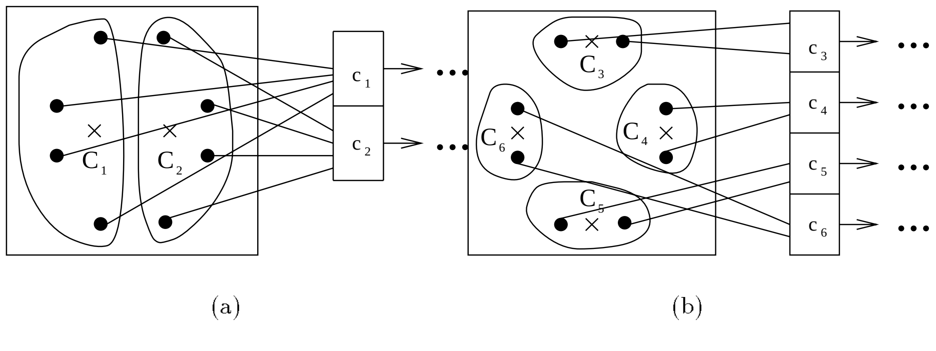


Fig. 1. (a) Two and (b) Four cluster partitions generated by the non-hierarchical approach. The dots are the database objects whereas the crosses are the centers. An inverted file (the right one) is used for indexing.

3 Experimental Results

We conducted four different sets of experiments for the four methods: RPCL, k-means, Competitive Learning (CL) clustering, and VP-tree to test their accuracy and efficiency for indexing and retrieval. All of the experiments were conducted on an Ultra Sparc 1 machine running Matlab V4.2c. Here we assume that a cluster of feature vectors is often retrieved as the result of a query for nearest-neighbor search. An indexing method which can locate natural clusters from the input feature vector set accurately and quickly will make nearest-neighbor search more effective and efficient. Therefore, in these experiments, we restrict to retrieve the first visited feature vector cluster or leaf node as the result of a nearest-neighbor query so that, based on the result, we can show that how accurate and efficient the tested methods are to locate natural clusters for indexing and retrieval.

We used two performance measurements: *Recall* and *Precision* in the experiments to measure the accuracy of the tested methods. Given a set of user-specified target database objects, *Recall* and *Precision* performance measurements are defined as:

$$Recall = \frac{\text{Number of target database objects retrieved}}{\text{Number of target database objects}} , \tag{3}$$

$$Precision = \frac{\text{Number of target database objects retrieved}}{\text{Number of database objects retrieved}} , \tag{4}$$

where $0 \leq Recall, Precision \leq 1$. *Recall* shows the ratio of target database objects are actually retrieved out of all the expected target database objects whereas *Precision* indicates the ratio of target database objects in the retrieved set. For example, there are 10 database objects and 4 of them are pre-specified as target database objects. For a query, 5 database objects are retrieved and 3 of them are target database objects. In this case, *Recall* is 0.75 and *Precision* is 0.6. Typically, the higher the *Recall* and *Precision*, the more accurate the method for retrieval. By using *Recall* and *Precision*, we can calculate the accuracy for each of the generated clusters based on the information of its corresponding natural cluster. If we do not use them for accuracy, we can only evaluate the accuracy by using a small set of queries. Therefore, we use *Recall* and *Precision* to evaluate the accuracy of these methods in the experiments.

There are three types of data being used in our experiments. They are (1) synthetic data with Gaussian distribution, (2) synthetic data with uniform distribution, and (3) real data. We now give more details to each of the data set used.

1 Synthetic Data in Gaussian Distribution:
We test our method with synthetic data sets in Gaussian distribution. It is because many distributions can be approximated by using Gaussian distribution. Let $\mu = (\mu_1, \mu_2, \ldots, \mu_n)$ and $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$, we generated the input distribution of the feature vectors from the mixture of n

Gaussian distributions $N(\mu, \sigma^2)$ with the generating function defined as $g(x) = 1/(\sigma\sqrt{2\pi})\exp[-[(x-\mu)^2/2\sigma^2]]$, $-\infty < x < \infty$. In our experiments, we used a constant 0.05 for σ. Moreover, we let $n = 2, 4, 8, 16$, and 32. Finally, for each input distribution, different numbers of cluster partitions are generated for the input feature vectors for testing.

2 Synthetic Data in Uniform Distribution:

We also use synthetic data sets in uniform distribution with no clear bias toward any cluster centers.

3 Real Data:

Apart from the two synthetic data sets, we also use real data in the experiments to test our method in a real world situation. For our experiments, the real data features are the feature vectors extracted from real images. Basically, we firstly find some real images from different kinds of catalogs. By considering the global color information of each image, we calculate an 8-bucket color histogram form the image and transform it into a feature vector. All of the output feature vectors form the real data set for testing.

We have devised four experiments to test out various aspects of the proposed indexing scheme. The first experiment tests the *Recall*, *Precision*, and the pre-processing speed performance of RPCL, CL, k-means, and VP-tree. Once the favorable results from RPCL has been established. The second experiment focuses on RPCL's performance under different sizes of input data set. The third experiment measures the RPCL performance with different numbers of dimensions. The last experiment compares the RPCL indexing scheme against actual nearest-neighbor results.

3.1 Experiment 1: Test for *Recall* and *Precision* Performance

In the first experiment, we evaluate the accuracy and efficiency of the four tested methods: RPCL, CL, k-means, and VP-tree to build indexing structures for retrieval. We measure the *Recall* and *Precision* performance of these methods for accuracy. Moreover, we also kept the time used for pre-processing which includes clustering and indexing of the feature vectors for efficiency. The main aim of this experiment is to find out which tested method has the best overall performance for locating natural clusters for indexing and retrieval.

We used three different kinds of data sets in this experiment: (1) synthetic data in Gaussian distribution, (2) synthetic data in uniform distribution, and (3) real data. Each of the data sets consists of 2048 8-dimensional feature vectors. In addition, we used 8-D feature vectors in our experiment. Moreover, for each input data set, different numbers of cluster partitions were generated for the input feature vectors by the four tested methods respectively. We conducted 20 trails with different initial starting points of the centers of the to-be generated cluster partitions for these methods to calculate their average *Recall* and *Precision* performance measurement and the average time used for building indexing structure.

The results are presented as follows. For the data sets in Gaussian distribution with different mixture groups, Tables 1 and 2 show the *Recall* and *Precision* results. For the data set in uniform distribution and the real data set, we can simply use Figures 2 and 3 to present their results respectively. Moreover, Tables 3, 4, and 5 show the main observations of this experiment.

	No. of Generated Clusters (RPCL, CL, k-means, VP-tree)				
$\#MG$	**2**	**4**	**8**	**16**	**32**
2	1.0, 1.0, 1.0, 1.0	.51, .45, .66, .50	.27, .25, .57, .25	.15, .14, .53, .13	.22, .09, .51, .06
4	1.0, 1.0, 1.0, 1.0	1.0, 1.0, 1.0, 1.0	.52, .58, .80, .50	.39, .43, .77, .25	.53, .31, .76, .13
8	1.0, .91, 1.0, .87	1.0, 1.0, 1.0, .71	1.0, 1.0, .94, .56	.75, 1.0, .89, .31	.73, .56, .88, .17
16	.96, .95, 1.0, .90	1.0, .99, 1.0, .86	1.0, 1.0, 1.0, .76	.99, .98, .96, .65	.93, .83, .94, .41
32	.96, .98, .99, .93	.98, .96, 1.0, .86	.97, .87, .99, .80	.98, .87, 1.0, .69	.98, .87, .94, .63

Table 1. *Recall* table for the data sets in Gaussian distributions in Experiment 1. $\#MG$ is the number of Gaussian mixture groups.

	No. of Generated Clusters (RPCL, CL, k-means, VP-tree)				
$\#MG$	**2**	**4**	**8**	**16**	**32**
2	1.0, 1.0, 1.0, 1.0	1.0, 1.0, 1.0, 1.0	1.0, 1.0, 1.0, 1.0	1.0, 1.0, 1.0, 1.0	1.0, 1.0, 1.0, 1.0
4	.50, .50, .50, .50	1.0, 1.0, 1.0, 1.0	1.0, 1.0, 1.0, 1.0	1.0, 1.0, 1.0, 1.0	1.0, 1.0, 1.0, 1.0
8	.25, .23, .25, .15	.46, .50, .50, .20	1.0, 1.0, .93, .36	1.0, 1.0, .97, .63	1.0, 1.0, 1.0, .79
16	.11, .12, .13, .10	.26, .24, .27, .16	.54, .54, .61, .21	.97, .93, .88, .39	.98, .85, .97, .68
32	.06, .05, .06, .05	.11, .11, .14, .08	.25, .18, .26, .13	.51, .39, .56, .21	.94, .71, .87, .41

Table 2. *Precision* table for the data sets in Gaussian distributions in Experiment 1. $\#MG$ is the number of Gaussian mixture groups.

Measures	**RPCL**	**CL**	**k-means**	**VP-tree**
Recall	high	middle	highest	lowest
Precision	high	middle	highest	lowest
Preprocessing Speed	highest	high	lowest	middle

Table 3. Comparison of the average performance of the four methods for indexing and retrieval with data sets in Gaussian distributions.

3.2 Experiment 2: Test for Different Sizes of Input Data Sets

In this experiment, we test the accuracy and efficiency of RPCL for indexing and retrieval with different sizes of input feature vector sets. We measure the *Recall*

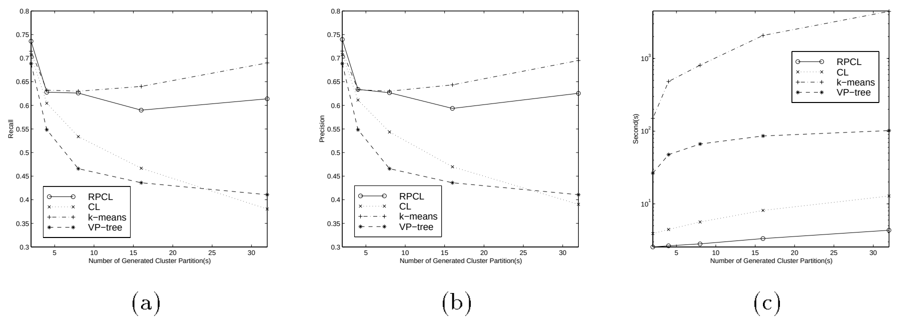


Fig. 2. Results for the uniform data set in Experiment 1. (a) The *Recall* results. (b) The *Precision* results. (c) The pre-processing time.

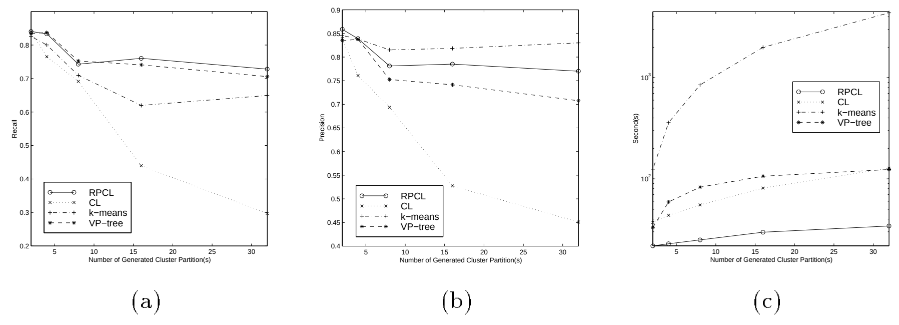


Fig. 3. Results for the real data set in Experiment 1. (a) The *Recall* results. (b) The *Precision* results. (c) The pre-processing time.

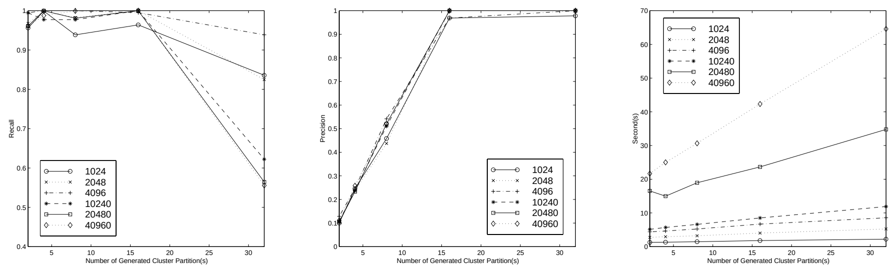
Measures	**RPCL**	**CL**	**k-means**	**VP-tree**
Recall	high	low	highest	low
Precision	high	low	highest	low
Preprocessing Speed	highest	high	lowest	middle

Table 4. Comparison of the average performance of the four methods for indexing and retrieval with the uniform data set.

Measures	**RPCL**	**CL**	**k-means**	**VP-tree**
Recall	high	low	high	high
Precision	high	low	high	high
Preprocessing Speed	highest	middle	lowest	middle

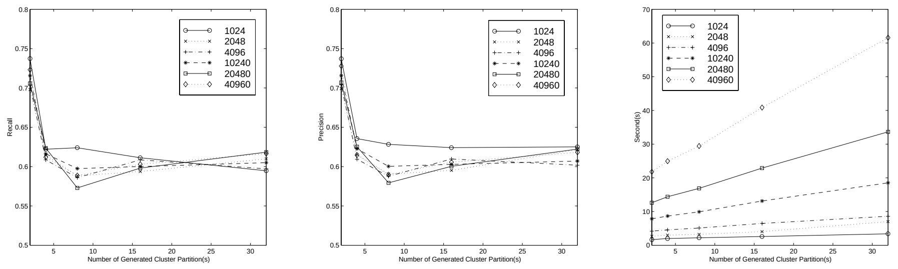
Table 5. Comparison of the average performance of the four methods for indexing and retrieval with a given real data set.

and *Precision* performance of our method for accuracy and record the time used for pre-processing for efficiency. We use two different kinds of data sets in this experiment: (1) synthetic data in Gaussian distribution and (2) synthetic data in uniform distribution. The data sets are 8-dimensional feature vector sets with sizes varying from 1024 to 40960. For each input data set, different numbers of cluster partitions are generated for the experiment. We conducted 20 trails with different initial starting points of the centers of the to-be generated cluster partitions for RPCL to calculate its average *Recall* and *Precision* Performance and the average time used for building indexing structure.



(a) (b) (c)

Fig. 4. Results for the data sets in Gaussian distribution with 16 mixture groups in Experiment 2. (a) The *Recall* results. (b) The *Precision* results. (c) The pre-processing time.



(a) (b) (c)

Fig. 5. Results for the uniform data sets in Experiment 2. (a) The *Recall* results. (b) The *Precision* results. (c) The pre-processing time.

	No. of Generated Clusters				
$\#MG$	**2**	**4**	**8**	**16**	**32**
2	1.0	.47	.28	.23	.17
	1.0	.47	.28	.12	.06
	1.0	.51	.26	.11	.11
	1.0	.47	.30	.12	.06
	1.0	.47	.29	.14	.06
	1.0	.53	.29	.14	.06
4	1.0	1.0	.54	.67	.57
	1.0	1.0	.50	.37	.43
	1.0	1.0	.64	.25	.14
	1.0	1.0	.62	.25	.15
	1.0	1.0	.64	.25	.12
	1.0	1.0	.65	.26	.13
8	1.0	1.0	1.0	1.0	.79
	.99	1.0	1.0	.78	.64
	1.0	.92	1.0	.53	.56
	1.0	1.0	1.0	.57	.31
	1.0	1.0	1.0	.62	.27
	1.0	1.0	1.0	.57	.27
16	.96	1.0	.94	.96	.84
	.97	1.0	.98	1.0	.82
	1.0	1.0	1.0	1.0	.94
	.99	.98	.98	1.0	.62
	.96	1.0	.98	1.0	.56
	.96	.99	1.0	1.0	.56
32	.97	.96	.97	.95	.95
	.98	.95	.97	.98	1.0
	.98	.90	.96	.96	1.0
	.97	.98	.98	.99	1.0
	.99	1.0	.97	1.0	.99
	.99	.98	1.0	.99	1.0

(a)

	No. of Generated Clusters				
$\#MG$	**2**	**4**	**8**	**16**	**32**
2	1.0	1.0	1.0	1.0	1.0
	1.0	1.0	1.0	1.0	1.0
	1.0	1.0	1.0	1.0	1.0
	1.0	1.0	1.0	1.0	1.0
	1.0	1.0	1.0	1.0	1.0
	1.0	1.0	1.0	1.0	1.0
4	.50	1.0	1.0	1.0	1.0
	.50	1.0	1.0	1.0	1.0
	.50	1.0	1.0	1.0	1.0
	.50	1.0	1.0	1.0	1.0
	.50	1.0	1.0	1.0	1.0
	.50	1.0	1.0	1.0	1.0
8	.27	.58	1.0	1.0	1.0
	.23	.58	1.0	1.0	1.0
	.27	.46	1.0	1.0	1.0
	.27	.50	1.0	.97	1.0
	.25	.50	1.0	1.0	1.0
	.25	.58	1.0	1.0	1.0
16	.10	.25	.46	.97	.98
	.11	.24	.44	1.0	1.0
	.13	.25	.54	.97	1.0
	.11	.24	.51	1.0	1.0
	.11	.23	.52	1.0	1.0
	.11	.26	.52	1.0	1.0
32	.06	.10	.20	.44	.83
	.06	.09	.23	.50	.97
	.05	.10	.19	.46	.97
	.05	.11	.23	.42	.97
	.05	.11	.19	.46	.97
	.05	.10	.20	.47	.98

(b)

Table 6. Results for the data sets in Gaussian distributions in Experiment 2. (a) The *Recall* table. (b) The *Precision* table. Each entry of the tables is a column of 6 values for 6 different sizes of the data sets: 1024, 2048, 4096, 10240, 20480, and 40960. $\#MG$ is the number of Gaussian mixture groups.

Size of	# Generated Clusters				
Data Set	**2**	**4**	**8**	**16**	**32**
1024	.73	.62	.62	.61	.59
2048	.70	.61	.59	.59	.61
4096	.70	.61	.59	.61	.60
10240	.72	.62	.60	.60	.61
20480	.71	.62	.57	.60	.62
40960	.72	.61	.59	.60	.62

(a)

Size of	# Generated Clusters				
Data Set	**2**	**4**	**8**	**16**	**32**
1024	.74	.64	.63	.62	.63
2048	.70	.61	.59	.60	.62
4096	.70	.61	.59	.61	.60
10240	.72	.62	.60	.60	.61
20480	.71	.63	.58	.60	.62
40960	.73	.61	.59	.61	.62

(b)

Table 7. Results for the data sets in uniform distribution in Experiment 2. (a) The *Recall* table. (b) The *Precision* table.

	No. of Generated Clusters (DIM = 4, 8, 16, 32)				
$\#MG$	**2**	**4**	**8**	**16**	**32**
2	1.0, 1.0, 1.0, 1.0	.51, .49, .53, .55	.25, .25, .26, .28	.13, .12, .14, .11	.07, .07, .06, .06
4	1.0, 1.0, 1.0, 1.0	1.0, 1.0, 1.0, 1.0	.61, .48, .62, .62	.27, .27, .27, .26	.14, .14, .13, .12
8	1.0, 1.0, 1.0, 1.0	1.0, 1.0, 1.0, 1.0	1.0, 1.0, 1.0, 1.0	.50, .49, .54, .64	.32, .24, .27, .28
16	1.0, .96, .98, 1.0	1.0, 1.0, 1.0, 1.0	.98, 1.0, 1.0, 1.0	.98, 1.0, 1.0, 1.0	.58, .57, .57, .57
32	.94, .99, .99, .98	.93, .97, .99, .99	.93, .95, 1.0, .98	.96, .97, 1.0, 1.0	.97, 1.0, 1.0, 1.0

Table 8. The *Recall* table for the data sets in Gaussian distributions in Experiment 3.

3.3 Experiment 3: Test for Different Numbers of Dimensions

Apart from different sizes of data sets, we also test the performance of RPCL for indexing with feature vectors having different numbers of dimensions in terms of *Recall* and *Precision* for accuracy and the pre-processing time for efficiency. Two different kinds of data sets are used in this experiment: (1) synthetic data in Gaussian distribution and (2) synthetic data in uniform distribution. All the data sets are fixed to have 10240 feature vectors with different numbers of dimensions such as 4, 8, 16, and 32. We fixed the size of each data set to 10240 as it is not too large or too small for testing and we do not use data more than 32-D because it is not so efficient for our method to work with such high dimensional data. Moreover, for each input data set, different numbers of cluster partitions are generated for the experiment. We conducted 20 trails with different initial starting points of the centers of the to-be generated cluster partitions for RPCL to calculate its average *Recall* and *Precision* Performance and the average time used for building indexing structure.

By increasing the number of dimensions, the experimental results show that the accuracy is not affected for the data sets in Gaussian distributions, but it may be lowered for the data sets in uniform distribution. The relatively lower *Recall* and *Precision* results found for uniform data because there are no explicit

	No. of Generated Clusters (DIM = 4, 8, 16, 32)				
$\#MG$	**2**	**4**	**8**	**16**	**32**
2	1.0, 1.0, 1.0, 1.0	1.0, 1.0, 1.0, 1.0	1.0, 1.0, 1.0, 1.0	1.0, 1.0, 1.0, 1.0	1.0, 1.0, 1.0, 1.0
4	.50, .50, .50, .50	1.0, 1.0, 1.0, 1.0	1.0, 1.0, 1.0, 1.0	1.0, 1.0, 1.0, 1.0	1.0, 1.0, 1.0, 1.0
8	.23, .27, .25, .57	.67, .58, .54, .80	1.0, 1.0, 1.0, 1.0	1.0, 1.0, 1.0, 1.0	1.0, 1.0, 1.0, 1.0
16	.11, .09, .12, .21	.23, .20, .22, .25	.48, .48, .50, .54	.80, 1.0, 1.0, 1.0	.92, 1.0, 1.0, 1.0
32	.05, .05, .07, .06	.08, .11, .11, .11	.15, .20, .25, .32	.39, .51, .53, .54	.72, 1.0, 1.0, 1.0

Table 9. The *Precision* table for the data sets in Gaussian distributions in Experiment 3.

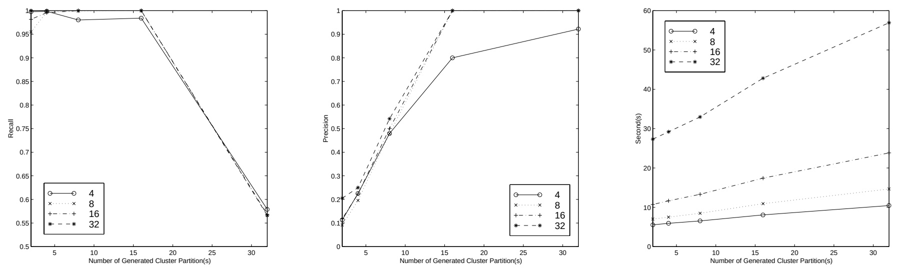
	# Generated Clusters				
DIM	**2**	**4**	**8**	**16**	**32**
4	.80	.74	.78	.83	.78
8	.71	.60	.59	.58	.62
16	.69	.55	.47	.45	.45
32	.65	.49	.41	.36	.34

(a)

	# Generated Clusters				
DIM	**2**	**4**	**8**	**16**	**32**
4	.80	.74	.78	.83	.78
8	.72	.60	.59	.58	.63
16	.69	.55	.47	.45	.45
32	.65	.49	.41	.36	.34

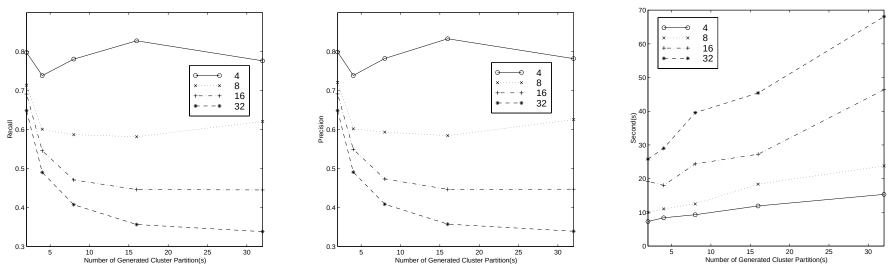
(b)

Table 10. Results for the uniform data sets in Experiment 3. (a) The *Recall* table. (b) The *Precision* table.



(a) (b) (c)

Fig. 6. Results for the data sets in Gaussian distribution with 16 mixture groups in Experiment 3. (a) The *Recall* results. (b) The *Precision* results. (c) The pre-processing time.



(a) (b) (c)

Fig. 7. Results for the uniform data sets in Experiment 3. (a) The *Recall* results. (b) The *Precision* results. (c) The pre-processing time.

natural clusters for RPCL to locate. Therefore, we can conclude that our method is more suitable for data sets with distributions similar to Gaussian distribution.

3.4 Experiment 4: Compare with Actual Nearest-neighbor Results

In this experiment, we compare the results given by our method with the actual nearest-neighbor results in order to check the actual accuracy of our method. In the first three sets of experiments, we mainly evaluate the *Recall* and *Precision* performance of the tested methods. We want to find out the (accuracy) percentage of the database objects retrieved by our method can also be found in the actual nearest-neighbor results in the experiment for accuracy.

We use three different kinds of data sets in this experiment: (1) synthetic data in Gaussian distribution, (2) synthetic data in uniform distribution, and (3) real data. Each of the data sets contains 8-dimensional 10240 feature vectors. Moreover, for each input data set, different numbers of cluster partitions are generated for the experiment. We conducted 20 trails with different initial starting points of the centers of the to-be generated cluster partitions for RPCL to find out the results of the given queries. The results of this experiment are presented by Tables 11, 12, and 13.

There are several observations for the accuracy percentages of the three different kinds of data sets similar to those in Experiment 1. For data sets in Gaussian distributions, when the number of generated clusters ($\#GC$) is the same as the number of Gaussian mixture groups ($\#MG$) of the input distribution, the percentages are higher than the others. The reason is the same as the one in Experiment 1. Another observation with the same reason as the one in Experiment 1 is that the percentages for the uniform data set are the lowest and those for the real data set are in the middle. These same observations show that *Recall* and *Precision* are good measurements for accuracy.

	No. of Generated Clusters				
$\#MG$	**2**	**4**	**8**	**16**	**32**
2	88.14	56.14	40.72	33.40	29.55
4	73.42	84.34	56.58	40.85	29.30
8	65.40	60.97	79.10	54.41	39.58
16	62.14	54.14	57.22	75.34	45.04
32	64.05	49.82	49.42	49.88	73.08

Table 11. Accuracy percentages for the data sets in Gaussian distributions in Experiment 4. $\#MG$ is the number of Gaussian mixture groups.

No. of Generated Clusters				
2	**4**	**8**	**16**	**32**
59.67	42.43	35.09	32.08	28.91

Table 12. Accuracy percentages for the uniform data set in Experiment 4.

From the experimental results, the accuracy percentages (for first cluster retrieval) are relatively high (73%-88%) for the data sets in Gaussian distribution when $\#GC = \#MG$, but we find that the larger the number of generated cluster partitions, the lower the accuracy percentage. It is because the chance of the occurrence of the boundary problem is higher when there are many generated clusters. It shows that our method can lessen the boundary problem, but it still cannot solve it completely.

4 Discussion and Conclusion

From Experiment 1, we find that although k-means is the most accurate way for clustering, it is also the slowest and most inefficient. RPCL turns out to be a good compromise when accuracy and efficiency are both taken into account. RPCL's accuracy is unaffected by the input size in general in Experiment 2. The results in Experiment 3 suggests that RPCL's accuracy depends on the underlying data distribution when the feature vector has high dimensions since the accuracy is lower in the uniform distribution case than the Gaussian distribution case. In the last experiment, the result correlates well with Experiment 1. Hence we may conclude that RPCL is most effective when the underlying data has a Gaussian distribution.

In summary, we propose a novel way to use RPCL algorithm to produce non-hierarchical cluster partitions for generating indexing structures. From the experimental results, we show that our RPCL indexing method gives good searching

No. of Generated Clusters				
2	**4**	**8**	**16**	**32**
67.72	56.97	48.39	44.76	37.51

Table 13. Accuracy percentages for the real data set in Experiment 4.

performance and it is the fastest method to build an indexing structure among the tested methods.

Our method using the non-hierarchical approach for indexing seems to be a good method, but there are still some limitations. First, it is not so efficient to perform insertion and deletion in our indexing method. Since our method uses a non-hierarchical indexing structure, there is no relationship in between two different levels' nodes. We have to find the target node at each level individually for insertion and deletion. Second, we find that our method still cannot solve the boundary problem completely. It does not give 100% nearest-neighbor result for a query in general. One possible extension of the proposed method is to add the branch-and-bound search technique to the indexing structure for better results.

References

1. N. Beckmann, H. P. Kriegel, R. Schneider, and B. Seeger. "The R*-tree: an efficient and robust access method for points and rectangles". *ACM SIGMOD Record*, 19(2):322–331, 1990.
2. J. L. Bentley. "Multidimensional Binary Search Trees Used for Associative Searching". *Communications of the ACM*, 18(9):509–517, 1975.
3. Tolga Bozkaya and Meral Ozsoyoglu. "Distance-Based Indexing for High-dimensional Metric Spaces". *SIGMOD Record*, 26(2):357–368, June 1997.
4. T. C. Chiueh. "Content-Based Image Indexing". In *Proceedings of the 20th VLDB Conference*, pages 582–593, September 1994.
5. R. A. Finkel and J. L. Bentley. "Quad Trees: A Data Structure for Retrieval on Composite Keys". *Acta Informatica*, 4(1):1–9, 1974.
6. A. Guttman. "R-trees: A Dynamic Index Structure for Spatial Searching". *ACM SIGMOD*, 14(2):47–57, June 1984.
7. Norio Katayama and Shinichi Satoh. "The SR-tree: an index structure for high-dimensional nearest neighbor queries". *SIGMOD Record*, 26(2):369–380, June 1997.
8. I. King, L. Xu, and L.W. Chan. Using rival penalized competitive clustering for feature indexing in Hong Kong's textile and fashion image database. In *Proceedings to the International Joint Conference on Neural Networks (IJCNN'98)*, pages 237–240. IEEE Computer Press, May 4-9, 1998.
9. T. Kohonen. The self-organizing map. *Proc. IEEE*, 78:1464–1480, 1990.
10. R. Rosenblatt. *Principles of Neurodynamics*. Spartan Books, New York, 1959. neural networks.
11. T. Sellis, N. Roussopoulos, and C. Faloutsos. "The R$^+$-tree: a dynamic index for multidimensional objects". In *Proceedings of the 13th VLDB Conference*, pages 507–518, 1987.
12. Lei Xu, Adam Krzyżak, and Erkki Oja. Rival penalized competitive learning for clustering analysis, RBF net, and curve detection. *IEEE Trans. on Neural Networks*, 4(4):636–649, 1993.
13. P. N. Yianilos. "Data structures and algorithms for nearest neighbor search in general metric spaces". In *Proc. of the 4th Annual ACM-SIAM Symp. on Discrete Algorithms*, pages 311–321, 1993.