DynDex: A Dynamic and Non-metric Space Indexer

King-Shy Goh, Beitao Li and Edward Chang
Electrical & Computer Engineering
University of California
Santa Barbara, CA 93106
kingshy, beitao@engineering.ucsb.edu, echang@ece.ucsb.edu

ABSTRACT

To date, almost all research work in the Content-Based Image Retrieval (CBIR) community uses Minkowski-like functions to measure similarity between images. In this paper, we first present a non-metric distance function, dynamic partial function (DPF), which works significantly better than Minkowski-like functions for measuring perceptual similarity; and we explain DPF's link to similarity theories in cognitive science. We then propose DynDex, an indexing method that deals with both the dynamic and non-metric aspects of the distance function. DynDex employs statistical methods including distance-based classification and bagging to enable efficient indexing with DPF. In addition to its efficiency for conducting similarity searches in very high-dimensional spaces, we show that DynDex remains quite effective when features are weighted dynamically for supporting personalized searches.

Keywords: high-dimensional index, non-metric distance function, similarity search.

1. INTRODUCTION

Similarity is one of the central theoretical constructs in information retrieval. To find objects that are similar to a query object, an information retrieval system must quantify similarity accurately and retrieve similar objects efficiently. This paper addresses the central issues of *quantifying perceptual similarity* and *indexing objects in perceptual (non-metric) spaces*.

Some important work in the cognitive psychology community has provided ample evidence that similarity is both *dynamic* and *non-metric* [19, 26]. Most recently, Goldstone [18] shows that similarity perception is the *process* that determines the respects (i.e., features, or attributes) for measuring similarity. More precisely, a similarity function for measuring a pair of objects is formulated only after the objects are compared, not before the comparison is made; and the respects for the comparison are *dynamically* activated in this formulation process. Let us use a simple example to explain. Suppose we are asked to name two places that are similar to England. Among several possibilities, Scotland and New England could be two reasonable answers. However, the respects in which England is similar to Scotland differ from those in which England is similar to New England. If we use the shared attributes of England and Scotland to compare England and New England, the latter pair might not seem similar, and vice versa. Thus, a distance function using a fixed set of respects (e.g., the traditional weighted Minkowski function) cannot capture objects that are similar in different sets of respects.

In this paper, we first present a non-metric distance function, dynamic partial function (DPF), which works significantly better than Minkowski-like functions for measuring perceptual similarity. Using our results from extensive image-data mining, together with similarity theories in cognitive science, we explain how DPF works and why it works well. Unfortunately, since DPF is a non-metric distance function, traditional indexing schemes that require metric-space properties [5, 16, 27] cannot work with DPF. Furthermore, since the *respects* for measuring similarity by DPF are determined in a pairwise fashion, and the feature space in which different object-pairs are compared can be different, coordinate-based indexing structures (e.g., tree-like structures [7, 14], hash-based structures [11]) cannot work with DPF for conducting similarity searches either.

We thus propose DynDex (DYnamic Non-metric-space inDEXer) to work with DPF. DynDex uses a statistical approach, which works in two steps. First, DynDex performs pairwise distance-based clustering to group similar objects together. To maximize IO efficiency, each cluster is stored in a sequential file. Second, DynDex models similarity searches as a classification problem. Given a query object q, DynDex predicts q's class membership and yields C most probable classes (i.e., clusters). DynDex then finds the objects that have the shortest pairwise distance with respect to q, using pairwise respects that are dynamically instantiated. We show that DynDex can achieve very high similarity search accuracy by probing just a small fraction of the dataset. We also show that DynDex scales well with data dimensionality and dataset size, and that DynDex holds quite well when features are dynamically weighted to support context-based (or personalized) similarity searches.

In summary, the contributions of this paper are as follows:

1. We present DPF to set up the context for proposing DynDex. While doing so, we present the link between

DPF and the similarity theories in cognitive science. We explain the need for DynDex and the design challenges it presents.

2. We propose DynDex, which employs statistical methods including distance-based classification and bagging to enable efficient indexing in non-metric spaces. Through extensive empirical studies with both high-dimensional data and very large datasets, we show that DynDex is both accurate and efficient.

The remainder of the paper is organized as follows: Section 2 presents DPF. Section 3 presents DynDex, showing how it clusters data in non-metric spaces. We also explain how a similarity search is conducted by DynDex through classification. Section 4 presents our empirical results. Finally, we provide our concluding remarks in Section 5.

2. DPF — DYNAMIC PARTIAL FUNCTION

Using some results of our extensive image-data mining [15], we explain how DPF is formulated and why DPF works effectively. We also discuss the link between DPF and similarity theories in cognitive psychology.

2.1 Visual Data Mining Results

To ensure that sound inferences could be drawn from our mining results, we carefully constructed our mining dataset as follows: First, we prepared for a dataset that was comprehensive enough to cover a diversified set of images. To achieve this goal, we collected 60,000 JPEG images from Corel CDs and from the Internet. Second, we defined "similarity" in a slightly restrictive way so that individuals' subjectivity could be excluded. For each image in the 60,000-image set, we performed 24 transformations (including down-sampling, cropping, rotation, scaling, and format transformation), and hence formed 60,000 similar-image sets. The total number of images in our testbed was 1.5 million.

We examined two popular distance functions used for measuring image similarity and found some of their assumptions questionable.

- Minkowski metric. The Minkowski metric is widely used for measuring similarity between objects (e.g., images). Suppose two objects X and Y are represented by two p dimensional vectors $(x_1, x_2, \cdots, x_p)$ and $(y_1, y_2, \cdots, y_p)$, respectively. The Minkowski metric $d(X, Y)$ is defined as

$$d(X,Y) = (\sum_{i=1}^{p} |x_i - y_i|^r)^{\frac{1}{r}}, \quad (1)$$

  where r is the Minkowski factor for the norm. Particularly, when r is set as 2, it is the well known Euclidean distance; when r is 1, it is the Manhattan distance (or L_1 distance). An object located close to a query object is deemed more similar to the query object. Measuring similarity by the Minkowski metric is based on one assumption: that similar objects should resemble the query object in all dimensions.
- Weighted Minkowski function. A variant of the Minkowski function, the weighted Minkowski distance function, has also been applied to measure image similarity. The basic idea is to apply weighting to prioritize important features. By assigning each feature a weighting coefficient w_i $(i = 1 \cdots p)$, the weighted Minkowski distance function is defined as:

$$d_w(X,Y) = (\sum_{i=1}^{p} w_i |x_i - y_i|^r)^{\frac{1}{r}}. \quad (2)$$

  By applying a static weighting vector for measuring similarity, the weighted Minkowski distance function assumes that similar images resemble the query images in the same set of features. For example, the important features for finding a scaled image are presumed to be the same as those for finding a cropped image.

We questioned the above assumptions upon observing how similar objects are located in the feature space. To better discuss our findings, we introduce a term we have found useful in our data mining work. We define the *feature distance* on the i^{th} feature as $\delta_i = |x_i - y_i|$, where $i = 1, \cdots, p$.

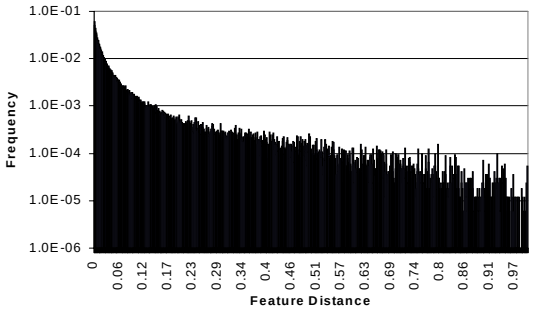
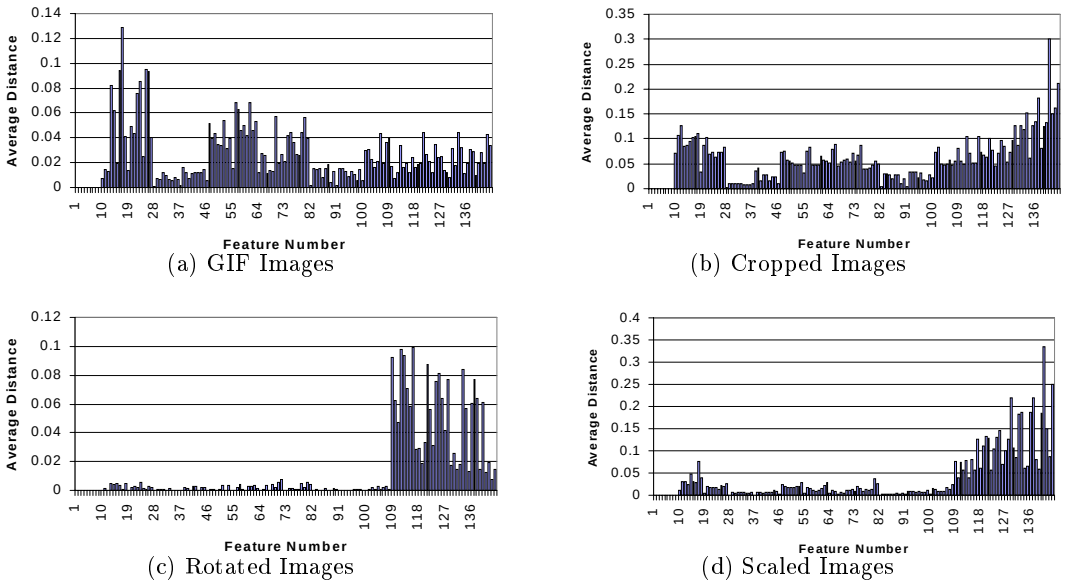


Figure 1: The Distributions of Feature Distances.

In our mining work, we first tallied the feature distances, δ, between similar images. Since we normalized feature values to be between zero and one, the range of δ is between zero and one. Figure 1 presents the distributions of δ between similar images. The x-axis shows the possible value of δ, from zero to one. The y-axis (in logarithmic scale) shows the percentage of the features at different δ values. We can see from the figure that a moderate portion of δ is in the high value range (≥ 0.5), which indicates that similar images may be quite dissimilar in many features. This observation suggests that the assumption of the Minkowski metric is not always accurate. Similar images are not necessarily similar in all features.

Next, we examined whether similar images resemble the query images in the same way. We tallied the feature distance (δ) of the 144 features[1] for different kinds of image transformations. Figure 2 presents four representative transformations: GIF, cropped, rotated, and scaled. The x-axis of the figure depicts the feature numbers, from 1 to 144. The first 108 features are various color features, and the last 36 are texture features. The figure shows that various similar images can resemble the query images in very different ways. GIF images have larger δ in color features (the first 108 features)

[1]For characterizing an image, we extracted a set of 144 image features. These features include color histograms, color means, color variances, color spreadness, color-blob elongation, and texture features in three orientations (vertical, horizontal, and diagonal) and three resolutions (coarse, medium, and fine) [25].



(a) GIF Images (b) Cropped Images

(c) Rotated Images (d) Scaled Images

Figure 2: The Average Feature Distances.

than in texture features (the last 36 features). In contrast, cropped images have larger δ in texture features than in color features. For rotated images, the δ in colors comes close to zero, although its texture feature distance is much greater. A similar pattern appears in the scaled and the rotated images. However, the magnitude of the δ of scaled images is very different from that of rotated images. This observation suggests that the assumption of the weighted Minkowski metric is also questionable. Images similar to the query images can be similar in differing features. (This observation runs counter to the assumption recently made in [30], that similar images are similar in the same way.)

2.2 Cognitive Science Interpretation

Using the above mining results, together with theories and examples from cognitive psychology, we discuss the progress of the following three similarity paradigms in cognitive science in the last few decades. At the end of the discussion, we present our dynamic partial function (DPF).

1. *Similarity is a measure of all respects.* As we discussed in Section 2.1, a Minkowski-like metric accounts for all respects (i.e., all features) when it is employed to measure similarity between two objects. Our mining result shown in Figure 1 is just one of a large number of counter-examples demonstrating that the key assumption of the Minkowski-like metric is questionable. The psychology studies of [18, 26] present many examples showing that the Minkowski model appears inadequate to describe human similarity judgments. (Please consult the references for the examples.)

2. *Similarity is a measure of a fixed set of respects.* Substantial work on similarity has been carried out by cognitive psychologists. The most influential work is perhaps that of Tversky [26], who suggests that similarity is determined by matching features of compared objects, and integrating these features by the formula

$$S(A,B) = \theta f(A \cap B) - \alpha f(A - B) - \beta f(B - A). \quad (3)$$

The similarity of A to B, $S(A,B)$, is expressed as a linear combination of the common and distinct features. The term $(A \cap B)$ represents the common features of A and B. $(A-B)$ represents the features that A has but B does not; $(B-A)$ represents the features that B has but A does not. The terms θ, α, and β reflect the weights given to the common and distinctive components, and function f is often assumed to be additive [18].

The weighted Minkowski function [23] and the quadratic-form distances [9, 12] are the two representative distance functions that match the spirit of Equation 3. The weights of the distance functions can be learned via techniques such as relevance feedback [22, 23, 24], principal component analysis, and discriminative analysis [30]. Given some similar and some dissimilar objects, the weights can be adjusted so that similar objects can be better distinguished from other objects.

However, the assumption made by these distance functions, that all similar objects are similar in the same respects [30], is also questionable. As we have shown in Figure 2, GIF, cropped, rotated, and scaled images are all similar to the original images, but in differing features.

3. *Similarity perception is a process that provides respects for measuring similarity.* Murphy and Medin [19] provide early insights into how similarity works in human perception: "The explanatory work is on the level of determining which attributes will be selected, with similarity being at least as much a consequence as a cause of a concept coherence." The result presented in Figure 2 echos this statement—that is, objects can be similar to the query object in different respects. A distance function using a fixed set of respects cannot capture objects that are similar in different sets of respects. A distance function for measuring a pair of objects is formulated only after the objects are compared, not before the comparison is made. The respects for the comparison are activated in this formulation process. The activated re-

spects are more likely to be those that can support coherence between the compared objects. The England, Scotland, and New England example presented in Section 1, and our mining result presented in Figure 2, both confirm this *dynamic activation* similarity paradigm.

These points lead to the design of the *dynamic partial* distance function. Let $\delta_i = |x_i - y_i|$, for $i = 1, \cdots, p$. We first define sets Δ_m as

$$\Delta_m = \{The \quad smallest \quad m \; \delta's \quad of \quad (\delta_1, ..., \delta_p)\}.$$

The first approximation of a desired perceptual function looks like

$$d_{DPF}(X, Y) = (\sum_{\delta_i \in \Delta_m} {\delta_i}^r)^{\frac{1}{r}}. \quad (4)$$

The above formulation shows that DPF activates different features for different object pairs. The activated features are those with minimum differences–those which provide coherence between the objects. If coherence can be maintained (because a sufficient number of features are similar), then the objects paired are perceived as similar. The empirical studies in image retrieval [15] and in shot-boundary detection [28] show that DPF performs significantly better than Minkowski-like functions.

3. DYNDEX - THE DYNAMIC NON-METRIC SPACE INDEXER

DPF is designed to model perceptual similarity more accurately. At the same time, it poses new challenges to the indexing process due to these special properties:

1. DPF compares different pairs of objects in different feature subspaces. That is, features are dynamically activated while comparing the similarity between objects.
2. DPF is a non-metric distance function. Notably, it does not satisfy the triangle inequality (see proof in Appendix A), on which many traditional indexing methods are founded.

The various traditional indexing structures proposed for indexing high-dimensional data can be broadly classified into two categories:

1. Coordinate-based methods. The coordinate-based methods work on objects residing in a vector space. These objects are usually described by a D-dimensional feature vector, and the similarity between any two of them is measured by the distance between their coordinates in the vector space. Most methods divide the feature space into partitions (e.g., R*-tree [2], SR-tree [14], TV-tree [17], and X-tree [3]). The partitioning is designed so that similar objects can be found by visiting a minimal number of partitions. By carrying out vector operations in the coordinate space, the schemes implicitly employ a metric function (Euclidean distance) as the measuring function for similarity. Besides, coordinate-based methods treat objects as vectors in a fixed feature space, which does not allow them to model the dynamic nature of quantifying similarity between objects. For these reasons, coordinate-based methods are unable to index DPF effectively. In Section 4.1, our empirical study further confirms that coordinate-based methods cannot be transplanted to index DPF effectively.

2. Distance-based methods. The distance-based methods index objects based on pairwise distances. Most distance-based indexing structures for a metric space are variations of the basic $m-$ary search tree (e.g., MVP-tree [4] and M-tree [7]). They mainly differ in their methods for determining equivalent classes among the objects, either by pivot or Voronoi partitioning. However, they all take advantage of the triangle inequality to prune their search trees, which limits their usage to indexing data in a metric space.

To overcome the difficulties plaguing traditional indexing methods, we designed DynDex to meet the following two critical DPF indexing requirements:

1. It must be able to work in a dynamic and non-metric feature space where only the pairwise-distances between objects are given.
2. It should support high-dimensional indexing efficiently.

To fulfill the above requirements, DynDex adopts a statistical approach that is able to support approximate similarity searches. Our hypothesis is that if a query object's class prediction yields C probable classes, then the probability is high that its nearest neighbors can be found in these C classes. Therefore, by searching for the most likely classes into which the query object might be classified, we can harvest most of the similar objects. DynDex consists of two steps:

1. Cluster similar objects on disk to minimize disk latency for retrieving similar objects. Similarity is measured by the DPF pairwise-distance between the objects. (Section 3.1)
2. Use classification methods to model a similarity search. First, DynDex identifies the top$-C$ clusters to which the query object most likely belongs. Then from these clusters, DynDex searches for k nearest objects, commonly called top$-k$ nearest neighbors ($k-$NNs), to the query object. (Section 3.2)

In the remainder of this section, we will first describe the clustering algorithm we use (Section 3.1). Second, we present the four query-object classification schemes (Section 3.2). We then discuss how we conduct a similarity search when presented with a query object (Section 3.3). We also show that our scheme can be adapted to include dynamic feature weighting used in context-based DPF (Section 3.4). Finally, we present an error-reduction method that improves classification accuracy and achieves better search results. (Section 3.5).

3.1 Clustering Phase

Clustering is the process of partitioning objects into groups (clusters), so that similar objects fall into the same group. Data clustering has been extensively studied in statistics, machine learning, and pattern recognition. Several algorithms (e.g., BIRCH [29], CLIQUE [1], and TSVQ [10]) have been developed to cluster data in coordinate space. However, these methods all implicitly employ metric function as the measure for similarity, since they rely on vector operations in a coordinate space.

A number of pairwise distance-based clustering algorithms have been proposed to cluster data in distance space, in which the only operation possible on data objects is the computation of distance. Among those, CLARANS [20] is a representative

one, based on randomized search. Since this existing algorithm would serve our purposes, we adopted CLARANS. We chose CLARANS for the following reasons:

1. CLARANS is a pairwise medoid-based algorithm, which only requires a distance matrix to do clustering. This property makes it appropriate for clustering in the non-metric DPF space.
2. CLARANS has been shown to be relatively efficient and effective[2] in clustering with a large dataset.

The formal description of the CLARANS algorithm can be found in Appendix B.

3.2 Classification Phase

Let q denote the query object. The classifier's mission is to find the top C clusters to which q most likely belongs. Typically, a cluster in a metric space is characterized by its centroid, its standard deviation of the cluster's data objects, and its covariance, which models the orientation of the cluster. These parameters are combined to form a cluster prediction. However, these traditional parameters are not directly applicable when we are dealing with the non-metric DPF. Instead, we select a subset of the cluster's objects to form a representative set R for that cluster. By computing the distance from q to the objects in R, we can estimate the closeness of q to the cluster. The distance from q to R determines the cluster membership of q.

In the following, we present four methods that are used to create the set of representative data objects for the clusters. The four schemes are as follows:

1. *Medoid Scheme*. The most centrally-located object of the cluster is used.
2. *Random Scheme*. We randomly select some objects from each cluster to represent it.
3. *Atypical Scheme*. We pick objects that are far from all other objects within the cluster.
4. *Correlation Scheme*. We choose objects that are the least correlated with each other within the cluster.

The notations used to describe the schemes are given in Table 1. Each scheme chooses a set R_j for each cluster C_j. Once R_j is formed for all clusters, we can compute the clusters' ranks (see Section 3.3) to process a query.

3.2.1 Medoid Scheme

Each cluster C_j generated by CLARANS is associated with a medoid m_j, the most central object in the cluster. In this scheme, the representative set R_j of C_j consists of only the medoid. The clusters are then ranked by the proximity of their medoids to the query object q using $d_{DPF}(q, m_j)$ of Equation 4. The scheme is analogous to traditional methods for predicting cluster membership. The closer q is to m_j, the higher the cluster C_j's ranking.

3.2.2 Random Scheme

[2]Of late, techniques [8] have been proposed to improve the IO efficiency of CLARANS. However, most of them rely on vector operations and thus limit their usage to a coordinate space.

D	Dimensionality of the data space
C_j	Cluster j of dataset
R_j	Representative set of points C_j
N_{Cj}	Size of cluster C_j
N_{Rj}	Size of set R_j
$d_{DPF}(x, y)$	DPF distance between objects x and y (see Equation 4)
$corr(x, y)$	Correlation between objects x and y (see Equation 5)
d_{max}	Maximum distance of an object in cluster C_j to all objects in R_j
c_{max}	The object in cluster C_j which gives d_{max}
$corr_{min}$	Minimum correlation of an object in cluster C_j to all objects in R_j
c_{min}	The object in cluster C_j which gives $corr_{min}$

Table 1: Notations for the Classification Schemes.

In this scheme, we randomly sample a percentage of objects from each cluster to form cluster C_j's representative set R_j. Out of N_{Rj} objects in the set R_j, we pick the one with the minimum distance to the query object q to estimate the q's proximity to cluster C_j. A smaller distance implies that the cluster rank is higher.

3.2.3 Atypical Scheme

An atypical object is one which is dissimilar to other objects in the cluster. Figure 3 shows an example where using atypical points to determine closeness to a cluster is important. Suppose both cluster A and B are represented only by their medoids a_0 and b_0 and the query object is as shown. In this case cluster A will be ranked higher and hence retrieved first even though cluster B has objects that are located closer to the query object. If we also use atypical objects to represent cluster B, the object b_1 will be added since it is the farthest from b_0. The distance from b_1 to the query will rank cluster B higher. Therefore, the retrieval of cluster B will precede that of cluster A, and hence more NNs will be returned after the first IO.

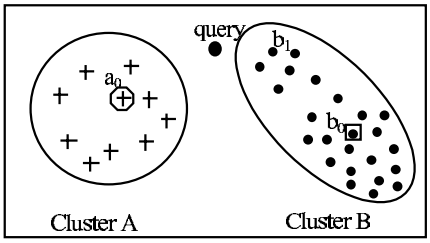


Figure 3: Importance of using Atypical Objects to Represent a Cluster Instead of Medoids Alone.

To find the set of atypical objects, we first choose an initial object randomly from the cluster C_j to insert into R_j. The second object to be added will be the object in C_j that is farthest away from the first object chosen. Subsequent objects are chosen if they are far away from all existing objects in R_j. Figure 4 shows the pseudo code for the algorithm.

3.2.4 Correlation Scheme

In this scheme, we want to form a representative set with objects that are uncorrelated with each other. Before we define what we mean by uncorrelated, we need to introduce the concept of redundancy. Jacobs et. al [13] stipulate that a suitable measure of whether an object is a good substitute

```
compute_atypical (C_j, N_Cj, R_j, N_Rj)
  initialize R_j to be an empty set
  randomly pick an object from C_j and insert into R_j
  count := 1
  while (count != N_Rj) do
    d_max := 0
    for i := 1 to N_Cj do
      dist := sum_{k=1}^{count} d_DPF( C_j[i], R_j[k] )
      if (d_max < dist) then  d_max := dist,  c_max := C_j[i]
    end do
    insert c_max into R_j and increment count by 1
  end do
```

Figure 4: Atypical Scheme for Choosing Atypical Objects to Represent a Cluster.

for another should be based on redundancy instead of distance alone. The redundancy between two objects x and y is defined as the probability that x and y have similar distances to other objects in the data set. Having two objects that are interchangeable in the representative set does not add useful information for performing cluster ranking. Thus, the set should consist of objects that are non-redundant to one another. A good estimator of redundancy is the correlation between the distance vectors of the objects.

Given two data objects x and y and their corresponding distance vectors $\mathbf{x}$ and $\mathbf{y}$, where $\mathbf{x}$ ($\mathbf{y}$) is the vector of distances from x (y) to all other objects in the cluster, the correlation between x and y is measured by:

$$corr(x, y) = \frac{\mathbf{x} - \mu_x}{\sigma_x} \cdot \frac{\mathbf{y} - \mu_y}{\sigma_y} \tag{5}$$

where μ_x denotes the mean of $\mathbf{x}$, and μ_y the mean of $\mathbf{y}$. Similarly, σ_x and σ_y denote the standard deviations of $\mathbf{x}$ and $\mathbf{y}$. The vector distances are computed using DPF from Equation 4.

We want to choose a set of objects from each cluster that are not redundant to each other, that is, the correlation value is low. The algorithm to select uncorrelated objects is similar to that for atypical objects. The distance computation is replaced by Equation 5, and we choose the objects that are least correlated with objects in R. Figure 5 shows the details of the correlation scheme.

```
compute_correlate (C_j, N_Cj, R_j, N_Rj)
  initialize R_j to be an empty set
  randomly pick an object from C_j and insert into R_j
  count := 1
  while (count != N_Rj) do
    corr_min := a very large real number
    for i := 1 to N_Cj do
      totcorr := sum_{k=1}^{count} corr( C_j[i], R_j[k] )
      if (corr_min > totcorr) then
        corr_min := totcorr,  c_max := C_j[i]
    end do
    insert c_max into R_j and increment count by 1
  end do
```

Figure 5: Correlation Scheme for Choosing Uncorrelated Objects to Represent a Cluster.

3.3 Similarity Search

Given a query object q, the clusters C_js and their representative set R_js, a similarity search proceeds in two steps:

1. Compute the ranks of the clusters with respect to q using the algorithm **compute_cluster_rank** in Figure 6. A highly ranked cluster is considered "closer" to the query object and hence more likely to contain similar objects that we call nearest neighbors (NNs). For each query object, a cluster-list is formed, and the clusters are sorted by their ranks in descending order.

2. Read the "nearest" cluster from disk into memory and perform a sequential scan on the cluster to find the top-k NNs. If a higher recall rate is desired, we read the next nearest cluster on the cluster-list.

```
compute_cluster_rank (R_j, N_Rj, q)
  mindist := a very large real number
  for i := 1 to N_Rj do
    dist := d_DPF( q, R_j[i] );
    if (dist < mindist ) then mindist := dist
  end do
  return (mindist)
```

Figure 6: Algorithm to Compute a Cluster's Rank

3.4 Context-based DPF

To take subjectivity into consideration, a distance function should allow features to be weighted differently by different users and in different search contexts. Cognitive study [18] has shown that perceptual similarity can be context-based. To best capture the context of a query, many relevance feedback schemes formulate a similarity function through weighting features based on their relevance to the query concept [21]. We thus propose the weighted DPF as

$$d_{DPF-w}(X, Y) = \Big(\sum_{\delta_i \in \Delta_m} w_i {\delta_i}^r\Big)^{\frac{1}{r}}. \tag{6}$$

where w_i is the weight value of feature i. The value of w_i can be obtain by introducing relevance feedback during the retrieval process.

A weighted similarity search is conducted in the same manner as when no weights are used. We replace the distance function $d_{DPF}(\ q, R_j[i]\ )$ in the **compute_cluster_rank** algorithm of Figure 6 with the Equation 6.

3.5 Bagging for Classification Error-Reduction

Bagging [6] has been applied to ensemble classification schemes. Bagging smoothes the variability of non-linear components within a classifier and thus reduces its classification error. When the errors made by different bags are independent, the overall error produced by an ensemble scheme will be smaller than that made by the bags separately. Since our NN-search is modeled as a classification problem, having lower classification errors can be beneficial. First, we need to construct several bags using the same dataset. During the classification phase, majority voting is conducted among the bags to achieve the smoothing effect.

We perform bagging by first using CLARANS to generate different sets of clusters, each created with a different set of seeding objects at the start of clustering. We then treat each

set of clusters as one bag. We find the top clusters of a query object to form each bag's cluster list. The final top clusters will be aggregated from the lists of all bags. Suppose we have two bags with cluster lists $bag\ 1 = \{C_1, C_2, ..., C_{30}\}$ and $bag\ 2 = \{D_1, D_2, ..., D_{30}\}$. To form the final top$-30$ cluster list, we compare the entries in each bag's cluster list and pick the cluster that is closest to the query object. For example, if C_1 is closer to the query than D_1, we select C_1 as the top-1 cluster. And if D_2 is closer to the query than C_2, then D_2 will be the top-2 cluster.

4. EMPIRICAL STUDY

Given a query, we perform a similarity search to return the k most similar objects, or k nearest neighbors ($k-$NN). We first establish a benchmark by scanning the entire dataset to find the top-k NNs for each query; this constitutes the "golden" results set. The metric we use to measure the search result is *recall*. In other words, we are interested in knowing what fraction of top-k golden results are retrieved after X IOs. We do not use precison, which is the fraction of retrieved objects that are among the top-k golden results. In our environment, we believe that precision is not the most useful metric since the main overhead is IOs, not the number of non-golden objects that are seen.

Our experiment involves the following evaluations:

1. **$k-$NN Query Evaluation** (Section 4.1). We evaluate the medoid, the random, the atypical, and the correlation schemes for a $k-$NN query. The accumulated recall for up to 30 IOs is used as the metric. The best scheme is that which achieves the highest recall given a fixed number of IOs.
2. **Cluster Size Evaluation** (Section 4.2). We investigate the effect of cluster size on recall and IO time.
3. **Context-based DPF Evaluation** (Section 4.3). We test the feasibility of using dynamic feature weighting from relevance feedback on the medoid scheme.
4. **Error-reduction Method Evaluation** (Section 4.4). Using bagging, we evaluate the effect of classification accuracy on recall of $k-$NN queries.
5. **Scalability Evaluation** (Section 4.5). Finally, we test the scalability of our indexing scheme by performing $k-$NN queries on a very large dataset.

The studies in [15] have shown that $m = 114$ and $r = 2$ can model perceptual similarity in images well, we use these values for the DPF distance function of Equation 4.

The two datasets we use for our experiments are:

- **51K-image dataset**. From Corel Image CDs, we select $51,000$ images to cluster. Images from this collection are widely used by the research communities dealing with computer vision and image processing. We then randomly select an additional $1,830$ separate images from the CDs to be used as the query set.
- **0.5M-image dataset**. The second set is generated from a base of $70,000$ images that include images from Corel CDs and the Internet. We perform 24 transformations on these images. The transformations include rotation, cropping, scaling up and down, down-sampling and GIF-to-JPEG conversion. We then randomly select 0.5 million images to cluster, and from the remaining ones, choose $1,000$ images to be used as query images.

Note that query images are not indexed. The first four parts of the experiment are done using the 51K-image dataset and the scalability test is done using the 0.5M-image dataset. Images in both sets have 144 features: 108 for colors and 36 for textures [25].

4.1 $k-$NN Query Evaluation

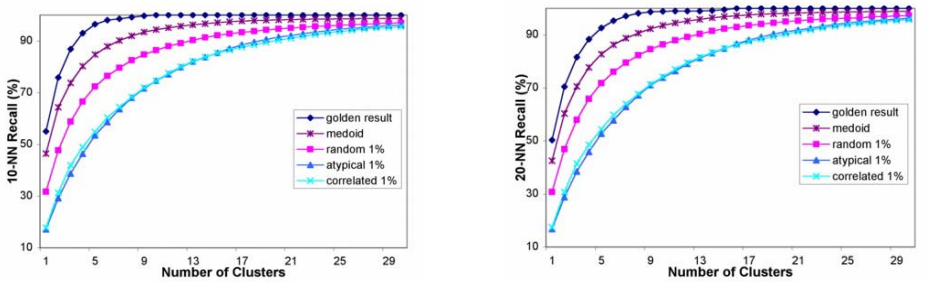
We evaluate the recall performance of the medoid, random, atypical, and correlation ranking schemes for $10-$NN and $20-$NN queries. We sample 1% of image objects from each cluster for the latter three ranking schemes. The accumulated recall is computed for the top-30 clusters into which a query image may be classified. Figure 7(a,b) shows the query results for the four schemes. The x-axis shows the number of IOs that are performed to achieve a particular recall, and the y-axis shows the accumulated recall. The "golden" results are also plotted for comparison. Of the four schemes, the medoid scheme performs best with more than 45% recall when only one cluster is retrieved. The other schemes reach recalls of 31% (random), 17% (atypical) and 17% (correlated). However, all schemes lags the "golden" results by at least 10%. As the medoid scheme performs best, we use it for all subsequent evaluations.

As the cluster sizes are not uniform, we also plot the $10-$NN and $20-$NN recalls against the percentage of data read (see Figure 7(c,d)). The trend is similar to that of recall versus number of clusters, with the medoid scheme outperforming all the others.

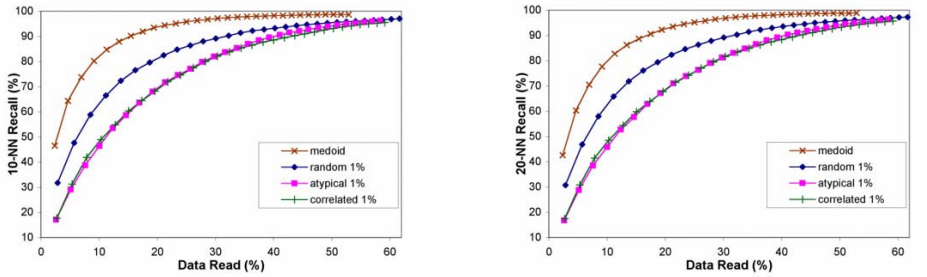
The good performance of the medoid scheme can be attributed to the central role that medoids play in the clustering algorithm. The cluster membership is determined by the distance between an image and the cluster medoid. Hence, when we classify the query image during a similarity query using medoids, the image will be placed in the same cluster as if the image had been present during the clustering phase.

In order to explain why the performance of the random scheme is superior to that of more complex ones like atypical and correlated, we first note that the query images are sampled randomly from the dataset and thus have similar distribution. The random scheme will tend to pick points that mirror the cluster distribution whereas the atypical or correlated schemes will pick points which are outliers within a cluster. Consequently, the random scheme can classify the query image into the cluster where it most likely belongs to. Besides, the curse of dimensionality could render all images atypical, making the heuristic less effective.

Next, we compare recall performance of CLARANS with the TSVQ clustering algorithm. TSVQ forms clusters with Voronoi partitioning and the metric Euclidean distance. During retrieval, the distance between the query image and the TSVQ cluster centroid is used for classification. In Figure 8, TSVQ method 1 uses Euclidean distance to classify the query image, and TSVQ method 2 uses DPF. The CLARANS results shown use the medoid scheme for classification. For both

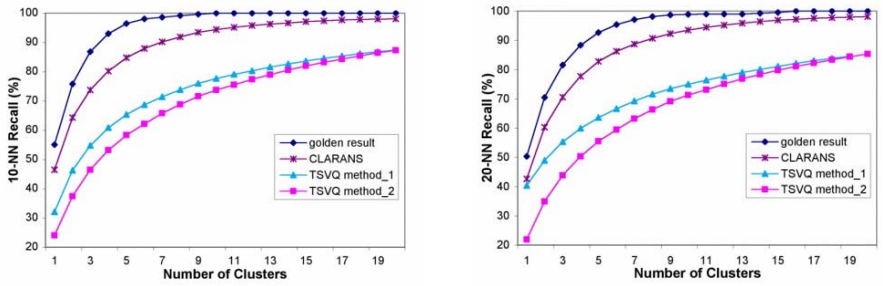


(a) 10−NN Recall versus No. of Clusters (b) 20−NN Recall versus No. of Clusters



(c) 10−NN Recall versus Data Read (%) (d) 20−NN Recall versus Data Read (%)

Figure 7: Recall k−NN Queries Using Various Cluster Ranking Scheme.



(a) 10−NN (b) 20−NN

Figure 8: Comparison of Recall using CLARANS and TSVQ for Clustering.

10−NN and 20−NN queries, CLARANS outperforms both TSVQ methods by more than 15%. This shows that an algorithm optimized for a metric space cannot be applied to a non-metric space without degradation in performance.

4.2 Cluster Size Evaluation

Next, we evaluate the effect of cluster size on recall rate. As CLARANS is a k−medoid clustering algorithm, we produce different numbers of clusters by altering the value of k. Having more clusters effectively reduces the size of each cluster. For our evaluation, we vary the value of k from 100 to 400, and Figure 9(a) shows the 10−NN recall results for the medoid and random schemes when different cluster sizes are used. The figure shows that having a larger cluster size (smaller k) achieves a better recall rate. By decreasing k from 400 to 100, the recall improves by at least 10%. This is due to the higher probability that a large cluster will enclose more NNs.

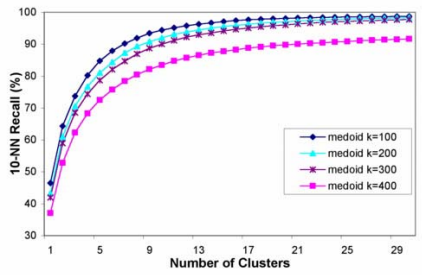
Usually, the drawback in using larger cluster sizes is in IO time. We use a quantitative model to compute IO time. Let C be the cluster size in bytes, N be the number of IOs, TR be the transfer rate, and T_{seek} be the average disk seek time. The IO time is estimated as

$$T = N \times T_{seek} + \frac{C \times N \times 8}{TR}. \quad (7)$$

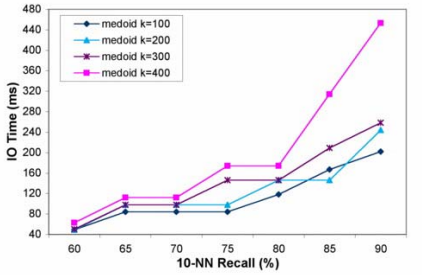
Assuming a transfer rate TR of 130Mbps, and a seek time T_{seek} of $14ms$, the IO times for each value of k are plotted in Figure 9(b). The estimated IO time is valid as we find that it matches closely to the wall-clock run time of our experiments. When a low recall (60%) is required, we see that all cluster sizes perform equally well in terms of retrieval speed (or IO time). However, we see that when higher recall is desired, a larger cluster size (smaller k) can give us a higher recall while retrieving fewer clusters. This means that reducing seek overhead causes the IO time to be lesser. We observe that having $k = 100$ gives us a good balance between recall performance and IO cost.

4.3 Context-based DPF Evaluation

To simulate a more realistic query environment, we test DynDex with dynamic weights learned through relevance feedback rather than just randomly assigning them. For each query image, we first retrieve 20 images and randomly assign a positive or neg-



(a) Recall for Different Cluster Size



(b) IO Time versus Recall

Figure 9: 10−NN Recall for Various Cluster Size

ative label to each image. Next, we refine the feature weights using the relevance feedback method presented in [24]. Each query image will then possess a unique set of weight values for each of its features. The weight values are incorporated into DPF function as shown in Equation 6. Figure 10 shows the 10−NN and 20−NN recall of up to 20 clusters. We also plotted the golden results when dynamic feature weighting is employed.

First, we observe that the weighted golden results for the initial 20 clusters retrieved are about 10% lower than those when no weights are used (compare with Figure 7). This observation suggests that DynDex will suffer some performance degradation when dynamic weighting is used. However, the recall results are still reasonable, especially when more clusters are retrieved. After retrieving three clusters, we can obtain a recall rate of 50% and by the tenth cluster, the recall is 80%.

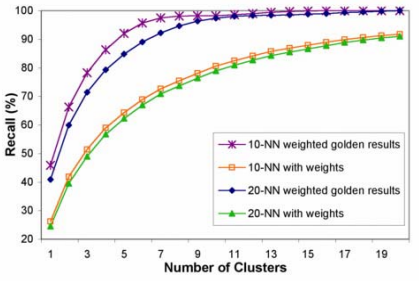


Figure 10: 10−NN and 20−NN Recall when Feature Weighting is Used.

4.4 Error-Reduction Method Evaluation

Figure 11 shows the bagging results using different sets of clusters for $k = 400$. We evaluate the effect on recall for up to three bags, as past studies have shown that the greatest gain is achieved with the first few bags. In this setting, we observe that bagging improves recall by about 3% with two bags and 5% with three bags for the first 20 retrieved clusters. The recall gain tapers off as more clusters are retrieved. Under some circumstances in which data replication is required, or when only a small number of IOs is permitted, the use of bagging can boost the initial recall results moderately.

4.5 Scalability Evaluation

We index a 0.5M-image dataset and test for changes in recall rate and query time. We create 100 clusters for this dataset and conduct 10−NN and 20−NN queries on them. Figure 12

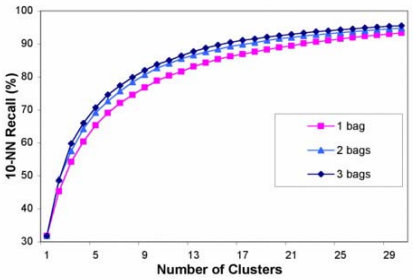


Figure 11: Recall of 10−NN Query Using Bagging.

shows the recall results and the IO time required. Despite using a dataset 10 times larger, the indexing scheme is still able to deliver good recall performance. After retrieving 10 clusters, recalls of both types of query are more than 90%. Since the cluster size is large, the IO cost incurred for the first few retrieved clusters is high compared to the recall gain. For recall of up to 80%, the IO time for the 0.5M-image dataset is about twice that of the 51K-image dataset. But as more clusters are retrieved, the IO cost to achieve a desired recall matches the cost of the smaller dataset. Again, this is due to the higher likelihood that the large clusters will enclose more NNs.

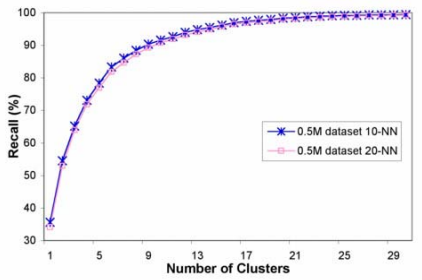
5. CONCLUSIONS

We introduced a distance function DPF discovered through extensive data mining work. And we also showed its effectiveness in measuring perceptual similarity and its connection with theories of modern cognitive psychology.

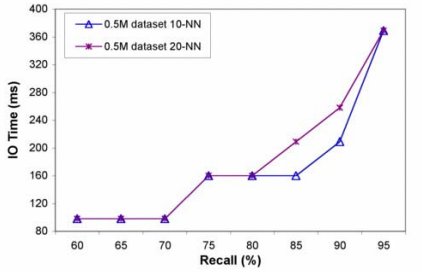
To support fast retrieval speed for high-dimensional data in a non-metric and dynamic space, we proposed a clustering and classification combined approach, DynDex, to index data. Our approach can support efficient similarity searches as well as context-based searches via relevance feedback.

We demonstrated the effectiveness of DynDex with the following findings:

- k−NN Query. With our clustering and medoid classification scheme, we showed that DynDex can index DPF competently. We also showed that DynDex outperforms TSVQ (an indexing method designed for metric spaces) by a large margin.
- Context-based DPF. DynDex is able to support dynamic



(a)Recall versus No. of Clusters



(b)IO Time versus Recall

Figure 12: Results for 0.5M-image dataset, (a) shows 10−NN and 20−NN recall, and (b) shows the IO time.

feature weightings for context-based searches with a small degradation in recall.

- Error-reduction Method. Bagging can help improve recall when a moderate number of bags are used. When data must be replicated for improving reliability and load-balancing, bagging can be employed to achieve these goals, as well as to improve search accuracy.
- Scalability. We also showed that DynDex is scalable to very large datasets, with little degradation in recall. We attained more than 95% of the 20-NN in less than one second on a dataset containing half million images.

Our future work extends in two main directions. First, we plan to improve the clustering algorithm to be more IO efficient and more scalable to very large datasets. Second, we plan to improve the performance of DynDex to better support context-based DPF with more effective query classification methods.

6. REFERENCES

[1] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. *Proceedings of ACM SIGMOD*, June 1998.

[2] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: an efficient and robust access method for points and rectangles. *Proceedings of ACM SIGMOD*, May 1990.

[3] S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-Tree: An index structure for high-dimensional data. *Proceedings of the 22nd VLDB*, August 1996.

[4] T. Bozkaya and M. Ozsoyoglu. Distanced-based indexing for high-dimensional metric spaces. *Proc. of ACM SIGMOD*, pages 357–368, 1997.

[5] P. S. Bradley, U. M. Fayyad, and C. A. Reina. Scaling em (expectation-maximization) clustering to large databases. *Microsoft Technical Report MSR-TR-98-34*, November 1999.

[6] L. Breiman. Bagging predicators. *Machine Learning*, pages 123–140, 1996.

[7] P. Ciaccia, M. Patella, and P. Zezula. M-Tree: An efficient access method for similarity search in metric spaces. *Proceedings of the 23rd VLDB*, August 1997.

[8] M. Ester, H.-P. Kriegel, and J. Sander. Focussing technique for efficient class indentification. *Proc. of the 4th Intl. Sym. of Large Spatial Databases*, 1995.

[9] M. Flickner, H. Sawhney, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yanker. Query by image and video content: the QBIC system. *IEEE Computer*, 28(9):23–32, 1995.

[10] A. Gersho and R. Gray. *Vector Quantization and Signal Compression*. Kluwer Academic, 1991.

[11] P. Indyk, A. Gionis, and R. Motwani. Similarity search in high dimensions via hashing. *Proceedings of the 25th VLDB*, September 1999.

[12] Y. Ishikawa, R. Subramanya, and C. Faloutsos. Mindreader: Querying databases through multiple examples. *VLDB*, 1998.

[13] D. W. Jacobs, D. Weinshall, and Y. Gdalyahu. Classification with non-metric distances: Image retrieval and class representation. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 2000.

[14] N. Katayama and S. Satoh. The SR-Tree: An index structure for high-dimensional nearest neighbor queries. *Proceedings of ACM SIGMOD*, May 1997.

[15] B. Li, E. Chang, and T. Wu. Dpf — a perceptual distance function for image retrieval. *IEEE International Conference on Image Processing*, September 2002.

[16] C. Li, E. Chang, H. Garcia-Molina, and G. Wiederhold. Clustering for approximate similarity queries in high-dimensional spaces. *IEEE Transaction on Knowledge and Data Engineering (to appear)*, 2001.

[17] K.-L. Lin, H. V. Jagadish, and C. Faloutsos. The TV-tree: an index structure for high-dimensional data. *VLDB Journal*, 3(4), 1994.

[18] D. L. Medin, R. L. Goldstone, and D. Gentner. Respects for similarity. *Psychological Review*, 100(2):254–278, 1993.

[19] G. Murphy and D. Medin. The role of theories in conceptual coherence. *Psychological Review*, 92:289–316, 1985.

[20] R. T. Ng and J. Han. Efficient and effective clustering methods for spatial data mining. *Proceedings of the 20th VLDB*, September 1994.

[21] K. Porkaew, K. Chakrabarti, and S. Mehrotra. Query refinement for multimedia similarity retrieval in mars. *Proceedings of ACM Multimedia*, November 1999.

[22] K. Porkaew, S. Mehrota, and M. Ortega. Query reformulation for content based multimedia retrieval in mars. *ICMCS*, pages 747–751, 1999.

[23] J. Rocchio. Relevance feedback in information retrieval. In G. Salton, editor, *The SMART retrival system: Experiments in automatic document processing*. Prentice-Hall, 1971.

[24] Y. Rui, T. S. Huang, and S. Mehrotra. Content-based image retrieval with relevance feedback in mars. *Proc. IEEE Int. Conf. on Image Proc.*, 1997.

[25] S. Tong and E. Chang. Support vector machine active learning for image retrieval. *ACM International Conference on Multimedia*, October 2001.

[26] A. Tversky. Feature of similarity. *Psychological Review*, 84:327–352, 1977.

[27] J. Z. Wang, J. Li, and G. Wiederhold. Wise: A wavelet-based image search engine with efficient feature vector clustering and classification. *Submitted for journal publication*, 1998.

[28] Y. Wu, E. Chang, and B. Li. Video shot detection. *UCSB Technical Report (submitted to ICME 2002)*, 2002.

[29] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: An efficient data clustering method for very large databases. *Proceedings of ACM SIGMOD*, June 1996.

[30] X. S. Zhou and T. S. Huang. Comparing discriminating transformations and svm for learning during multimedia retrieval. *Proc. of ACM Conf. on Multimedia*, pages 137–146, 2001.

Appendix A. Proof of DPF as Non-Metric Distance Functions.

A distance function, $d(i_1, i_2)$, is metric [13] when for any three objects i_1, i_2, and i_3:

1. $d(i_1, i_2) \geq 0$.
2. $d(i_1, i_2) = 0$ iff. $i_1 = i_2$.
3. $d(i_1, i_2) = d(i_2, i_1)$ (symmetry).
4. $d(i_1, i_3) \leq d(i_1, i_2) + d(i_2, i_3)$ (the triangle inequality).

For DPF functions as:

$$d_{DPF}(X, Y) = (\sum_{\delta_i \in \Delta_m} {\delta_i}^r)^{\frac{1}{r}},$$

where

$$\Delta_m = \{The \;\; smallest \;\; m \; \delta's \;\; of \;\; (\delta_1, ..., \delta_p)\}.$$

Obviously, when $m < p$, condition (2) will not be satisfied since two different objects i_1 $(0, 0, ..., 0)$ and i_2 $(1, 0, ..., 0)$ will have zero distance.

Furthermore, we proceed to prove that condition (4), the triangle inequality, will not hold when $0 < m < p$:

First, when $p/2 \leq m < p$, consider the following points:

$$A : (1, ..., 1, \underbrace{0, ..., 0}_{m}),$$

$$B : (\underbrace{0, ..., 0}_{m}, 1, ..., 1)$$

and

$$C : (0, ..., 0).$$

It is easy to see that $d_{DPF}(A, C) = 0$, and also $d_{DPF}(B, C) = 0$. However, $d_{DPF}(A, B) = (p - m)^{\frac{1}{r}}$, thus $d_{DPF}(A, B) > d_{DPF}(A, C) + d_{DPF}(B, C)$. Therefore, the triangle inequality does not hold.

On the other hand, when $0 < m < p/2$, consider the following points:

$$A : (1, ..., 1),$$

$$B : (0, ..., 0)$$

and

$$C : (0, ..., 0, \underbrace{1, ..., 1}_{\lfloor p/2 \rfloor}).$$

we have $d_{DPF}(A, C) = d_{DPF}(B, C) = 0$, while $d_{DPF}(A, B) = (m)^{\frac{1}{r}}$. Again, $d_{DPF}(A, B) > d_{DPF}(A, C) + d_{DPF}(B, C)$. Hence, the triangle inequality does not hold too.

So for any $0 < m < p$, triangle inequality does not hold for d_{DPF}.

To conclude, it is proven that for any $0 < m < p$, condition (2) and (4) does not hold for d_{DPF}. And thus d_{DPF} is a non-metric distance function when $0 < m < p$. Besides, when $m = 0$, condition (2) does not hold. In this case, DPF is also non-metric. □

Appendix B: CLARANS Algorithm

CLARANS first chooses k seeding objects as medoids and then proceeds to associate other objects with the medoid closest to them to form clusters. The goodness of a set of k medoids is measured by the average minimal distance from each object to its medoid. We can view the process of finding k medoids as searching through a graph (denoted by $G_{n,k}$) of all possible combinations of finding k objects given a total of n objects. A node in the graph represents a set of k objects $O_{m_1}, ..., O_{m_k}$ where O_{m_j} is a selected medoid. Two nodes are neighbors if and only if they differ by a single object.

In the algorithm, CLARANS will search for *numlocal* local minimals in the graph starting with a random node (Step 2). In each iteration, it tests a number of its neighbors (*maxneighbor*) and check if the neighbor yields a lower cost. If it does, the current node is moved to this neighbor (Step 4 and 5). A local minimal is found when none of the neighbors yield a lower cost. Next, the costs of all the local minimals are compared, and the one that yields the lowest cost is selected as the medoid set to generate clusters (Step 7 and 8).

The formal description of the CLARANS algorithm is as follows:

- **Inputs**: *numlocal*, *maxneighbor*, $G_{n,k}$;
- **Output**: *bestnode*;
- **Variables**:
  mincost, minimum cost of clustering;
  current, current node to consider from the graph $G_{n,k}$;
  S, randomly selected neighbor of *current*;
  i, j, counters;
- **Algorithm CLARANS**:
  1: Initialization:
    - $i \leftarrow 1$;
    - $mincost \leftarrow MAX\_DOUBLE$;
  2: $current \leftarrow arbitrary\_node(G_{n,k})$;
  3: $j \leftarrow 1$;
  4: $S \leftarrow random\_neighbor(current)$;
  5: if ($Cost(S) < Cost(current)$) then
    $current \leftarrow S$;
    goto step 3;
  6: else $j \leftarrow (j + 1)$;
  7: if ($j \leq maxneighbor$) goto step 4;
  8: else if ($Cost(current) < mincost$)
    $mincost \leftarrow Cost(current)$;
    $bestnode \leftarrow current$;
  9: $i \leftarrow (i + 1)$;
  10: while ($i < numlocal$) goto step 2;