Similarity Search with Implicit Object Features

Yi Luo[1] *Zheng Liu*[1] *Xuemin Lin*[1] *Wei Wang*[1] *Jeffrey Xu Yu*[2]

[1] The University of News South Wales, Sydney, Australia
{luoyi, zliu, lxue, weiw}@cse.unsw.edu.au
[2] The Chinese University of Hong Kong, Hong Kong, China, yu@se.cuhk.edu.hk

Abstract. Driven by many real applications, in this paper we study the problem of similarity search with implicit object features; that is, the features of each object are not pre-computed/evaluated. As the existing similarity search techniques are not applicable, a novel and efficient algorithm is developed in this paper to approach the problem. The R-tree based algorithm consists of two steps: feature evaluation and similarity search. Our performance evaluation demonstrates that the algorithm is very efficient for large spatial datasets.

1 Introduction

Similarity search is fundamental to many applications involving spatial data analysis. Many research results [1, 4, 6, 8, 7, 10] have been published in the last decade, where the most popular similarity model is based on a feature vector for each data object. In such a model, each data object, available for similarity search, is represented as a vector, and the similarity between objects is measured by the distance between the vectors. Such applications include image similarity retrieval [4, 10], shape similarity search [6, 8] and similarity search on spatio-temporal trajectories [1, 7].

KNN (the k-nearest neighbor search) is one of the most important similarity search queries. For a query object q and a query parameter k, KNN is to find the k objects that are most similar to q [5, 11].

Consider that in many applications, objects for similarity search are not pre-defined; consequently, the feature vector for each object is not pre-computed and stored in a database.

For instance, ornithologists may want to identify similar bird communities for selecting a future research target or for behavior predication. A cluster of bird nests is an object. In the application, nest positions are changing regularly and definition of a cluster may vary from time to time because of difference research orientation. Feature groups are represented as groups of polygons. For example, the open water map is a feature group, including lakes, rivers and springs as polygons. Other feature groups are the vegetation map including forests of specific vegetation, the predator distribution map including communities of predatory birds, and man-made structure map including towns, high ways and villages. Moreover, maps of rainfall precipitation and temperature should also be considered; but in these contour maps, each value range could correspond to a feature group. In the application, a cluster of bird nests can be evaluated based on the distances to the nearest feature in each feature group, such as the nearest open water place and the nearest town. Figure 1 illustrates a cluster of nests and a nearby lake represented as a feature polygon.

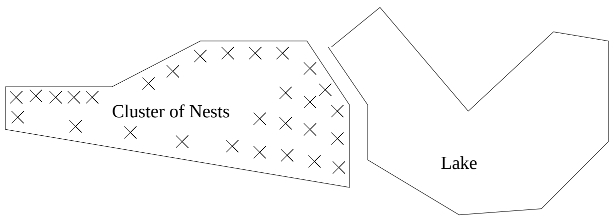


Fig. 1. Ornithology Study

Similar applications lie in road traffic analysis, urban development, crime analysis, etc.

Motivated by the above applications, in this paper we study the problem of a non-conventional KNN, where the feature vector of an object is not pre-computed, namely SSIOF (Similarity Search with Implicit Object Features). In particular, we study the KNN problem where each object is a set of points in 2-dimensional space, and each object is evaluated against d groups of features to obtain a d-dimensional feature vector.

By effectively characterizing the results' properties, we develop an efficient and novel R-tree based algorithm to evaluate features of each object. Then, an effective filtering technique is developed to prune away objects (clusters) as many as possible before a precise computation. These are the contributions of the paper. Our performance study demonstrates that our techniques are very efficient to process large spatial datasets.

The remaining paper is organized as follows. Section 2 presents the preliminaries. Section 3 and 4 presents our algorithms and the analysis. Experiment results are reported in section 5. This is followed by conclusions.

2 Preliminaries

In this section, we start with formally defining the problem and then introduce some necessary background.

2.1 Statement

In a 2-dimensional space, given n clusters $C_1, C_2, \ldots, C_n$ and d categories/groups of features $\pi_1, \pi_2, \ldots, \pi_d$. Each cluster C_i is a set of points and each feature is a polygon. We use F_j to denote a feature and pt as a point.

Suppose the distance between a cluster C_i and a feature(polygon) F_j, denoted as $d(C_i, F_j)$, is defined as the average Euclidean distance from each point pt in C to the polygon. Here, the distance between a point and a feature $dist(pt, F)$

is the minimum distance between the point and the edges of the feature. The aggregational *feature evaluation* of a cluster C_i with respect to a feature category π_k is the distance from C_i to its nearest polygon in π_k, denoted as $\phi(C_i, \pi_k)$. The problem of Similarity Search in Implicit Feature Space (SSIOF) is to find k most similar clusters to the given cluster C_0 based on the following similarity measure:

$$Sim(C_i, C_0) = \|(\phi(C_i, \pi_1), \ldots, \phi(C_i, \pi_d)), (\phi(C_0, \pi_1), \ldots, \phi(C_i, \pi_d))\|_f \qquad (1)$$

$\|.\|_f$ is Euclidean or Manhattan distance function; we use the Manhattan distance in our paper.

2.2 *R*-tree index

R-tree is a widely used index for spatial objects based on B^+-trees, which organises geometric objects by recursively grouping neighbouring objects and representing them by minimum bounding rectangles(MBRs). A node of *R*-tree corresponds to a disk page. An intermediate node maintains a set of MBRs and pointers which represent the children nodes, while a leaf node contains a set of spatial objects with their positions in the database. Fig. 2 shows an instance of R-tree.

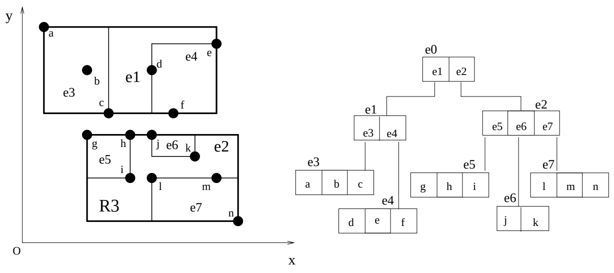


Fig. 2. R-Tree Example

In this paper, we choose one of the most popular variations R^*-tree to index each feature categories and perform our evaluations. Each polygon is represented by its MBR first, then those MBRs is indexed by R^*-tree.

3 Evaluation and Search (ES) Algorithm

Our proposed algorithm *ES* for solving the SSIOF problem contains two major steps:

1) **Feature Evaluation:** in this step, we try to find all possible feature candidates for each pair of cluster and feature group.
2) **Similarity Search:** this step is to compute the k most similar clusters to the query C_0, based on the candidates outputted in the previous step.

3.1 Feature Evaluation

Let N_{C_i} be the MBR of a cluster C_i with 4 edges r_1, r_2, r_3 and r_4; and N_{F_j} be the MBR of a feature polygon F_j with 4 edges s_1, s_2, s_3 and s_4. We assume that N_{C_i} and N_{F_j} do not overlap. We will first define some useful metrics between MBR's for later discussion.

$L(r_k, s_l)$ denotes the minimum distance between two points falling on r_k and s_l, and $U(r_k, s_l)$ denotes the maximum distance between two points falling on r_k and s_l[2]. Thus the minimum of distance between two points contained in N_{C_i} and N_{F_j} can be expressed as:

$$minL(N_{C_i}, N_{F_j}) = \min\{L(r_k, s_l)\} \tag{2}$$

Similarly, we have:

$$minU(N_{C_i}, N_{F_j}) = \min\{U(r_k, s_l)\} \tag{3}$$

We also define the following distance. For the cluster C and polygon F,

$$maxminU(N_{C_i}, N_{F_j}) = \max_k\{\min_l\{U(r_k, s_l)\}\} \tag{4}$$

Figure 3 shows the different metrics.

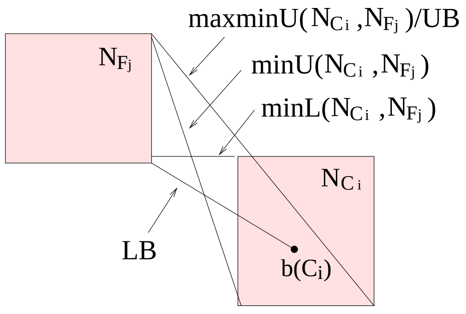


Fig. 3. Distance Matrices

Pruning With the Lower and Upper Bounds

In the SSIOF problem, the similarity isn't measured between points but clusters and polygons, so it's too expensive to compute precise distances on pairs of clusters and features, which makes it necessary to use relatively tight lower and upper bounds for pruning. Then precise distances could be computed only on a small set of clusters and features.

Consider a cluster C_i bounded by MBR N_{C_i} and a feature F_j in MBR N_{F_j}, the lower and upper bounds of the distance between these two are :

$$d_{LB}(C_i, F_j) = L(b(C_i), N_{F_j}) \tag{5}$$

and

$$d_{UB}(C_i, F_j) = maxminU(N_{C_i}, N_{F_j}). \tag{6}$$

$b(C_i)$ in the Equation 5 is the centroid of the cluster C_i computed by the average coordinates of all points in C_i on each dimension. We use $L(b(C_i), N_{F_j})$ to denote the minimal distance between $b(C_i)$ and a point in rectangle $N(F_j)$. The lower and upper bounds are illustrated in Figure 3 as LB and UB respectively.

The correctness of lower bound is proved in [9]. By definition, $d(C, F)$ is the average of $dist(pt, F)$ for all $pt \in C$. Based on the inequality:

$$\sum_{i=1}^{K} \sqrt{x_i^2 + y_i^2} \geq \sqrt{(\sum_{i=1}^{K} x_i)^2 + (\sum_{i=1}^{K} y_i)^2}$$

$d(C, F)$ is no less than the distance from $b(C)$ to some points inside N_C, which is no less than $L(b(C), N_F)$.

Lemma 1 shows the correctness of the upper bounds.

Lemma 1. *For a cluster C_i in MBR N_{C_i} and a feature F_j in MBR N_{F_j}, the upper bound of $d(C_i, F_j)$ is $maxminU(N_{C_i}, N_{F_j})$.*

Proof. Suppose that N_{F_j} is bounded by s_l $(l = 1..4)$. Since N_{F_j} is the minimal bound rectangle of F_j, there must be at least a point of F_j on each s_l. Thus the upper bound of $dist(pt, F_j)$ equals $\min_l U(pt, s_l)$. Consider all points on N_{C_i}, the upper bound of $d(C_i, F_j)$ is $\max_{pt \in C_i} dist(pt, F_j)$, which is no larger than $maxminU(N_{C_i}, N_{F_j})$. □

When N_{F_j} and N_{C_i} overlaps, it can be immediately verified that the above bounds hold. When a feature group is indexed by an R-tree, the lemma still holds if we change N_{F_j} to the MBR of an R-tree node. This gives us the opportunity to prune features while traversing the index.

R-tree Based Pruning

Making use of the index on each feature group could speed up the process of feature evaluation. Next we will introduce the pruning technique for a feature group π_k indexed by an R-tree T_{π_k}, as shown in Algorithm 1. Each node of T_{π_k} corresponds to a disk page. To lower the disk I/O cost, we traverse T_{π_k} using the following strategy which allow us to visit each R-tree node at most once.

The goal is to find a set of candidate features for each cluster. For each cluster C_i, we maintain a candidate list $L(C_i, \pi_k)$, implemented as a heap. Each list entry e is either the MBR of an non-leaf R-tree node or the MBR of a feature polygon, corresponding the intermediate levels and the leaf level in the R-tree. As mentioned above, the lower and upper bounds hold on both kinds of MBRs, denoted as $d_{LB}(C_i, e)$ and $d_{UB}(C_i, e)$. For any pair of entries in the list, their bounds overlap. $\phi_{LB}(C_i, \pi_k)$, $\phi_{UB}(C_i, \pi_k)$ and $q(C_i, \pi_k)$ are used to record the minimum of $d_{LB}(C_i, e)$, the minimum of $d_{UB}(C_i, e)$ and the maximum of $d_{LB}(C_i, e)$ for each list, respectively. $\phi_{LB}(C_i, \pi_k) = \phi_{UB}(C_i, \pi_k) = \infty$ and $q(C_i, \pi_k) = 0$ initially.

At the beginning of Algorithm 1, we assume the root of T_{π_k} is a candidate for all clusters, and insert it in all lists. In each iteration from Line 2 to Line 10 in

Algorithm 1 Feature Evaluation

```
Input: clusters $C_i$ $(i = 0..n)$, $R$-tree of $\pi_k$ $T_{\pi_k}$.
Output: $\phi_{LB}(C_i, \pi_k)$, $\phi_{UB}(C_i, \pi_k)$, feature list $L(C_i, \pi_k)$.
Description:
 1: repeat
 2:   for each cluster $C_i$ do
 3:     let $e$ be the non-leaf entry in $L(C_i, \pi_k)$ with minimal $d_{LB}(C_i, e)$;
 4:     for each cluster $C_j$ containing $e$ do
 5:       replace $e$ with its children in $T_{\pi_k}$;
 6:       remove entries $e_r$ if $d_{LB}(C_j, e_r) \geq \phi_{UB}(C_j, \pi_k)$;
 7:       update $\phi_{LB}(C_j, \pi_k)$, $\phi_{UB}(C_j, \pi_k)$, $q(C_j, \pi_k)$;
 8:       if $q(C_j, \pi_k) \geq \phi_{UB}(C_j, \pi_k)$ then
 9:         remove entries $e_r$ if $d_{LB}(C_j, e_r) \geq \phi_{UB}(C_j, \pi_k)$;
10:         update $q(C_j, \pi_k)$;
11: until entries in $L(C_i, \pi_k)$ for all $i$ are leaf entries
```

Algorithm 1, the lists are visited in a round-robin fashion. A non-leaf entry with the minimum lower bound $d_{LB}(C_i, e)$ is selected for the current list. Here, a non-leaf entry means the corresponding R-tree node is not a leaf node. We replace it by its children in the R-tree in all lists. A child e_r is inserted into C_j's list, when its low bound $d_{LB}(C_j, e)$ isn't less than $\phi_{UB}(C_j, \pi_k)$, the minimum upper bound of all entries in the list. After updating $\phi_{LB}(C_j, \pi_k)$, $\phi_{UB}(C_j, \pi_k)$ and $q(C_j, \pi_k)$, we verify the list and filter those entries whose lower bounds $d_{LB}(C_j, e)$ is greater than the updated $\phi_{UB}(C_j, \pi_k)$. This verification could be skipped when $q(C_i, \pi_k)$ is between $\phi_{LB}(C_i, \pi_k)$ and $\phi_{UB}(C_i, \pi_k)$. We repeat these steps until there is not any non-leaf entry in all lists.

3.2 Similarity Search

In this section, we will discuss how to compute the exact distances between pairs of clusters and feature groups based on the generated candidate features for answering the SSIOF queries. Our goal is the find the cluster most similar to the query C_0 while minimising the computation complexity.

Algorithm 2 presents the overview of the similarity search step.

The input parameter $L(C_i, \pi_k)$ is the candidate list for cluster C_i and feature group π_k. Function *ComputeExact*(C_i, π_k) in Line 1 and 5 computes the exact distance between C_i and feature group π_k. C_{min} is the cluster with minimal $Sim_{LB}(C_i, C_0)$. $Sim_{LB}(C_i, C_0)$ and $Sim_{UB}(C_i, C_0)$ denote the lower and upper bound of similarity between C_i and C_0, as computed from the input as follows.

Firstly $\phi(C_0, \pi_k)$ are pre-computed for all feature groups. Suppose $f_L = \phi_{LB}(C_i, \pi_k) - \phi(C_0, \pi_k)$ and $f_U = \phi_{UB}(C_i, \pi_k) - \phi(C_0, \pi_k)$, then we have

$$LB_{i,j} = \begin{cases} 0 & f_L \times f_U < 0 \\ min(|f_L|, |f_U|) & otherwise \end{cases}$$

and

$$UB_{i,j} = max(|f_L|, |f_U|).$$

Algorithm 2 Similarity Search

Input: $L(C_i, \pi_k)$ for $i = 0..n, j = 1..d$, cluster set $\{C_i(i = 0..n)\}$
Output: The cluster C_i with minimal $Sim(C_i, C_0)$ $(i \neq 0)$.
Description:

```
1: $ComputeExact(C_0, \pi_k)$ for all $j$; remove $C_0$ from cluster set;
2: $result = \infty$;
3: while $Sim_{LB}(C_{min}, C_0) \leq result$ do
4:    for all feature groups $\pi_k$ do
5:       $ComputeExact(C_{min}, \pi_k)$;
6:       if $Sim_{LB}(C_{min}, C_0) > result$ then
7:          break; //from FOR
8:    $result = min\{result, Sim(C_{min}, C_0)\}$;
9:    remove $C_{min}$ from cluster set;
10: return all clusters $C_i$ with $Sim(C_i, C_0) = result$.
```

Thus,

$$Sim_{LB}(C_i, C_0) = \sum_j LB_{i,j}$$

$$Sim_{UB}(C_i, C_0) = \sum_j UB_{i,j}$$

For example, the results of the feature evaluation step, including the lower bound ($\phi_{LB}(C_i, \pi_k)$) and upper bound $\phi_{UB}(C_i, \pi_k)$ are stored in a 2-dimensional array as shown in Figure 4. The initial bound of similarity between C_i and C_0 are computed as shown on the last column.

	π_1	π_2	π_3	$Sim(C_i, C_0)$
C_0	6	2	9	$[0, 0]$
C_1	$[3, 4]$	$[11, 12]$	$[8, 9]$	$[11, 14]$
C_2	$[8, 12]$	$[1, 11]$	$[9, 14]$	$[2, 20]$
C_3	$[4, 7]$	$[2, 4]$	$[10, 10]$	$[1, 5]$

Fig. 4. Similarity Evaluation

The clusters are sorted on lower bound of $Sim(C_i, C_0)$ and iteratively computed for precise similarity, until the next lower bound is larger than an already-found result. This sequence can minimise the number of clusters that is precisely computed. Also in Line 6, after each calling of $ComputeExact$, the current lower bound of similarity is refined using the exact distance returned from the function, and is compared with the result, which greatly reduce the number of feature groups need to be computed.

For the example in Fig. 4, the cluster C_3 is first chosen since lower bound of $Sim(C_3, C_0)$ is the minimal in all clusters. Suppose its precise similarity is 3. The next cluster is C_2. After calling $ComputeExact(C_2, \pi_1)$, assume the lower bound of $Sim(C_2, C_0)$ is updated to be 4, which is larger than the current result 3. As a result, C_2 is dropped as it can not be the result. Also the lower bound of $Sim(C_1, C_0)$ is larger than the current result 2, and C_1 is eliminated as well and the final result C_3 is returned.

Lemma 2. *Algorithm 2 gives the correct answer to the similarity search query.*

Proof. Consider the case that Algorithm 2 returns C_i as result and the exact answer is C_j where $i \neq j$. This is impossible since after C_i is precisely computed, the lower bound of $Sim(C_j, C_0)$ must be smaller than $Sim(C_i, C_0)$, in consequence, C_j is chosen to be precisely computed and C_j should be returned instead of C_i. □

Also, it is easy to see that Algorithm 2 minimize the number of chosen clusters. Suppose that the cluster returned is C_r with result r, and there exists an algorithm A which minimizes the number of chosen clusters. In algorithm A, a cluster C_i such that lower bound of $Sim(C_i, C_0)$ is larger than r must not be visited while all other clusters must be considered for precise computation. This is exactly the case of Algorithm 2. For a cluster C_i that $Sim_{LB}(C_i, C_0) > r$, $Sim_{LB}(C_i, C_0) > Sim_{LB}(C_r, C_0)$. Thus in Algorithm 2, C_r is chosen before C_i. After processing C_r, *result* is updated to r and C_i are dropped.

Edge Pruning

ComputeExact(C_i,π_k) is used to compute the exact distances between a cluster C_i and a feature group π_k. It need to calculate all the distance between the points in C_i and the candidate features in every feature groups. The brute-force way is to compute the distance between a point and every edge in some feature and choose the minimum one as the distance of the point to the feature. We proposed some techniques that can avoid useless computations and save much more time than the brute-force way.

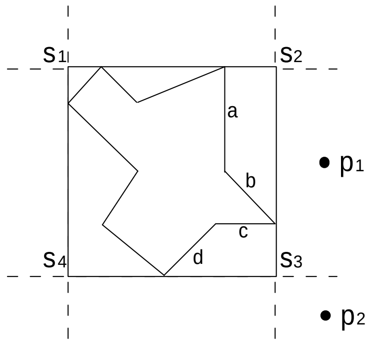


Fig. 5. Edge Pruning

The optimisation comes from reducing feature edges need to be computed. As shown in Figure 5, the rectangle is the minimum bounding rectangle of a certain feature. By extending the four edges of the MBR, we partition the whole space into 8 areas expect the MBR itself. $s_1, s_2, s_3 and s_4$ are vertices of the MBR and a, b, c and d are four edges on the feature. p_1 and p_2 are points belonging to some cluster.

Take p_1 as an example. It need calculating all the distance between p_1 and all edges of the feature in the brute-force way. In fact, we can found that the minimum distance from p_1 to the feature must be the minimum distance of p_1 to one of the four edges a, b, c and d. In case of p_2, the minimum distance from p_2 to the feature must be the minimum distance of p_2 to one of the two edges c and d.

To formalise, if the project of a point p_k to the closest edge $s_i s_j$ of the MBR falls in the edge, then we only need to compute such kind of edges that $s_i s_j$ can be project on. If not, suppose the nearest vertex of MBR to p is s_i, only the edges that s_i can be project on are computed.

To further reduce the time complexity, edge projections of a feature are computed at most once and then stored in memory for all other points. Also, when the MBR of a cluster is wholly contained in one of the 8 areas, it is not necessary to check the position of each point any more.

Extend to *k*-clusters

The above algorithm is extended to return the k clusters which are most similar to the given cluster C_0. In Line 8 of Algorithm 2, variant *result* should be set to the k-th lowest similarity,and k most similar clusters are returned in Line 13.

4 Discussion

As mentioned in the above section, for a node on the R^*-tree, we visit it at most once. In each step, the node to be visited is chosen considering only one cluster while ignoring the preference of other clusters. This searching strategy is based on an assumption that the number of clusters is relatively small, since the strategy sacrifices local optimization for each cluster to achieve a better global I/O cost. Since reading disk is much more costly than in-memory computation, our algorithm works well when the number of clusters is not too large.

For the case that the number of clusters is so large that the sacrifice of local computation is unbearable, we can use following divide-and-conquer strategy which is similar to the Nested Loops Join. We first partition the clusters into several parts by grouping near clusters. Then we use our proposed algorithm on each part of clusters. In this case, if there are n groups of clusters, each node of an R^*-tree is visited for at most n times.

5 Experiments

We implemented our proposed *ES* algorithm and evaluate its performance on synthetic data. We use the algorithm *CPM* (Compute Proximity Matching) as a benchmark based on [9]. The algorithm *CPM* solves a problem that is similar to our problem assuming the number of feature polygons is relatively small and there is no spatial index built on the features. It reads the relevant clusters into buffer first, then read features batch by batch into buffer and determine their groups. For each cluster C_i and feature group π_j, it computes the approximate distance between C_i and each feature in π_j for filtering out features that are too far from C_i. Maintain a list of candidate features for computing $\phi(C_i, \pi_j)$. Then it computes the approximate similarity for each cluster and filter out clusters that are not the solution. Finally it calculates the exact similarities to the remaining clusters and their associate features, and return the query result.

Suppose the number of cluster is n and the number of features is m. Feature number is the same in each of the g features groups. The number of points in each cluster is nc and nf gives the number of edges in each feature polygon. In the experiments, average nc is 100 and average nf is 15.

To generate data, we firstly generate $m + n$ rectangles that are uniformly distributed in the 2-dimensional space. The size of rectangles are randomly chosen within a limited range. Number of features in each group is determined such that the summary is m. Rectangles corresponding to the clusters or a features group do not intersect with each other. In each of n rectangles, nc points are uniformly generated, based on which the MBRs are computed. This gives the nc clusters. In each of the remaining rectangles, nf points are randomly generated. To generate a simple polygon which is linked by the nf points. We will apply a Graham's scan-like algorithm [3].

We use $R*$-trees, a variant of R-tree, to index the feature groups. Two algorithms CPM and ES are implemented using C++, Experiments are run on a Linux machine with 1.8G P4 CPU and 512M memory. For each dataset, we process extensive queries and get the average result.

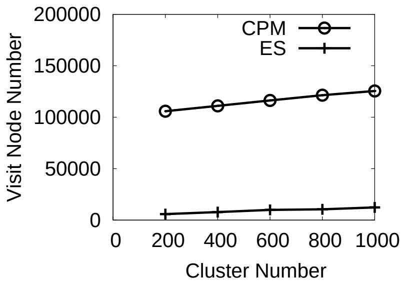
5.1 Scalability Comparison

We first compare the algorithms with different number of clusters, ranging from 200 to 1000. 100000 features are clustered in 10 groups. The experiment results are shown in Figure 6.

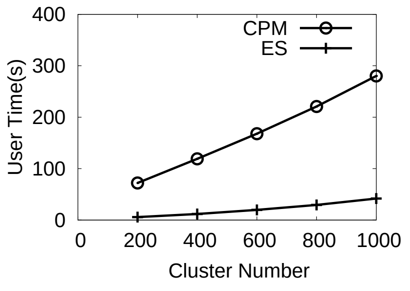
The first sub-figure compares the I/O cost, which is the summary of the number of pages that corresponding to $R*$-tree index and features. CPM does not use index, but reads a large amount of features; ES reads a small number of index pages in the first step and a few features in the second step. It is clear that the I/O cost of ES is much smaller than CPM.

User time for precessing a query is compared in the second sub-figure of Fig. 6. For 1000 clusters, ES responds in about 42 seconds while CPM needs nearly 5 minutes to get the result.

We also study the performance of both algorithms with different number of features. Feature number varies from 5000 to 100000. There are 10 feature groups and the number of clusters is set to 200. Results are shown in Fig. 6. Similar with the previous experiments, the first sub-figure shows the I/O cost while the second compares the precessing time.



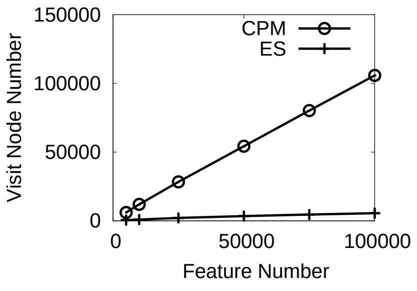
(a) I/O



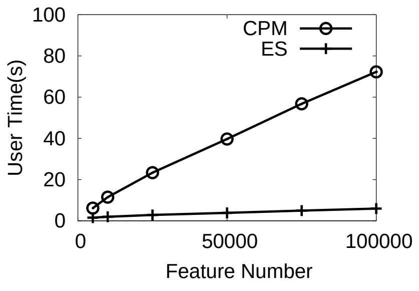
(b) Time

Fig. 6. Compare Cluster Number

With 100000 features, our algorithm processes a query in 6 seconds and less than 6000 disk pages read in memory, compared with large I/O cost and more than 1 minute processing time of CPM.



(a) I/O



(b) Time

Fig. 7. Compare Feature Number

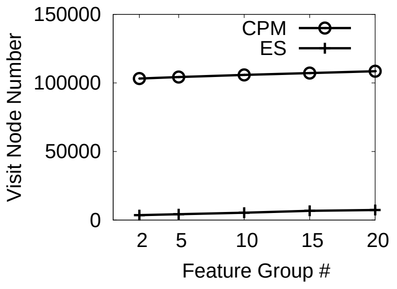
5.2 Dimensionality Comparison

We evaluate our algorithm with different dimensionality of feature space. The number of feature group varies from 2 to 20. 100000 features are categorised to the feature groups and number of clusters is 200. Fig. 8 shows the I/O cost and user time of the two algorithms, which demonstrates the large performance difference between the two algorithms.

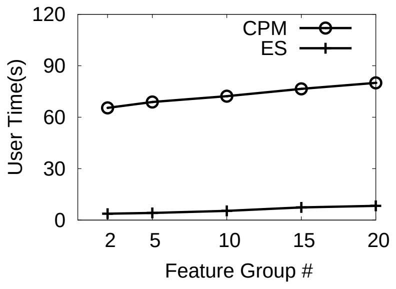
6 Conclusions

In this paper, a similarity search problem which is based on an implicit feature space is investigated. By making use of the spatial indexes like R-trees built on the feature categories, we present an effective algorithm for the queries, which consists two steps: feature evaluation and similarity search. Experiments show the efficiency of the algorithm on all cases.

For the future work, we will investigate the problem of similarity join, which joins a set of clusters to itself, with respect of d different categories of features.



(a) I/O



(b) Time

Fig. 8. Compare Feature Group Number

References

1. Yuhan Cai and Raymond Ng. Indexing spatio-temporal trajectories with chebyshev polynomials. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 599–610, New York, NY, USA, 2004. ACM Press.
2. Antonio Corral, Yannis Manolopoulos, Yannis Theodoridis, and Michael Vassilakopoulos. Closest pair queries in spatial databases. In *SIGMOD '00: Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 189–200, New York, NY, USA, 2000. ACM Press.
3. M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry - Algorithm and Applications.* Springer-Verlag, Berlin, 1997.
4. Christos Faloutsos, Ron Barber, Myron Flickner, Jim Hafner, Wayne Niblack, Dragutin Petkovic, and William Equitz. Efficient and effective querying by image content. *Journal of Intelligent Information Systems*, 3(3/4):231–262, 1994.
5. Gísli. R. Hjaltason and Hanan Samet. Distance browsing in spatial databases. volume 24, pages 265–318, New York, NY, USA, 1999. ACM Press.
6. H. V. Jagadish. A retrieval technique for similar shapes. In *SIGMOD '91: Proceedings of the 1991 ACM SIGMOD international conference on Management of data*, pages 208–217, New York, NY, USA, 1991. ACM Press.
7. Tamer Kahveci, Ambuj K. Singh, and Aliekber Gürel. Similarity searching for multi-attribute sequences. In *SSDBM '02: Proceedings of the 14th International Conference on Scientific and Statistical Database Management*, page 175, Washington, DC, USA, 2002. IEEE Computer Society.
8. Hans-Peter Kriegel, Stefan Brecheisen, Peer Kröger, Martin Pfeifle, and Matthias Schubert. Using sets of feature vectors for similarity search on voxelized CAD objects. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 587–598. ACM Press, 2003.
9. Xuemin Lin, Xiaomei Zhou, and Chengfei Liu. Efficient computation of a proximity matching in spatial databases. *Data Knowledge Engineering*, 33(1):85–102, 2000.
10. Apostol Natsev, Rajeev Rastogi, and Kyuseok Shim. WALRUS: a similarity retrieval algorithm for image databases. pages 395–406, 1999.
11. Nick Roussopoulos, Stephen Kelley, and Frédéric Vincent. Nearest neighbor queries. In *SIGMOD '95: Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, pages 71–79, New York, NY, USA, 1995. ACM Press.