An Approach to Nearest Neighboring Search for Multi-dimensional Data

Yong Shi
Department of Computer Science and Information Systems Kennesaw State University 1000 Chastain Road Kennesaw, GA 30144 yshi5@kennesaw.edu

Li Zhang
Department of Computer Science Eastern Michigan University Ypsilanti, MI 48197 li.zhang@emich.edu

Lei Zhu
Department of Information Technology Clayton State University 2000 Clayton Stat Blvd. Morrow, GA 30260 leizhu@clayton.edu

Abstract

Finding nearest neighbors in large multi-dimensional data has always been one of the research interests in data mining field. In this paper, we present our continuous research on similarity search problems. Previously we have worked on exploring the meaning of K nearest neighbors from a new perspective in PanKNN [20]. It redefines the distances between data points and a given query point Q, efficiently and effectively selecting data points which are closest to Q. It can be applied in various data mining fields. A large amount of real data sets have irrelevant or obstacle information which greatly affects the effectiveness and efficiency of finding nearest neighbors for a given query data point. In this paper, we present our approach to solving the similarity search problem in the presence of obstacles. We apply the concept of obstacle points and process the similarity search problems in a different way. This approach can assist to improve the performance of existing data analysis approaches.

Keywords: *K-nearest search, multi-dimensional data, obstacles*

1. Introduction

Huge amount of data have been generated in many disciplines nowadays. The similarity search problem has been studied in the last decade, and many algorithms haves been proposed to solve the K nearest neighbor search [15, 19, 2, 14, 11]. We previously proposed PanKNN [20] which is a novel technique that explores the meaning of K nearest neighbors from a new perspective. It redefines the distances between data points and a given query point Q, and selects data points which are closest to Q efficiently and effectively. In this paper, we first give a brief introduction about our previous work on PanKNN and discuss the Fuzzy concept; then, we propose to use the Fuzzy concept to design OPanKNN algorithm that targets solving the nearest neighbors problems in the presence of obstacles.

2. Related work

The similarity between two data points used to be based on a similarity function such as Euclidean distance which aggregates the difference between each dimension of the two data

points in traditional nearest neighbor problems. In those applications, the nearest neighbor problems are solved based on the distance between the data point and the query point over a fixed set of dimensions (features). However, such approaches only focus on full similarities, i.e., the similarity in full data space of the data set. Also early methods [1, 8, 23] suffer from the "curse of dimensionality". In a high dimensional space the data are usually sparse, and widely used distance metric such as Euclidean distance may not work well as dimensionality goes higher. Recent research [9] shows that in high dimensions nearest neighbor queries become unstable: the difference of the distances of farthest and nearest points to some query point does not increase as fast as the minimum of the two, thus the distance between two data points in high dimensionality is less meaningful. Some approaches [16, 4, 3] are proposed targeting partial similarities. However, they have limitations such as the requirement of the fixed subset of dimensions, or fixed number of dimensions as the input parameter(s) for the algorithms.

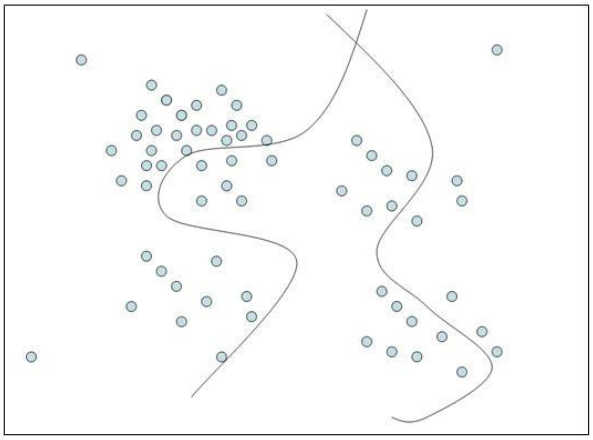

Figure 1: A Data Set with Obstacles

There are quite a few approaches designed to detect clusters in the presence of obstacles and facilitators. For example, COD CLARANS [6] is modified version of the CLARANS [18] partitioning algorithm which performs clustering processes in the presence of obstacles. AUTOCLUST+ [13] is version of AUTOCLUST[12] enhanced to handle obstacles, which does not require parameters. DBRS+ [25] is derived from DBRS [22], and it handles both obstacles and facilitators.

However, none of these algorithms considers detecting outliers simultaneously with clustering process. In many cases, outliers are as important as clusters, such as credit card fraud detection, discovery of criminal activities, discovery of computer intrusion, and etc. Analyzing the data distribution with the consideration of obstacles is critical for many data sets. For example, figure 1 shows two dimensional data set where there are two curves (obstacles) that cut through the data set, separating it into isolated subgroups, some of which would have been in the same clusters have these two curves not existed.

3. Fuzzy Concept

Various data sets in the real world are not naturally well organized and fuzzy concept can be applied to further improve the data analysis approaches. The concept of fuzzy sets was first

introduced by Zadeh [24] to represent vagueness. The use of fuzzy set theory is becoming popular because it produces not only crisp decision when necessary but also corresponding degree of membership. Usually, membership functions are defined based on a distance function, such that membership degrees express proximities of entities to cluster centers. In conventional clustering, sample is either assigned to or not assigned to group. Assigning each data point to exactly one cluster often causes problems, because in real world problems crisp separation of clusters is rarely possible due to overlapping of classes. Also there are exceptions which cannot be suitably assigned to any cluster. Fuzzy sets extend to clustering in that objects of the data set may be fractionally assigned to multiple clusters, that is, each point of data set belongs to groups by membership function. This allows for ambiguity in the data and yields detailed information about the structure of the data, and the algorithms adapt to noisy data and classes that are not well separated. Most fuzzy cluster analysis methods optimize subjective function that evaluates given fuzzy assignment of data to clusters.

One of the classic fuzzy clustering approaches is the Fuzzy C-means Method designed by Bezdek, J. C [10]. In brief, for data set X with size of n and cluster number of c, it extends the classical within groups sum of squared error objective function to fuzzy version by minimizing the objective function with weighting exponent m, $1 \leq m < \infty$:

$$J_m(U,V) = \sum^{n}_{k=1}\sum^{c}_{i=1} (u_{ik})^m d^2(x_k,v_i), \qquad (1)$$

where U is partition of X in c part, V = v = v (v_1,v_2,...,v_c) are the cluster centers in R^p , and A is any (p×p) symmetric positive definite matrix defined as the following:

$$d(x_k,v_i)=((x_k - v_i)^T(x_k - v_i))^{1/2}, \qquad (2)$$

where $d(x_k,v_i)$is an inner product induced norm on R^p, u_{ik} is referred to as the grade of membership of x_k to the cluster i.

The fuzzy C-Means (FCM) uses an iterative optimization of the objective function, based on the weighted similarity measure between x_k and the cluster center v_i. During each iteration, it calculates the *c* cluster centers $\{v_{i,t}\}$,i =1,...,c

$$v_{i,t} = \sum^{n}_{k=1} u^{m}_{ik,t-1}\, x_k / \sum^{n}_{k=1} u^{m}_{ik,t-1}\,, \qquad (3)$$

for those data points not of any current cluster center, it calculate the following

$$(u_{ik,t})^{-1} = \sum^{c}_{j=1} (d_{ik,t} / d_{jk,t})^{(2/m-1)}, \qquad (4)$$

When a predefined termination condition is satisfied, the algorithm is terminated.

4. Solving Similarity Problem

We will briefly introduce our previous work on PanKNN [20] in this section. PanKNN is a novel approach in which we analyze the nearest neighbor problems from a new perspective. We define the new meaning for the K nearest neighbor problem, and design algorithms accordingly. The similarity between data point and query point is not based on the difference aggregation on all the dimensions. We propose self-adaptive strategies to dynamically select dimensions based on the different situations of the comparison.

Consider query point Q(1,1,1,1,1) and two data points X_1 (2,3,8,10000,10000) and X_2 (50,50,50,50,50) in 5 dimensional data space, with D_i, i=1, 2, ..., 5 representing each dimension, respectively. Which data point is closer to Q? If we use the tradition Euclidean distance, the conclusion is that X_2 is closer to Q than X_1 is in the full data space. However, if we take a closer look at the first three dimensions, we can easily find that X_1 is much closer to than X_2 in the subspace of those dimensions. This example illustrates why we not only need to consider how close data point is to the query point, but also need to consider which and how many dimensions are involved.

For a given data point X_i, and a given query point Q, we call the distance between X_i and Q as Pandistance $PD(X_i,Q)$. $PD(X_i, Q)$ does not calculate the aggregated differences between X_i and Q on all dimensions. Instead, it only takes into account those dimensions on which X_i is close enough to Q, and sums them up. This strategy not only avoids the negative impacts from those dimensions on which X_i is far to Q, but also eliminate the curse of dimensionality caused by similarity functions such as Euclidean distance which calculates the square root of the sum of squares of distances on each dimensions. On more dimensions X_i is close (within the sets of nearest neighbor) to Q, the smaller Pandistance X_i has to Q. If we have two data points X_i and X_j, we judge which data point is closer to Q based on how many dimensions on which they are close enough (within dimension-wise nearest neighbors) to Q, as well as their average distances to Q on such dimensions.

Given a data set DS, we first calculate the difference δ_{il} of each data point X_i to the query point on each dimension D_l. Then we sort the *ids* on each dimension D_l based on δ_{il}, and select the first K *ids* on each dimension D_l and put them into KS_l. We move the *ids* in all KS_l to the set GS, and calculate the $PD(X_i, Q)$ for each data point if its *id* is in GS. Finally, we sort the *ids* based on the Pandistance and select the first K *ids* in the sorted list as the *ids* of nearest neighbors of Q. We do not need to calculate the difference using different number of dimensions. The number of dimensions and the subset of dimensions associated with data point X_i are both dynamically decided depending on the values of X_i and their rankings on different dimensions.

5. Searching Nearest Neighbors in the Presence of Obstacles

The PanKNN algorithm solves the similarity search problems in a new perspective efficiently and effectively. However, it does not consider the cases where there are obstacles in the data sets from which we try to find the nearest neighbors for given query point (an example is shown in figure 1). In this section we propose to design an algorithm in the presence of obstacles, which will be referred to as OPanKNN.

5.1. Definition

Let *n* denote the total number of data points and *d* be the dimensionality of the data space. Let *D*l be the *l*th dimension, where *l* = 1, 2, ..., d. Let the input d-dimensional data set be **X**

$$\mathbf{X} = \{X_1, X_2, ..., X_n\} \quad (5)$$

which is normalized to be within the hypercube $[0, 1]^d \subset R^d$. Each data point X_i is d-dimensional vector:

$$X_i = [x_{i1}, x_{i2},..., x_{id}] \quad (6)$$

Data point X_i has the *id* number *i*. Let Q be the query point: $Q = [q_1, q_2, ..., q_d]$. Let $\Delta_i = [\delta_{i1}, \delta_{i2}, ..., \delta_{id}]$ as the array of differences between the data point X_i and the query point Q on each dimension. There are obstacles existing in the data set as well. Obstacles can be represented in various ways. One simple and efficient way is to represent them as multidimensional points like the data points in the data set and the query point Q. Let m be the total number of obstacle points, and we can represent the set of obstacle points as:

$$C = \{C_1, C_2, ..., C_m\} \quad (7)$$

which is also normalized to be within the hypercube $[0,1]^d \subset R^d$. Each obstacle point C_h is a d-dimensional vector:

$$C_h = [c_{h1}, c_{h2}, ..., c_{hd}] \quad (8)$$

Each value c_{hl} where $h=1,2,...,m$ and $l=1,2,...,d$ represents obstacle point on dimension D_l where values on the two different sides of c_{hl} are obstructed to be in the same segment (zone).

5.2. Segments on Each Dimension

5.2.1 Segments:

Since the full data space is normalized, the value range of the data points on each dimension D_l, where $l = 1, 2, ..., d$ should be within the interval [0,1], as well as the value range of the obstacle points. On dimension D_l, the values of all the obstacle points are:

$$c_{1l}, c_{2l}, ..., c_{ml} \quad (9)$$

We sort them in ascending order

$$c_{1l'}, c_{2l'}, ..., c_{ml'} \quad (10)$$

where $c_{1l'} \geq 0$ and $c_{ml'} \leq 1$. For the purpose of consistency, let $c_{0l'}$ represent 0, and let $c_{m+1,l'}$ represent 1. Thus the value range on dimension D_l can be divided into m+1 zones (segments):

$$[c_{0l'}, c_{1l'}), [c_{1l'}, c_{2l'}), ..., [c_{ml'}, c_{m+1,l'}] \quad (11)$$

We use $Z_{l0}, Z_{l1}, ..., Z_{l,m}$ to represent them respectively. Figure 2 shows an example of segments on dimension D_l represented by Z_{lj} where j=0,1,...,m.

For a given query point, $Q = [q_1, q_2, ..., q_d]$, suppose its value q_l on $D_l \in [c_{kl'}, c_{k+1l'})$, or Z_{lk} where $k=0,1,...,m$ (as shown in figure 2). For each data point X_i in **X**, on each dimension D_l, where $l=1,2,...,d$, we not only check if its value x_{il} on D_l is close to q_l which is the value of Q on D_l, but also check if x_{il} is in the segment q_l belongs to on D_l.

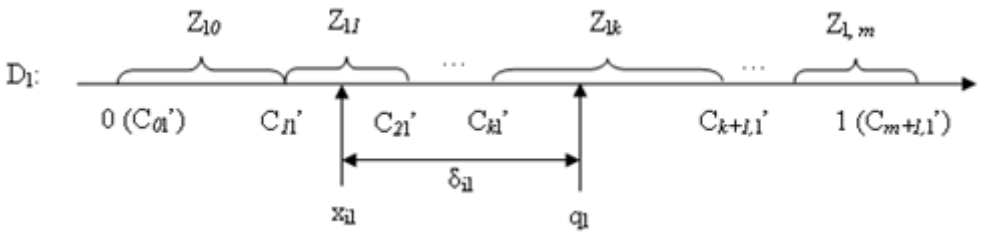

Figure 2: Segments on Dimension D_l

5.2.2 Example 1:

Here is an example. Suppose we have a 4-dimensional data set which contains 3 data points, and each of them can be represented as 4-dimensional vector:

X_1: [0.12, 0.13, 0.14, 0.21];
X_2: [0.41, 0.25, *0.101, 0.232];*
X3: [0.91, 0.32, 0.14, 0.52].

We represent each dimension as D_1, D_2, D_3 and D_4. If there are two obstacle points: C_1 = [0.15, 0.27, 0.94, 0.55] and C_2 = [0.66, 0.46, 0.88, 0.31], they divide each dimension (D_1, D_2, D_3 and D_4) into 3 zones (segments):

on D1 : [0, 0.15), [0.15,0.66), [0.66,1];
on D2: [0. 0.27) [0.27, 0.46), [0.46,1];
on D3: [0, 0.88) [0.88, 0.94) [0.94.1];
*on D4: [0, 0.*31) [0.31,0.55) [0.55,1].

We can use Z_{10} to represent the first segment on D_1: [0, 0.15), use Z_{11} to represent the second segment on D_1: [0.15, 0.66), etc.

For a given query point Q = [0.20, 0.20, 0.30, 0.40], its value on D_1 which is q1=0.20 falls into the second segment on $D1$: Z_{11} = [0.15, 0.66). For data point X_1: [0.12, 0.13, 0.14, 0.21], its value on D_1 is x_{11}=0.12. x_{11} is the closest to q_1=0.20 compared to x_{21}=0.41 and x_{31}=0.91.

However, x_{11} is not in the same segment with q_1 on D_1. On the other hand, x_{21} is farther from q_1 than x_{11}, however, it is in the same segment Z_{11} with q_1. Figure 3 shows the example.

5.2.3 Example 2:

Here is another example. Suppose we have 3-dimensional data set which contains 4 data points, and each of them can be represented as 3- dimensional vector:

X4: [0.21, 0.91, 0.32];
X5: [0.33, 0.45, 0.11];
X6: [0.10, 0.72, 0.53];
X7: [0.72, 0.15 0.37];

We represent each dimension as D_1, D_2, and D_3. If there are three obstacle points: C_3 = [0.11, 0.85, 0.66], C_4 = [0.79, 0.32, 0.10], and C_5 = [0.51, 0.20, 0.43], they divide each dimension (D_1, D_2, and D_3) into 4 zones (segments):

on D1 : [0, 0.11), [0.11,0.51), [0.51,0.79), [0.79,1];
on D2: [0. 0.20), [0.20, 0.32), [0.32,0.85), [0.85,1];
on D3: [0, 0.10), [0.10, 0.43) [0.43,0.66), [0.66,1].

We can use Z_{10} to represent the first segment on D_1: [0, 0.11), use Z_{11} to represent the second segment on D_1: [0.11, 0.51), etc. For a given query point Q' = [0.77, 0.84, 0.23], its value on D_2 which is q'_2=0.84 falls into the third segment on D_2: Z_{22} = [0.32, 0.85). For data point X_4: [0.21, 0.91, 0.32], its value on D_2 is x_{42} = 0.91. x_{42} is the closest to q'_2 =0.84 compared to x_{52} = 0.45, x_{62} = 0.72, and x_{72} = 0.15. However, x_{42} is not in the same segment with q'_2 on D_2. On the other hand, x_{62} is farther from q'_2 than x_{42}, however, it is in the same segment Z_{22} with q'_2. Figure 4 shows the example.

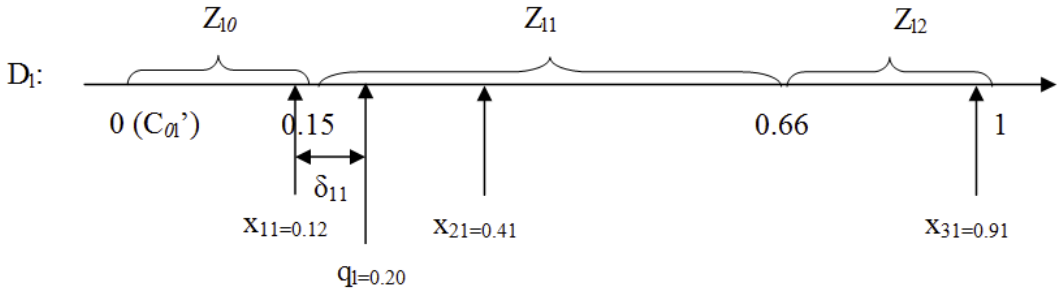

Figure 3: An Example of Segments, Obstacle Points, Data Points and Query Point on Dimension D_1

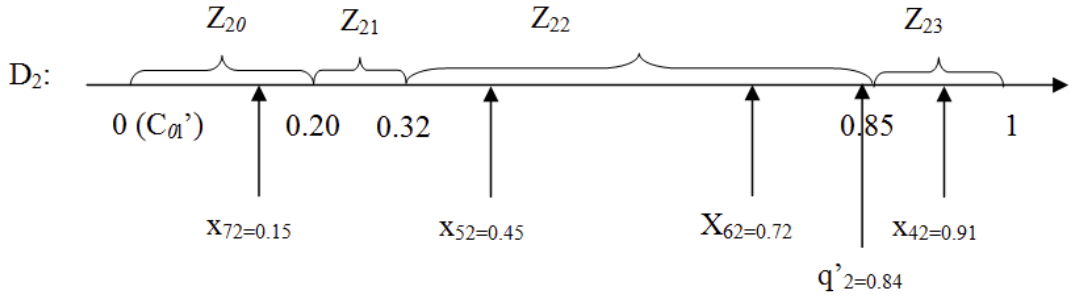

Figure 4: An Example of Segments, Obstacle Points, Data Points and a Query Point on Dimension D_2

5.3. Distance Calculation

From the examples above we can see that, if x_{il} is not in the same segment of q_l (figure 2), even if x_{il} is one of the K closest value to q_l, we still can not say it is very close Q on D_l. On the other hand, it is also inappropriate to completely discard x_{il} in the following calculation.

Here we adopt the fuzzy concept to determine the weight x_{il} should have when we calculate the distance between X_i and Q.

Given data set DS of n data points $\boldsymbol{X}$ = {X_1, X_2, ..., X_n} with d dimensions D_1, D_2, ..., D_d, query point Q, and set of obstacle points C = {C_1, C_2, ..., C_m} in the same data space, we first sort the data points on each dimension D_l, l=1, 2, ..., d, based on δ_{il} which is the difference between data point X_i and Q on dimension D_l. On each dimension D_l, l=1, 2, ..., d, let KS_l be the set which contains the *ids* of the first K data points in the sorted list. We call these first data points as dimension-wise K nearest neighbor to Q on D_l.

Those data points whose *ids* are in KS_l, however, might not be in the same segment (zone) with q_l. This is due to the possibility that Z_{lk} which q_l belongs to contains less than K data points.

For each data point X_i, $i=1, 2, \ldots n$, let $F_i = [f_{i1}, f_{i2}, \ldots, f_{id}]$ in which

$$f_{il} = \begin{cases} 0 & if \quad i \notin KS_l \\ 1 & if \quad i \in KS_l \quad and \quad x_{il} \in Z_{lk} \\ min(\frac{|\delta_{il}|}{min(|q_l - c_{kl'}|, |q_l - c_{k+1,l'}|)}, \frac{1}{|\delta_{il}|}) & otherwise \end{cases} \tag{12}$$

From the formula above, we can see that if X_i is one of the K nearest neighbors to Q on dimension D_l, but it is not in the segment with Q on D_l, its distance to Q on D_l will have weight as the minimum of $|\delta_{il}| / min(|q_l - c_{kl'}|, |q_l - c_{k+1,l'}|)$ and $1/|\delta_{il}|$ which are both larger than 1. This ensures that the distance between X_i and Q on D_l will be enlarged as "penalty" for X_i not being in the same segment with Q on D_l. The part of $1/|\delta_{il}|$ ensures that the enlarged distance will not exceed 1 which is the value range on D_l.

In the first example mentioned previously in this section, for X_1: [0.12, 0.13, 0.14, 0.21], its value on D_1 is $x_{11} = 0.12$. $q_1 = 0.20$ is in the segment $Z_{11} = [0.15, 0.66)$. $\delta_{11} = 0.12 - 0.20 = -0.08$. Here we demonstrate how to calculate f_{11}: if $i \notin KS_1$, $f_{11} = 0$; if $i \in KS_1$ and $x_{il} \in Z_{11}$, $f_{11} = 1$; otherwise, $f_{11} = min(|\delta_{11}| / min(|q_1 - c_{11'}|, |q_1 - c_{2,l'}|, 1/|\delta_{11}|)) = min(0.08/min(|0.20-0.15|, 0.20-0.66), 1/0.08) = min(1.6, 12.5) = 12.5$

Given two d-dimensional points $Xi = [x_{i1}, x_{i2}, \ldots, x_{id}]$ and $Q = [q_1, q_2, \ldots, q_d]$, with the existence of obstacle points $C = \{C_1, C_2, \ldots, C_m\}$, and D_l as the dimension l, $l=1, 2, \ldots, d$, the Pan-distance of X_i to Q in the presence of obstacles

$$PDO(X_i, Q) = \frac{\sum_{l=1}^{d} \delta_{il} * f_{il}}{(\sum_{l=1}^{d} f_{il})^2} \tag{13}$$

where δ_{il} is the difference between X_i and Q on D_l, f_{il} is the weight for x_{il} whose value depends on whether $i \in KSl$ and whether x_{il} is in the same segment with q_l on D_l. $PDO(X_i, Q)$ can also be defined as the product of the average distance of X_i to Q on those dimensions where X_i is in the sets of dimension-wise K nearest neighbors to Q, and the weight to the average difference based on how many dimensions there are where X_i is in the sets of K nearest neighbors to Q.

5.4. Finding Nearest Neighbors

Given a data set DS of n data points $X = \{X_1, X_2, \ldots, X_n\}$ with D_l as the dimension l, $l=1, 2, \ldots, d$, a query point Q in the same data space, and a set of obstacle points $C=\{C_1, C_2, \ldots, C_m\}$, we try to find set PKS which consists of k data points from DS so that for any data point $X_i \in PKS$ and any data point $X_j \in DS - PKS$, $PDO(X_i, Q) \leq PDO(X_j Q)$. The set PKS is the Pan-K Nearest Neighbor set of Q in DS in the presence of obstacles.

The OPanKNN algorithm is described in Figure 5.

Algorithm OPanKNN
(DS: data set,
Q: query point,
d: dimensionality of DS,
K: number of data points required)

Begin

1) For each $X_i \in DS$, we first calculate $\Delta_i = [\delta_{i1}, \delta_{i2}, ..., \delta_{id}]$ in which $\delta_{il} = |x_{il} - q_l|$;
2) On each dimension D_l, $l=1, 2, ..., d$, we sort the set of obstacle points $c_{1l}, c_{2l}, ..., c_{ml}$ in ascending order to $c_{1l}', c_{2l}', ..., c_{ml}'$. The value range on dimension D_l can be divided into m+1 zones (segments): $[c_{0l}', c_{1l}')$,$[c_{1l}', c_{2l}')$,..., $[c_{ml}', c_{m+1l}']$ represented by $Z_{l0}, Z_{l1}, ..., Z_{lm}$;
3) On each dimension D_l, l=1, 2, ..., d, we sort the *ids* of the data points in *DS*, based on δ_{il} for X_i. Let S_l be the sorted list on D_l;
4) Let KS_l be the subset of S_l which contains the first *K ids* in S_l. For each data point X_i, i=1, 2, ... n, we generate $F_i = [f_{i1}, f_{i2}, ..., f_{id}]$ in which the value of f_{il} is based on the calculation in the formula (12);
5) Let set $GS = \{i\}$ in which $i \in KS_l$, l=1, 2, ..., d. For each data point X_i, where $i \in GS$, we calculate $PDO(X_i, Q)$;
6) Sort $GS = \{i\}$ based on $PDO(X_i, Q)$;
7) Let set *PKS* contain the first $ids \in GS$. Return *PKS*.

End

Figure 5: Proc: OPanKNN

5.5. Time and Space Analysis

Suppose the size of the data set is *n* and there are *m* obstacle points. Throughout the process, we need to keep track of the information of all points, which collectively occupies $O(n + m)$ space.

For one query point *Q*, we need to sort the data points, sort the obstacle points, and select K distances to Q on each dimension. The time required is $d(nlogn + mlogm + K)$. With *l* query points, on each dimension, only one sorting is needed, and we need to select *K* distances to different query points for *l* times. The time requires is $d(nlogn + mlogm + lK)$.

6. Experiment

We conducted comprehensive experiments on both synthetic and real data sets to assess the accuracy and efficiency of the proposed approach. Our experiments were run on Intel(R) Pentium(R) 4 with CPU of 3.39GHz and Ram of 0.99 GB.

6.1. Experiments on High-Dimensional Data Set

To test the scalability of our algorithm over dimensionality, data size, K as the number of nearest neighbors required for the query points, and M as the number of obstacle points, we designed a synthetic data generator to produce data sets with normalized distributions. The sizes of the data sets vary from 10,000, 15,000, ... to 50,000, with the gap of 5,000 between each two adjacent data set sizes, and the dimensions of the data sets vary from 15, 20, ... to 50, with the gap of 10 between each two adjacent numbers of dimensions. We also generated random data points as obstacle points for the experiment.

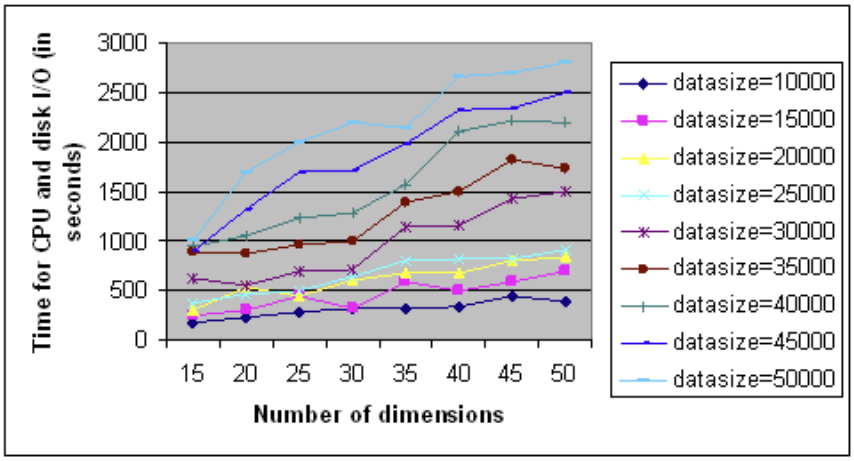

Figure 6: Running Time on One Query Point with Increasing Dimensions (K=20 and M=10)

Figure 6 shows the running time of groups of data sets with dimensions increasing from 15 to 50. Each group has fixed data size (from 10,000, 15,000, ... to 50,000). We set K as 20 and M as 10. Figure 7 shows the running time of groups of data sets on one query with sizes increasing from 10,000 to 50,000. Each group has fixed number of dimensions (from 15, 20, ... to 50). We set K as 20 and as M 10. The two figures indicate that our algorithm is scalable over dimensionality and data size. Figure 8 shows the running time of 3 groups of data sets with the size of 10000, 20000 and 30000 on one query with increasing from 5,10,... to 30. We set dimension K as 15 and M as 10.

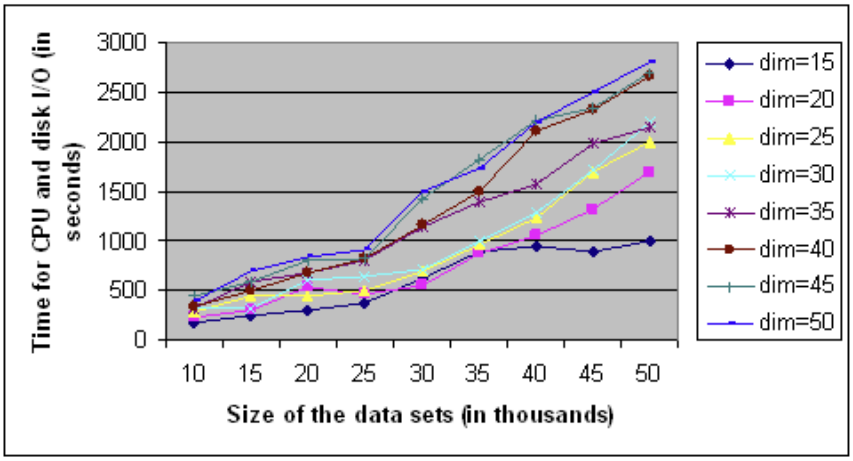

Figure7: Running Time on One Query Point with Increasing Dataset Sizes (K=20 and M=10)

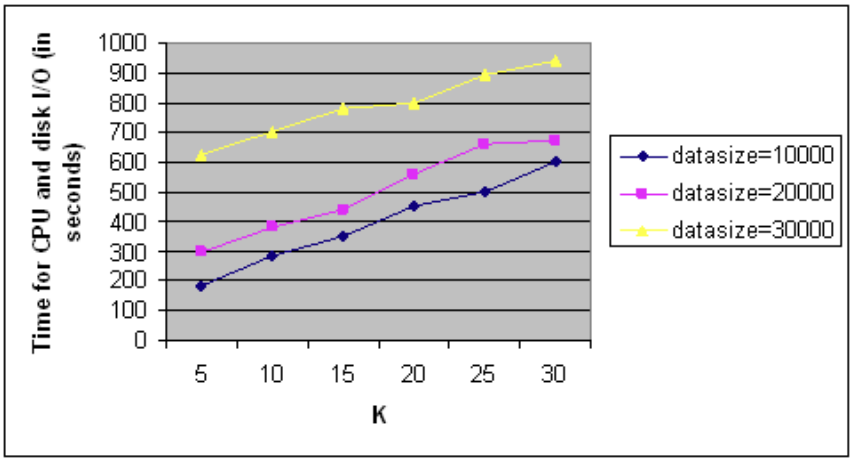

Figure 8: Running Time on One Query Point with Increasing K Values (Dimensionality=15 and M=10)

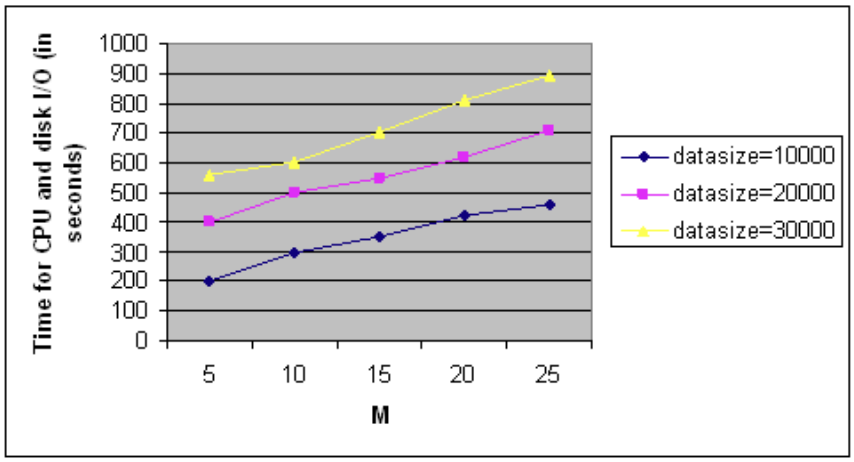

Figure 9: Running Time on One Query Point with Increasing M Values (Dimensionality = 15 and K=20)

Figure 9 shows the running time of 3 groups of data sets with the size of 10000, 20000 and 30000 on one query with increasing from 5, 10, ... to 25. We set dimension K as 15 and M as 20. Figure 8 and figure 9 indicate that our algorithm is scalable over the number of nearest points and the number of obstacle points M.

6.2. Experiments of PanKNN vs. OPanKNN

In this section we will demonstrate how OPanKNN improves the performance compared to the original PanKNN which does not consider the presence of obstacles.

We use two real data sets from UCI Machine Learning Repository [7] to demonstrate the performance difference of PanKNN vs. OPanKNN.

The first data set is Wine Recognition data set which contains the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. It contains 178 instances, each of which has 13 features 14 (dimensions), including alcohol, magnesium, color intensity, etc. The data set has three clusters with the sizes of 59, 71 and 48.

The second data set is Ecoli data set which contains data regarding Protein Localization Sites. This data set is made up of 336 instances, with each instance having seven features (dimensions). It contains 8 clusters with the sizes of 143, 77, 52, 35, 20, 5, 2 and 2.

We first use VizCluster on data sets to demonstrate the distribution of the data sets in an intuitive way. VizCluster [17] is an interactive visualization tool for multidimensional data. It combines the merits of both multidimensional scatterplot and parallel coordinates. Integrated with useful features, it can give a simple, fast, intuitive and yet powerful view of the data set. Due to the space limitation, here we just demonstrate the data distributions for Wine data and Ecoli data. Figures 10 and 11 show the demonstration on Wine data and Ecoli data respectively. Different shapes of the points present different cluster id information.

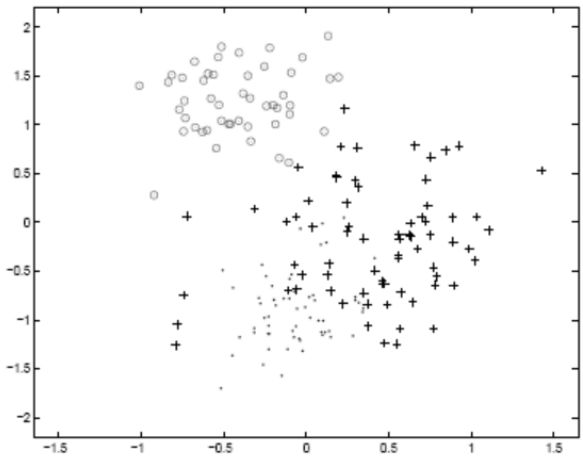

Figure 10: Demonstration of Wine Data Using VizCluster

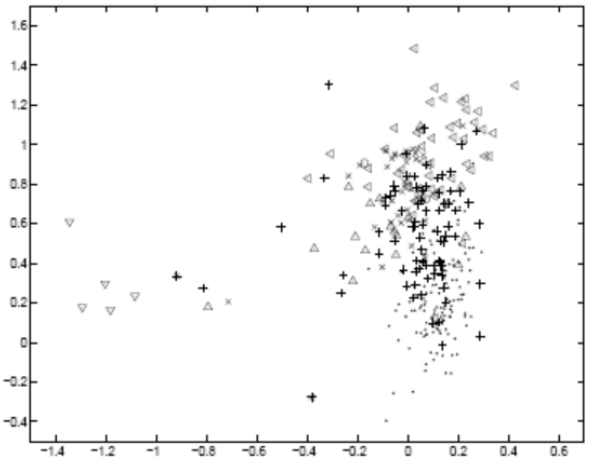

Figure 11: Demonstration of Ecoli Data Using VizCluster

We first perform the algorithms on the Wine data set. The accuracy rate of OPanKNN is 93.3%, which is higher than the accuracy rate of PanKNN (92.9%). We also run the algorithm on Ecoli data set. Again the OPanKNN algorithm has the higher accuracy rate (90.3%) than PanKNN (89.7%).

6.3. Experiments on More Real Data Set

We next evaluate the effectiveness of our proposed approach, OPanKNN, for finding nearest neighbors in the presence of obstacles. The real data sets were also obtained from UCI Machine Learning Repository [7].

The first one is the ionosphere data set which is a radar data set collected by system in Goose Bay, Labrador. It contains 351 data points, each of which has 34 dimensions. There are two classes in the ionosphere data: *g* as *good*, and *b* as *bad*.

The second data set is the glass data set for different glass types. It contains 214 data points, each of which has 9 dimensions. There are 7 classes in the glass data, class 1 to class 7.

The third data set is the iris data set for various iris plant types. It contains 150 data points, each of which has 4 dimensions. There are 3 classes in the iris data: Irissetosa, Irisversicolor, and Irisvirginica.

Here we demonstrate the testing results of those data sets and compare the results with other algorithms such as Frequent K-n-match algorithm [21] and IGrid [5].

We apply strategy to design the experiment and evaluation which is similar to the one described in [21]. For each real data set, we randomly select data points as the query points and obstacle points, and perform our algorithm using K as 10. 200 query points are randomly selected. For each of them, 5 data points are randomly selected as obstacle points, and 10 data points are retrieved as its nearest neighbors. If a retrieved data point has the same class with the query point it is associated with, and there is no obstacle point in between the retrieved data point and the query point, we call it successful retrieval. Otherwise, we call the data point unsuccessful retrieval. We calculate how many successful retrievals we have among the results from performing OPanKNN on these 200 query points, and divide it by 2000 (which is the number of query points times K) to calculate the accuracy rate.

We first perform the algorithms on the ionosphere data set. The accuracy rate of OPanKNN algorithm is 93.1%, which is higher than the accuracy rate of IGrid (87.9%), and that of Freq. K-n-match algorithm, which is 90.6%.

We next use the glass data set to test various algorithms. The accuracy rate of OPanKNN algorithm is 91.2%, which is higher than the accuracy rate of IGrid (86.5%), and that of Freq. K-n-match algorithm, which is 90.8%.

We conduct experiments on the iris data set as well. Among the three algorithms, OPanKNN has the highest accuracy rate which is 90.4%, higher than both IGrid (83.1%) and Freq. K-n-match algorithm (90.1%).

7. Conclusion

In the paper we present our strategy to design the similarity search approaches in the presence of obstacles. On each dimension we divide the value range into segments based on the obstacle points and conduct our OPanKNN algorithm to find K nearest neighboring points for a given query point *Q*. In the future work, we will conduct more experiments on synthetic and real data sets to test and demonstrate the efficiency and effectiveness of our approach.

References

[1] White D.A. and Jain R. Similarity Indexing with the SS-tree. In Proceedings of the 12th Intl. Conf. on Data Engineering, pages 516–523, New Orleans, Louisiana, February 1996.

[2] E. Achtert, C. Bohm, P. Kroger, P. Kunath, A. Pryakhin, and M. Renz. Efficient reverse k-nearest neighbor search in arbitrary metric spaces. In SIGMOD'06, pages 515–526, New York, NY, USA, 2006. ACM.

[3] C. C. Aggarwal. Towards meaningful high-dimensional nearest neighbor search by human-computer interaction. In ICDE, 2002.

[4] C. C. Aggarwal, A. Hinneburg, and D. A. Keim. On the surprising behavior of distance metrics in high dimensional space. Lecture Notes in Computer Science, 1973, 2001.

[5] C. C. Aggarwal and P. S. Yu. The IGrid index: reversing the dimensionality curse for similarity indexing in high dimensional space. In Knowledge Discovery and Data Mining, pages 119–129, 2000.

[6] Anthony K.H. Tung, Jean Hou and Jiawei Han. Spatial clustering in the presence of obstacles. In ICDE '01: Proceedings of the 17th International Conference on Data Engineering, page 359, Washington, DC, USA, 2001. IEEE Computer Society.

[7] S. D. Bay. The UCI KDD Archive [http://kdd.ics.uci.edu]. University of California, Irvine, Department of Information and Computer Science.

[8] D. A. Berchtold S., Keim and H.P. Kriegel. The X-tree: An index structure for high-dimensional data. In VLDB'96, pages 28–39, Bombay, India, 1996.

[9] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is "nearest neighbor" meaningful? In International Conference on Database Theory 99, pages 217–235, Jerusalem, Israel, 1999.

[10] J. C. Bezdek. Pattern Recognition with Fuzzy Objective Function Algorithms. Kluwer Academic Publishers, Norwell, MA, USA, 1981.

[11] B. Cui, H. Shen, J. Shen, and K. Tan. Exploring bit-difference for approximate KNN search in high-dimensional databases. In Australasian Database Conference, 2005.

[12] V. EstivillCastro and I. Lee. Autoclust: Automatic clustering via boundary extraction for mining massive point-data sets. In Proceedings of the 5th International Conference on Geocomputation, pages 23–25, 2000.

[13] V. EstivillCastro and I. Lee. Autoclust+: Automatic clustering of point-data sets in the presence of obstacles. In TSDM '00: Proceedings of the First International Workshop on Temporal, Spatial, and Spatio-Temporal Data Mining-Revised Papers, pages 133–146, London, UK, 2001. SpringerVerlag.

[14] R. Fagin, R. Kumar, and D. Sivakumar. Efficient similarity search and classification via rank aggregation, 2003.

[15] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In The VLDB Journal, pages 518–529, 1999.

[16] A. Hinneburg, C. C. Aggarwal, and D. A. Keim. What is the nearest neighbor in high dimensional spaces? In The VLDB Journal, pages 506–515, 2000.

[17] Li Zhang, Chun Tang, Yong Shi, Yuqing Song, Aidong Zhang and Murali Ramanathan. VizCluster: An Interactive Visualization Approach to Cluster Analysis and Its Application on Microarray Data. 2002.

[18] R. T. Ng and J. Han. Efficient and effective clustering methods for spatial data mining. In VLDB '94: Proceedings of the 20th International Conference on Very Large Data Bases, pages 144–155, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[19] T. Seidl and H.P. Kriegel. Optimal multi-step k-nearest neighbor search. SIGMOD Rec., 27(2):154–165, 1998.

[20] Y. Shi and L. Zhang. A dimension-wise approach to similarity search problems. In the 4th International Conference on Data Mining (DMIN'08), 2008.

[21] A. K. H. Tung, R. Zhang, N. Koudas, and B. C. Ooi. Similarity search: a matching based approach. In VLDB '06, pages 631–642. VLDB Endowment, 2006.

[22] X. Wang and H. J. Hamilton. Dbrs: A density-based spatial clustering method with random sampling. In PAKDD, pages 563–575, 2003.

[23] R. Weber, H.J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In Proc. 24th Int. Conf. Very Large Data Bases, VLDB, pages 194–205, 24–27 1998.

[24] L. A. Zadeh. Fuzzy sets. Information and Control, 8(3):338–353, 1965.

[25] O. R. Zaıane and C.H. Lee. Clustering spatial data when facing physical constraints. In ICDM '02: Proceedings of the 2002 IEEE International Conference on Data Mining, page 737, Washington, DC, USA, 2002. IEEE Computer Society.

Authors

Yong Shi received the BS and MS degrees, both in computer science, from the University of Science and Technology of China in 1996 and 1999, respectively. He received Ph.D. in computer science from the State University of New York at Buffalo in 2006. He is currently an assistant professor in the Department of Computer Science and Information Systems in Kennesaw State University. His research interests include data mining, database, machine learning, and information retrieval.

Li Zhang received his BS and MS degrees in computer science in 1991 and 1998, MA in mathematics in 1997, and Ph. D. in computer science from the State University of New York at Buffalo in 2004. Currently he is an associate professor of the department of computer science, Eastern Michigan University. His research interests include bioinformatics, data mining, pattern recognition, visualization, computer graphics, non-well-founded set theory, computational linguistic, knowledge representation, logical reasoning system, artificial intelligence, database, e-commerce and web services.

Lei Zhu received Ph.D. degree in Computer Science and Engineering from State University of New York at Buffalo in 2001, and received M.S. degree and B.S. degree in Computer Science and Technology from Peking University in 1998 and 1995, respectively. Currently he is an Associate Professor of Information Technology, Clayton State University, Morrow, Georgia. Dr. Zhu's research interests include content-based multimedia indexing and retrieval, Information retrieval and data mining.