Efficient Methods in Finding Aggregate Nearest Neighbor by Projection-Based Filtering

Yanmin Luo[1,2], Hanxiong Chen[1], Kazutaka Furuse[1], and Nobuo Ohbo[1]

[1] Dept.Computer Science,University of Tsukuba,Tennoudai 1-1-1,Tsukuba,Ibaraki 305,Japan
{lym,chx,furuse,ohbo}@dblab.is.tsukuba.ac.jp
[2] Dept. Computer Science, HuaQiao University, QuanZhou, FuJian, China, 362021
lym@hqu.edu.cn

Abstract. Aggregate Nearest Neighbor (ANN) queries developing from Nearest Neighbor (NN) queries are the relatively new query type in spatial database and data mining. An ANN queries return the object that minimizes an aggregate distance function with respect to a set of query points. Because of the multiple query points, ANN queries are much more complex than NN queries. For optimizing the query processing and improving the query efficiency, many ANN queries algorithms utilizes pruning strategies. In this paper, we propose two points projecting based ANN queries algorithms which can efficiently prune the data points without indexing. We project the query points into special ”line”, on which we analyses their distributing, then pruning the search space. Unlike many other algorithms based on the data index mechanisms, our algorithms avoid the curse of dimensionality and are effective and efficient in both high dimensional space and metric space. We conduct experimental studies using both real dataset and synthetic datasets to compare and evaluate their efficiencies.

Keywords: Aggregate Nearest Neighbor, Pruning strategies, Search Region.

1 Introduction

Similarity search is a very important query operation in data mining especial for information retrieval. It is most common that the similarity between two retrieval objects can be measured by their distance metric. Nearest Neighbor queries can be used to compute the similarity between objects in large database especially in spatial database retrieval such as in Geographic Information System(GIS). For minimizing the query processing time in similarity queries, most of the NN queries algorithms use the index mechanisms in their query processing. Utilizing the index-structure helps pruning or restriction of the search space. In spatial databases, datasets are usually indexed by spatial access methods (SAM) such as the R-tree[3] and the R*-tree[16]. Therefore many NN queries algorithms used them to implement query in multi-dimensional data space[1,9,15]. Some metric indices such as MB$^+$-tree[14], GNAT[17] and MVP-tree[18] also have been proposed for similarity queries.

ANN Queries are novel forms of NN queries. Given two sets of points in spatial database *P* and *Q*, *P={p_1,p_2,...p_N}* is the static source dataset and *Q={q_1,q_2,q_3......,q_n}* is the query points set. As defined in [6], the aggregate distance between a data point and query points set *Q* can be expressed by *adist(p,Q)=f(*$|pq_1|$, $|pq_2|$,...,$|pq_n|$*)*, where $|pq_i|$ is the Euclidean distance between point *p* and *qi*. Given a set P of static points, ANN Queries retrieves the data point *p* in *P* having minimal aggregate distance to *Q*. For ANN, different function f gives ANN different meaning. For example, if *f* is a *sum* function, the ANN queries will find the point *p* with minimal total aggregate distance to *Q*. And if f is the max function, the ANN query will return the point p which minimizes the maximum distance to the points in *Q*. Fig.1 shows the example. Fig.1(a) shows the case that f is the sum function, and the ANN queries will report the data point p_5 because it minimizes the total distance to Q. In other words, $adist(p_5,Q)=\Sigma_{i=1..3} |p_5q_i|<=adist(p,Q)$ for all *p* in *P*. Fig.1(b) shows the case when *f* is a *max* function, where the ANN queries report the data point p_7 which minimizes the maximum distance to *Q*. This is to say that $adist(p_7,Q)=max_{i=1..3}|p_7q_i|=|p_7q_1|<= adist(p,Q)$ for all *p* in *P*. Considering that several persons at specific locations (query points of *Q*) want to meet at a certain restaurant (a data point of *P*). The sum aggregate query return the restaurant (p_5) which leads to the minimum sum of distances that all the persons have to move while the max aggregate query return the restaurant (p_7) where all members can arrive together in the shortest time. Furthermore, ANN query now is becoming more and more important in many domains, such as clustering [2], outlier detection[4], GIS and mobile computing application [5,10,12].

Now NN problems and ANN problems are reasonably well solved for low dimensional applications for which many efficient indexing structures have been proposed. A wide variety of indexes have been proposed such as R-tree, most of them work well in low dimensional space. Many studies have shown that traditional indexing methods fail in high dimensional space. According to [11], NN and RNN search could be meaningless in high dimensional spaces due to the well-known curse of dimensionality. This motivates our interesting in non-indexing approaches. In order to compute as few distances as possible, some pruning strategies are always used to optimize the query processing in ANN queries [6,7,8]. Given a static dataset and several query points, and suppose that the static dataset is indexed by an R-tree. [6,7]

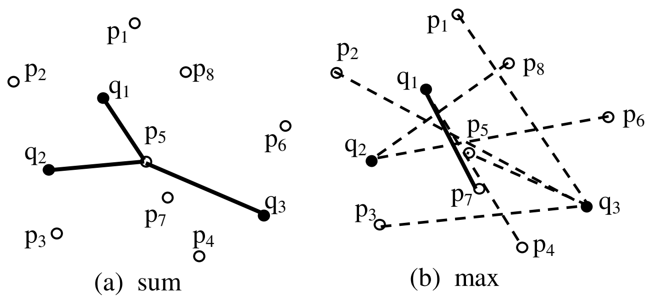


Fig. 1. Examples of ANN Queries. The solid points are query points and the hollow points are the data points. The point(s) connected by bold line(s) are the ANN results.

develop several ANN query algorithms for processing both memory-resident and disk-resident query sets. Some metrics are utilized to prune the search space for queries optimization. [8] uses an ellipse to approximate the extent of all query points, and the distance or MBR derived from the ellipse is used to prune intermediate index nodes during search via the R*-tree.

In this paper, we propose two non-indexing pruning strategies for ANN query processing. We assume that all query points can fit in main memory and only consider the *sum* function. Our methods find the search region in data space and prune other region. The remainder of this paper is organized as follows. Section 2 gives a brief survey of related work. Section 3 describes our pruning strategies for ANN queries and gives out the algorithms. Section 4 experimentally evaluates the effectiveness of our algorithms, and Section 5 concludes the paper.

2 Related Works

Similarity search in metric space is performed by measuring the (dis)similarity between objects by their distances. We adopted similar notations as those used in [13] with application-specific metric distance function d. Formally a metric space is a pair, $MS = (O, d)$, where O is a set of data values and d is a total (distance) function with the following properties (the metric postulate):

$$\forall x, y \in O; \quad d(x, y) = d(y, x). \tag{1}$$

$$\forall x \in O; \quad d(x, x) = 0. \tag{2}$$

$$\forall x, y \in O; \quad x \neq y \rightarrow 0 < d(x, y) < \infty. \tag{3}$$

$$\forall x, y, z \in O; \quad d(x, z) \leq d(x, y) + d(y, z). \tag{4}$$

Given a query object $q \in O$ and a nonnegative radius r, a r-neighbor search (r-N search) of q is to retrieve the objects in the r-neighbor of q:$\mathrm{RN}(q,r)=\{o|o \in O \wedge d(q,o) \leq r\}$. We call this region by r-neighbor region, see Fig. 2(a).

Assuming that the set Q of query points fits in memory and the data points are indexed by an R-tree, Dimitris Papadias et al. introduced the algorithms for memory -resident ANN queries in [7], one of them is Multiple Query Method (MQM). MQM utilizes the main idea of the threshold algorithm. It executes incremental NN search for each point in Q, and combines their results. MQM stores a threshold t_i for each query point q_i, The sum of all thresholds is defined to be the total threshold T. MQM computes the nearest neighbor for each query point incrementally, updates the thresholds and gets the current best nearest neighbor with the best distance, until threshold T is larger than best distance. In order to achieve locality of the node accesses for individual queries, MQM sorts the points in Q according to their Hilbert value. Thus, two subsequent queries are likely to correspond to nearby points and access similar R-tree nodes. But for different query points Q, MQM would have to sort again. The algorithm for computing nearest neighbors of query points should be incremental because the termination condition is not known in advance.

3 Our Methods

In this section we describe our methods on ANN queries and give out the algorithms. For the following discussion, we consider Euclidean distance and 2-Dimendional point datasets. But the proposed techniques are applicable to higher dimension. And we only focus on the sum function due to its significance in practice. Table 1 lists the symbols and their definitions in our algorithms.

Table 1. Symbols in this paper

Symbols	Definitions
N	Number of objects (points) in database
n	Number of objects in query dataset
$Q=\{q_1,q_2...q_n\}$	The query dataset
MBR	Minimum Bounding Rectangle
M	The area of MBR
$d(p,q)$	Distance function between two objects
$D(p,q)=\lvert pq\rvert$	Euclidean distance between two objects
$adist(p,Q)$	$=f(\lvert pq_1\rvert,\lvert pq_2\rvert,...,\lvert pq_n\rvert)$, aggregate distance
$C(q,r)$	$=\{p \mid p\in M, \lvert pq\rvert<=r\}$
$A(q,r_1,r_2)$	$=\{ p \mid p\in M, r_1<= \lvert pq\rvert<=r_2 \}=\{C(q,r_2)-C(q,r_1)\}$
$S(p)$	$\equiv S(p,Q)=\bigcup C(q_i,\lvert pq_i\rvert)$

Thinking about the points in dataset P are distributing uniformly in data space. As for the *sum* function, it is seen from Fig.2(b) that the result of ANN queries (we call it the best ANN point) should be in the MBR of the query points set. The points outside of the MBR can be pruned. We discuss non-index pruning methods. Pruning means we have to find a search region in data space and get the best ANN point in this region, instead of searching in full data space. The most important technique of our methods is how to find the search region which is equal to or includes the MBR of query points set and can substitute it. As we introduce our algorithms, we always first analyze how to get the proper search region.

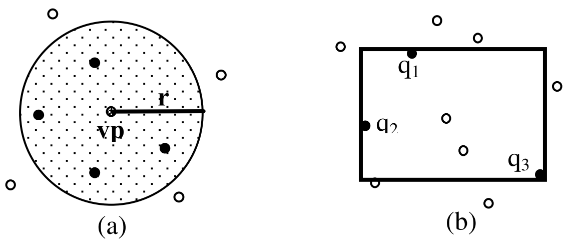


Fig. 2. (a) an example of the r-neighbor region of pivot *vp* , (b) an example of the MBR of Q, which includes three points q_1,q_2,q_3

3.1 *vp* -ANN Algorithm

Given a point *vp* and radius *r* we can divide the data space into two partitions. The shadow region in Fig.2 (a) is the r-neighbor region of *vp*. We call the other region by *out-neighbor* region. Assuming that *p* is the best ANN point, then *p* minimizes $adist(p,Q) = sum(|pq_1|, |pq_2|, \ldots, |pq_n|)$. It can be ensured that for every point q_i the best ANN point should be in its r-neighbor region. This means that the out-neighbor region of q_i can be pruned in our ANN queries. According to table 1, we can describe the r-neighbor region of the point *vp* by $C(vp, r) = \{p | p \in P, |p,vp| <= r\}$. For each point of q_i in *Q* we can get the search region of ANN query by the definition of Lemma 1.

Lemma 1. *Let $R_i = C(q_i, r_i)$ be r-neighbor region of q_i, and $R = \bigcup_{i=1..n} R_i$ be the union area of R_i. If each r_i is chosen that $\bigcap_{i=1..n} R_i$ is not empty, then the best ANN point must be found in R.*

Lemma 1 is clear and by it, we can search only region *R* to find the best ANN point. Now what we have to do is to select r_i for each query point q_i. We have to choose each r_i carefully in order to make the initial region *R* as small as possible. One way to do this is to choose a point *vp* in dataset *P* randomly, and select the Euclidean distance between *vp* and q_i as r_i. However, a randomly selected *vp* point may make *R* cover almost all of the data points in dataset *P*, and only a few points can be pruned. Therefore a good *vp* would lead to fewer distance calculations. Thinking over all the query points we know the aggregate centroid of *Q* is the best choice of *vp*, because as for sum function this point is the one in data space that minimizes the value of *adist(q,Q)*. We can say that the aggregate nearest neighbor is a point of *P* "near" the aggregate centroid of *Q*. The aggregate centroid depends on the function f and its exact computation is not always possible. It should be computed at the initialization phase of *vp*-ANN. Thinking about the case $f=sum$, *q* minimizes the function $adist(q,Q) = \sum_{i=1..n} |qq_i|$. Enlightened from the SPM algorithm in [6], let (x,y) be the coordinates of q and (x_i,y_i) be the coordinates of query point q_i . Since the partial derivatives of function *adist(q,Q)* with respect to its independent variables x and y are zero at the centroid q, we have the following equations:

$$\partial adist(q,Q)/\partial x = \sum_{i=1..n} ((x-x_i)/\mathrm{sqrt}((x-x_i)^2+(y-y_i)^2))=0. \quad (5)$$

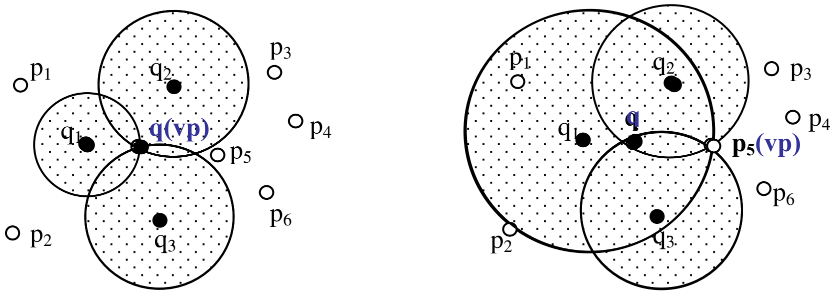
$$\partial adist(q,Q)/\partial y = \sum_{i=1..n} ((y-y_i)/\mathrm{sqrt}((x-x_i)^2+(y-y_i)^2))=0. \quad (6)$$

Unfortunately, the above equations cannot be solved into closed form for $n>2$. This is to say that it is difficult to calculate this centroid. In our implementation, we just think about the case of coordinate space and use the geometric centroid to substitute it. The geometric centroid q(x,y) of *Q* is:

$$x = (1/n)\cdot\sum_{i=1..n} x_i. \quad (7)$$

$$y = (1/n)\cdot\sum_{i=1..n} y_i. \tag{8}$$

Although the geometric centroid of *Q* is the best choice of *vp*, it is not certain a point of the dataset P. If the dimension is high and the data points are few and far away from each other in data space, it is possible that R would be a NULL set, see Fig.3(a). For this reason we find the nearest neighbor of the geometric centriod in P (Assuming it is p_k), and choose it(p_k) as the *vp* point instead of the geometric centriod. On this *vp*, there is at least one point p_k in R, Fig.3(b) shows this. p_5 is selected to be *vp*, for it is the nearest neighbor of q. Fig.4 shows *vp*-ANN algorithm.



(a) *vp* is the geometric centroid q of *Q* (b) *vp* is the nearest neighbor of *q*

Fig. 3. The search region according to different *vp*. *q* is the geometric centroid of *Q* but q is not a point of P. p_5 is the nearest neighbor of q of P. (a) selects q as the pivot, and (b) selects p_5 as pivot.

In the *vp*-ANN algorithm the dataset R always covers the region of the MBR of query points set, therefore this algorithm is strong and always can get the best ANN query point. We will discuss its efficiency in section 4.

```
Algorithm vp-ANN(Q:set of the query points) /* vp is the vantage point */
1. initialize $R=\Phi$ ;
2. compute the aggregate centroid $q$ of $Q$
3. find vp point as the nearest neighbor of q in dataset P
4. for each point $q_i$ in query dataset $Q$ do
5.    compute $r_i$; // $r_i=|q_i vp|$
6.    $R_i = C(q_i, r_i)$;
7.    Updata $R$; // $R=R \cup R_i$
8. end for
9. search $R$ to find the best ANN point;
```

Fig. 4. The *vp*-ANN algorithm

3.2 Projection-Based Pruning Algorithm

In this section, if a search region is small enough, and can cover the search region discussed in *vp*-ANN algorithm, then we say that this search region is a "*satisfied*" one. The heart of the method proposed in this section is to project query points into a carefully selected 'line'. The result of projection can approximately reflect the distributing of query points. On that we can determine the search region in which the query points distribute concentrated, and prune other region of data space. To do this, firstly we choose two so-called *pivots* points: p_a and p_b and consider the 'line' that passes through them in data space. The algorithm to choose pivot points will be discussed later in this section (see Fig.8). Then we project every query point into this line. The distance between pivot p_a and the projecting point of the query point on that line can be computed by using the cosine law. We mark this distance by $proj(p_a,q)$ and illustrate it in Fig.5. According to the cosine law, we can get the Eq.9, where $D(p_a,q)$ is the Euclidean distance between p_a and q. By this equation we know $Proj(p_a,q)$ is great than 0 when $\alpha<\pi/2$ and less then 0 when $\alpha>\pi/2$. This means that $Proj(p_a,q)\neq D(p_a,q')$, but $|Proj(p_a,q)|=D(p_a,q')$.

$$Proj(p_a,q)=(D(p_a,q)^2+D(p_a,p_b)^2-D(p_b,q)^2)/(2D(p_a,p_b)). \quad (9)$$

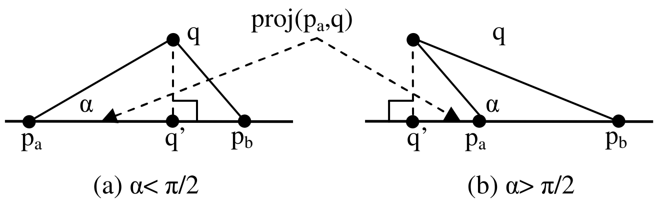
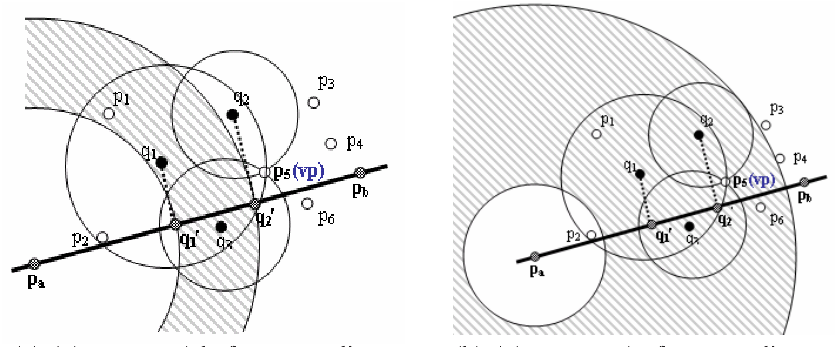


Fig. 5. Illustration of the projecting law

For each query point q_i we repeat the projecting and get the value of $proj(p_a,q_i)$. Then we find out the query point q_{max} with the maximal value of $proj(p_a,q_{max})$, and the query point q_{min} with minimal value of $proj(p_a,q_{min})$. Let q_{max}' be the projecting point of q_{max} on the line and q_{min}' be the projecting point of q_{min} on the line. Let $r_{max}=proj(p_a,q_{max})$ and $r_{min}=proj(p_a,q_{min})$. Also if $proj(p_a,q_{min})<0$, let $r_{min}=0$. Taking p_a as the centre of a circle we draw two circles with the radii r_{max} and r_{min} respectively, see Fig.6 (a). There must be a cirque between the two circles, see the shadow region. We call this region the candidate region, and express it by $A(p_a,r_{min},r_{max})$ (before amending) as in Tab.1. Comparing this candidate region with the search region in *vp*-ANN algorithm, the candidate region cannot cover all or most of the search region in *vp*-ANN algorithm, and even the point q_{max} (see q_2 in Fig.6 (a)) is not be included. Therefore this region is not a satisfied one and cannot be the search region. We have to extend this cirque region (candidate region), and enlarge it to cover all or most of the search region in *vp*-ANN algorithm. In our implementation we modify the value of r_{max} and r_{min} according to Eq.10 and Eq.11. Now we redraw the two circles and get the new cirque $A(p_a,r_{min},r_{max})$ which is the shadow region in Fig.6(b). We can see that the new cirque covers all or most of the search region of the *vp*-ANN algorithm.

$$r_{max} = r_{max} + D(q_{max}, q_{max}'). \tag{10}$$

$$r_{min} = r_{min} - D(q_{min}, q_{min}') \ \ (if\ r_{min} > 0). \tag{11}$$



(a) A(p_a,r_{min},r_{max}) before amending (b) A(p_a,r_{min},r_{max}) after amending

Fig. 6. The search region according to different *vp* points

Unfortunately this candidate region is still too large. As we can see in Fig.6 (b), the candidate region may include almost all the points of dataset and only a small number of points can be pruned. But this candidate region will be used to get the search region in our algorithm. Giving two cirques with different centres, if they are close to each other, there will be a overlapping region between them. Our strategy is: firstly, select two points p_{a1} and p_{b1}, and draw the line passing through them. Using the theory above, we can get the 1st candidate region: $A(p_{a1},r_{min1},r_{max1})$. Then we select the other two points p_{a2} and p_{b2} to determine the second line which different from the first one, and get the second candidate region $A(p_{a2},r_{min2},r_{max2})$. Because their centres: p_{a1} and p_{a2} are different points, and we project the same dataset P to the two different line, there must have an overlapping region between $A(p_{a1},r_{min1},r_{max1})$ and $A(p_{a2},r_{min2},r_{max2})$. Let $S= A(p_{a1},r_{min1},r_{max1}) \cap A(p_{a2},r_{min2},r_{max2})$, see Fig.7. We select S as the search region in our algorithm, because it is a "satisfied" one.

From the discussion above, it can be seen that pivot points affect the performance significantly. Clearly, we would find a line on which not only the projections of the query points are as far apart from each other as possible but also let all or most of the values of proj(p_a,q_i) be positive. To achieve this, the pivot points p_a and p_b should maximize the distance $|p_ap_b|$. However, this spends $O(N^2)$ distance computation. Moreover, if the line passes through the center of query set Q, the distance $|q_{max}q_{max}'|$ in Eq.6 and Eq.7 will be as small as possible. That would make the area $A(p_a,r_{min},r_{max})$ smaller too. This means we can prune more points of dataset P. For those reason, we propose a linear heuristic algorithm to get the pivot points which requires only $O(N)$ distance computations. Firstly we compute the aggregate centroid Q which is also necessarily in projecting operation, and compute its farthest point as the 1st pivot (p_a). Then we find the farthest point apart from p_a in dataset P, and make it be the 2nd pivot (p_b). The pivot points choosing algorithm is showed in Fig.8.

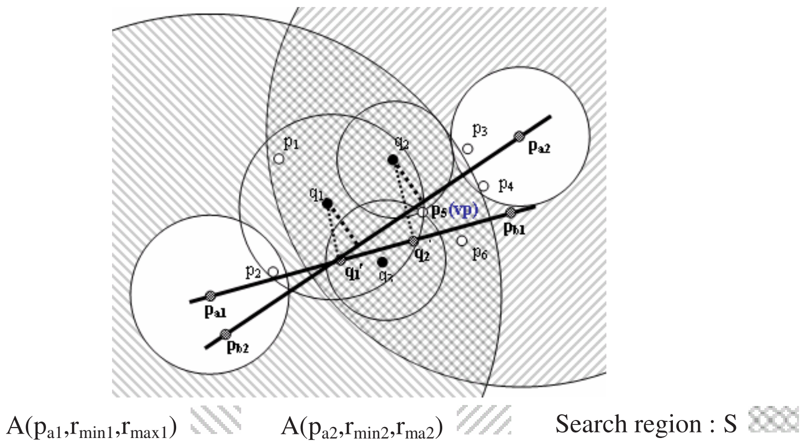


Fig. 7. The search region of projection-based algorithm

```
Algorithm pivot_points_choosing(Q:set of the query points)
1.    let $q_{cen}$ = the aggregate centroid of $Q$
2.    let $p_a = arg\ max_p \{|pq_{cen}|\}$ // the farthest point from $q_{cen}$
3.    let $p_b = arg\ max_p \{|pp_a|\}$
4.    return  $p_a$ and $p_b$
```

Fig. 8. The pivot point choosing algorithm of projection-based algorithm

```
Algorithm projection_based_pruning algorithm(Q:set of the query points)
1.  initialize S=Φ;
2.  for $s$=1, 2 do
3.      $\{p_{as}, p_{bs}\}$= pivot_points_choosing($Q$)
4.      for each point $q_i$ in $Q$ do
            project $q_i$ to the line $p_{as}p_{bs}$ passing through $p_{as}$ and $p_{bs}$
            calculate the value of $proj(p_{as},q_i)$
        endfor
5.      get the points $p_{max,s}$,$p_{min,s}$ with the maximal and minimal projection in
    line $p_{as}p_{bs}$
6.      calculate the value of $r_{max,s}$ and $r_{min,s}$
7.      pointset_A$s$= $A(p_{as},r_{min,s},r_{max,s})$   /*get the candidate region */
8.  endfor
9.  $S$ = $pointset\_A1 \cap pointset\_A2$ /*get the search region */
10. retrieve in $S$ and get the best ANN point;
```

Fig. 9. The projection-based pruning algorithm

Now we are ready to describe the projection-based pruning algorithm. Our algorithm accepts the query points set as input and find out the best ANN point in dataset. Fig.9 shows the pseudo code of the projection-based pruning algorithm.

4 Experiments

All the experiments explained in this section were programmed in C^{++} under the GNU/Linux operating system, and are run on an Intel Pentium(R) 4 CPU 2.40GHz PC, 512MB main memory. Datasets are as the following, with each coordinate been normalized to [0, 1]:

(i) WUpppoint, points of populated places of West part of United Stated of America (10493) which are available at www.rtreeportal.org.

(ii) A synthetic high dimensional dataset generated randomly according to the uniform distribution.

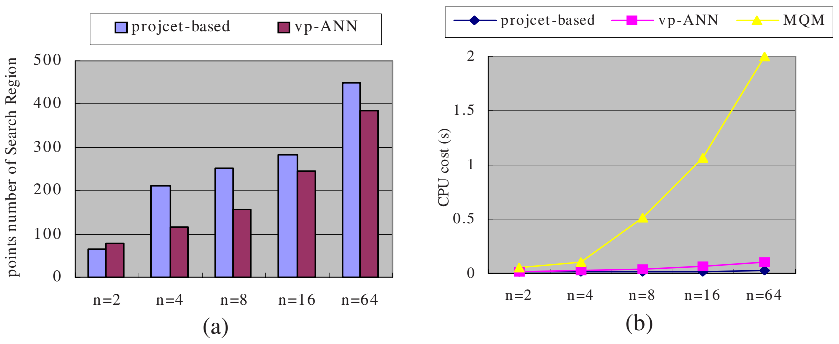


Fig. 10. Comparison on query set size (N=10000, M=0.08, 2-Dimension)

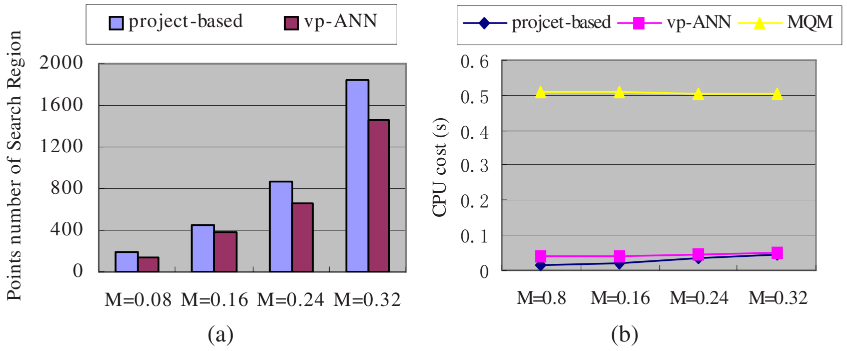


Fig. 11. Comparison on MBR area (N=10000, n=8, 2-Dimension)

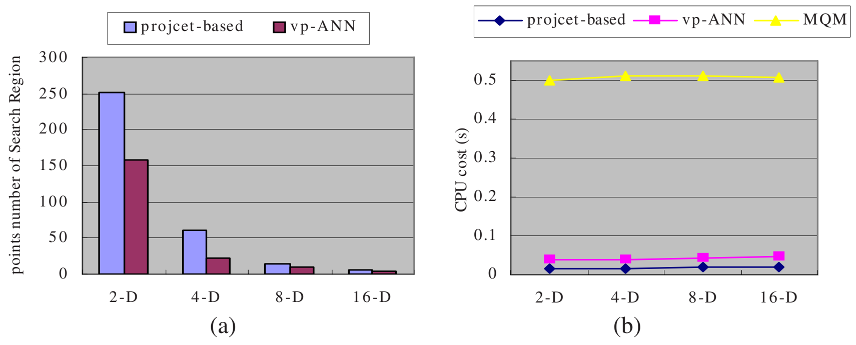


Fig. 12. Comparison on dimensionality (N=10000, n=8, M=0.08)

In our experiments, the query dataset is memory-resident. We use workloads of 50 queries. For each query, *n* is the point number in query points set *Q*. The query points are generated randomly in the workspace of *P*, and distributed uniformly in a MBR of area *M*. We compare the CPU cost and the screening effect (number of data points in search region) of our algorithms against MQM algorithm, studying the following cases of variation. That is, number of dimensions of data space, query set size (number of query points in *Q*) and the area of the MBR of *Q*. Fig.10 shows the average number of points in search region and CPU cost as functions of *n* on datasets (i). Fig.10 (a) shows that with the increasing of *n*, the point number of search region will become more. The points number of search region for projection-based always a little bigger than *vp*-ANN. This is because the search region of projecting-based always covers that of *vp*-ANN. Nevertheless, the CPU cost of projection-based is low. Fig.10 (b) shows that a lager *n* makes the CPU costs of all the three algorithms increase. But the CPU cost of projection-based algorithm goes up slower than the others, and MQM becomes worse rapidly. As in Fig.11, with the increasing of *M*, both the points number in search region and CPU cost of projection-based and *vp*-ANN algorithms increase as well. And the CPU cost of projection-based one increases more sharply while the CPU cost of MQM is almost unchanged. *n* is fixative, therefore the value of *n* and the area *M* become the dominative factors of our algorithms. For CPU cost, the projection-based algorithm is better than *vp*-ANN algorithm. As in Fig.12, if the dimensionality increases, the data is distributed more infrequently, and the number of data points in the search region will decrease sharply. While the CPU cost is stable and increases slightly, for fixed n and M. This result shows that the non-indexing algorithms perform well in high dimension space.

Focusing on the CPU costs, we know if *n* is small the CPU cost of all the three algorithms are similar. If *n* increases, the CPU cost of *vp*-ANN algorithm increases faster than projection-based algorithm. This is because *vp*-ANN algorithm has to calculate more r-neighbor region of q_i to find candidates, while a larger *n* will not influence the calculations of the candidate regions in projection-based method. In Fig.10 (b) we can see that when *n* is larger, the CPU cost of MQM method increases very sharply. This is because MQM has to sort the points of *P* for every query point *q* in *Q* according to their distance to the points in *P*. Thus a larger n will let the sorting become more time-consuming. As for different query, the query points set *Q* will change, and we have to redo the sorting in MQM algorithm. This is the main deficiency of MQM algorithm. On the other hand a different *Q* will not influence the executing of our algorithms.

5 Conclusion

ANN Queries are the extension of NN queries but are more complex because of multiple query points. Based on the coordinate space, we consider the case of *sum* function in ANN queries. Assuming that Q is memory-resident we describe and analyze two non-index algorithms for best ANN queries, and compare them with index-based methods. Our strategy finds the proper search region and enables pruning of irrelevant ones. The experimental results demonstrate that the methods we proposed performance well, especially in high dimensional space. In the future we intend to explore the case of other functions.

References

[1] Papadopoulos, A., Manolopoulos, Y.: Performance of Nearest Neighbor Queries in R-Trees. In: Afrati, F.N., Kolaitis, P.G. (eds.) ICDT 1997. LNCS, vol. 1186, pp. 394–408. Springer, Heidelberg (1996)

[2] Jain, A.K., Narasimha Murty, M., Flynn, P.J.: Data Clustering: A Review. ACM Comput. Surv. 31(3), 264–323 (1999)

[3] Guttman, A.: R-Trees: A Dynamic Index Structure for Spatial Searching. In: Proc. of ACM SIGMOD Int'l Conference, pp. 47–57 (1984)

[4] Aggarwal, C.C., Yu, P.S.: Outlier Detection for High Dimensional Data. In: SIGMOD Conference, pp. 37–46 (2001)

[5] Jensen, C.S., Kolárvr, J.: Torben Bach Pedersen, Igor Timko: Nearest neighbor queries in road networks. In: GIS 2003, pp. 1–8 (2003)

[6] Papadias, D., Tao, Y., Mouratidis, K., Hui, C.K.: Aggregate nearest neighbor queries in spatial databases. ACM Trans. Database Syst. 30(2), 529–576 (2005)

[7] Papadias, D., Shen, Q., Tao, Y., Mouratidis, K.: Group Nearest Neighbor Queries. In: ICDE 2004, pp. 301–312 (2004)

[8] Li, H., Lu, H., Huang, B., Huang, Z.: Two ellipse-based pruning methods for group nearest neighbor queries. In: GIS 2005, pp. 192–199 (2005)

[9] Uhlmann, J.K.: Satisfying general proximity/ similary queries with metric trees. Information Processing Letters 40, 175–179 (1991)

[10] Lin, K.-I., Nolen, M., Kommeneni, K.: Utilizing Indexes for Approximate and On-Line Nearest Neighbor Queries. In: IDEAS 2005, pp. 83–88 (2005)

[11] Beyer, K., Goldstein, J., Ramakrishnan, R., Shaft, U.: When is Nearest Neighbors Meaningful? In: Proc. of the Int. Conf. Database Theorie, pp. 217–235 (1999)

[12] Yiu, M.L., Mamoulis, N., Papadias, D.: Aggregate Nearest Neighbor Queries in Road Networks. IEEE Trans. Knowl. Data Eng. 17(6), 820–833 (2005)

[13] Paolo Ciaccia, M.P., Zezula, P.: M-tree: An efficient access method for similarity search in metric spaces. In: Proceedings of the 23rd VLDB Conference, Athenes, Greece, 1997, pp. 357–368 (1997)

[14] Ishikawa, M., Chen, H., Furuse, K., Yu, J.X., Ohbo, N.: MB+Tree: A Dynamically Updatable Metric Index for Similarity Searches. Web.-Age. Information Management, 356–373 (2000)

[15] Roussopoulos, N., Kelley, S., Vincent, F.: Nearest Neighbor Queries. In: SIGMOD Conference 1995, pp. 71–79 (1995)

[16] Beckmann, N., Kriegel, H.-P., Schneider, R., Seeger, B.: The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles. In: Proc. of ACM SIGMOD Int'l Conference, pp. 322–331 (1990)

[17] Brin, S.: Near neighbor search in large metric spaces. In: Proceedings of the 21st VLDB Conference, pp. 574–584 (1995)

[18] Bozkaya, T., Ozsoyoglu, M.: Distance-based inde- xing for high-dimensional metric spaces. In: Proceedings of the ACM SIGMOD Conference on Management of Data, Tucson, Arizona, pp. 357–368 (May 1997)