Voronoi-Based Aggregate Nearest Neighbor Query Processing in Road Networks

Liang Zhu Yinan Jing Weiwei Sun Dingding Mao Peng Liu
School of Computer Science, Fudan University
Shanghai, China
{09210240054, jingyn, wwsun, maodingding, 0572150}@fudan.edu.cn

ABSTRACT
Aggregate nearest neighbor (ANN) query returns a common interesting data object that minimizes an aggregate distance for multiple query points. In this paper, we propose a novel approach to efficiently process ANN queries in road networks. This approach includes two processes: initializing process and pruning process, which are both based on computing *k*NN of query points in network Voronoi diagram. Experimental results show that our approach outperforms existing approaches on both response time and page accesses.

Categories and Subject Descriptors
H.2.8 [**Database Management**]: Database Applications. *Spatial Databases and GIS*

General Terms
Algorithms, Performance

Keywords
Query Processing, ANN, Voronoi Diagram, Road Networks

1. INTRODUCTION
Many researchers have focused on nearest neighbor (NN) queries and its variants in road networks. These queries are widely applied to many geographic applications. As one variant of the *k*NN query, *aggregate nearest neighbor* (ANN) queries return the object that minimizes an aggregate distance with respect to a set of query points [3]. This type of query is often raised in our real life. For example, several friends at different locations usually want to find a common meeting place on weekends. Different from normal *k*NN queries, in an ANN query, there are multiple query points and the query result is dependent on the specific aggregate function, which can usually be *sum*, *max* or *min*. Take the problem of finding a common meeting restaurant for example, the *sum* function makes sure that the sum of distance that all persons have to travel is minimum, the *max* function makes sure the travel time consumed by all persons is shortest, and the *min* function makes sure one person's arriving time is earliest.

Figure 1(a) shows an example of the road network, where points such as n_1,…, n_7 are intersections in the road network and the weight on edges is the time or distance travelled from one endpoint of this edge to another. And we assume p_1, p_2, p_3 are interesting data objects, e.g. gas station. In addition, there are two query points (e.g. two persons) who locate at q_1, q_2 respectively. We let $P = \{p_1, p_2, p_3\}$ and $Q = \{q_1, q_2\}$. In this example, when the aggregate function is *sum*, the ANN for Q is p_2. Similarly, when the aggregate function is *max*, then obviously the ANN is p_3. When function is *min*, we can easily find out p_2 is ANN.

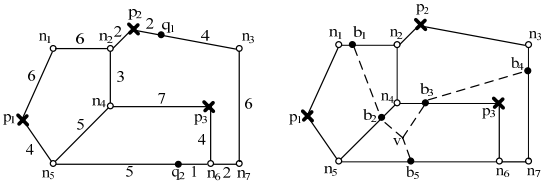


Figure 1. An example of road network and its Network Voronoi Diagram

As shown in the above example, it is more complex to process ANN queries in road networks than in Euclidean spaces, because the position and accessibility of spatial objects are constrained by the computation of network distance, which is usually the length of the shortest path connecting two objects. In this paper, we solve ANN queries in road networks based on network Voronoi diagram. In our approach, we use the Voronoi-based *k*NN query processing method [4] to extend our search space step by step. The advantage of our approach is that we needn't retrieve the network from disk, and just utilize the look-up tables of network Voronoi diagram generated in advance. Thus the ANN query processing performance of our approach will be better than that of other existing approaches. To the best of our knowledge, this is the first work dealing with the ANN query based on network Voronoi diagram.

2. RELATED WORK
Group nearest neighbor (GNN) query, which is first proposed by D. Papadias et al. in [1], can be viewed as the predecessor of ANN query. This problem is defined as follows: Given two sets of points P and Q, a GNN query retrieves the point(s) of P with the smallest sum of distances to all points in Q. So GNN query problem is equal to ANN query problem when the aggregate function in ANN query is *sum*. Then D. Papadias et al. extended this problem and proposed ANN query problem in [2]. They have proposed three algorithms to process ANN queries. However, these three algorithms are not applicable in the context of road networks, because they are applicable in Euclidean space. Furthermore, M.L. Yiu et al [3] have proposed three algorithms to

ACM GIS '10 , November 2-5, 2010. San Jose, CA, USA.

solve ANN queries in road networks. The first algorithm is called *Incremental Euclidean Restriction* (*IER*), which uses R-tree to index the interesting data objects and prunes the search space by comparing the shortest aggregate distance with the Euclidean distance from the query point to MBR. *Threshold algorithm* (*TA*) is the second algorithm, which incrementally expands the network around each query point and applies some top-k aggregate query processing technique to guide and terminate the search. The third algorithm, *Concurrent Expansion* (*CE*), is similar to *TA*. In these three algorithms, the *IER* is the best one on performance.

3. BACKGROUND

A road network can be modeled as a weighted graph, in which the intersections (n_i) and interesting objects (p_i) are vertices. The distance between two vertices is the length of the shortest path rather than their Euclidean distance. A *network Voronoi diagram* (*NVD*) divides a network into *network Voronoi polygons*, namely *NVP* (p_i). Figure 1(b) is the NVD of the original network depicted in Figure 1(a). In the NVD, each interesting object p_i called *generator point* is the generator of the *NVP* (p_i). And we call the points such as b_1,…, b_5 as *border points*. A border point is the intermediate point of a path from one generator point to another. S. Louis Hakimi [6] has proposed an algorithm to generate a NVD from a network graph. For a NVD, we need to store all of border points of NVD. Furthermore, to search k nearest neighbors of q based on NVD, we also need to store all of NVP's adjacent NVP and the pre-computed distance information. We use a similar storage schema as in [4] to store the loop-up table of a NVD.

Based on NVD, we can apply the VN^3 algorithm [4] proposed by Kolahdouzan, M et al to process kNN queries, which outperforms another algorithm proposed in [5]. The VN^3 algorithm consists of filter process and refinement process. The filter process generates candidate set which contains the adjacent *NVPs* of the generators that have been selected as the nearest neighbor of q. The refinement process accurately computes the shortest network distance from q to the newly-added generators in the candidate set according to the pre-computed distance which stored in the look-up tables. The filter/refinement process is iterative and must be invoked k times to find the first k nearest neighbors of q.

4. VORONOI-BASED ANN QUERY PROCESSING

Table 1. Frequently used symbols

Symbol	Description
P	set of interesting points (data objects)
Q	set of query points, $Q = \{q_1,..., q_n\}$
S	ANN candidates set, i.e. set of $p_i \in P$ which is possible to be ANN
S_i	set of objects which are extended by q_i
H	heap which is used to store all q_i with the min or max network distance from q_i to its latest kNN as its priority
p_{i1nn}	q_i's first NN, and $p_{i1nn} \in P$
p_{iknn}	q_i's kNN, and $p_{iknn} \in P$
$dist(p_i, q_i)$	the min network distance from p_i to q_i
$dist(p_{i1nn}, q_i)$	the min network distance from q_i to its first NN
$dist(p_{iknn}, q_i)$	the min network distance from q_i to its kNN
$dist(p_i, Q) = f(dist(p_i, q_1),…, dist(p_i, q_n))$	aggregate distance between p_i and query points in Q

A naïve method to process an ANN query is to traverse the whole network, and compute all aggregate distances $dist(p_i, Q)$ for each $p_i \in P$. Then we can find the ANN with the minimum aggregate distance. However, this method is very expensive because the entire network has to be traversed. In this section, we will introduce our approach based on NVD to process ANN queries. Our approach consists of two processes: initializing process and pruning process. Table 1 shows symbols used in our description.

4.1 Initializing Process

The main objective of initializing process is to obtain an ANN candidates set S, in which the ANN we wanted is definitely included. We assume there are n query points in Q. Initializing process is as follows: to begin with, we search the first NN p_{i1nn} for each $q_i \in Q$ and put p_{i1nn} respectively into the set S_i, and then decide whether there is an intersection among S_1,…, S_n. If not, we will continue to extend the search space for one query point q_i according to one certain extending strategy by computing its next nearest neighbor. When retrieving the kNN for each q_i, we apply VN^3 algorithm [4] based on the NVD generated in advance. And then we put p_{iknn} into S_i, which is actually a ordered list of nearest neighbors of q_i, namely $S_i = \{p_{i1nn}, p_{i2nn},…, p_{iknn}\}$. This process will end until there is an intersection among S_1,…, S_n, i.e. $S_1 \cap S_2 \cap … \cap S_n \neq \emptyset$. This means we have found out the first common $p \in P$ which is accessed by all q_i. So p is the current best candidate of ANN, and we can compute aggregate distance $dist(p, Q)$ which denotes the *best_dist*.

Lemma 1. When we have found out the first common $p \in P$ which is extended by all q_i ($S_1 \cap S_2 \cap … \cap S_n = \{p\}$), let $S = S_1 \cup S_2 \cup … \cup S_n$, then ANN must be in S.

Proof. Suppose p' is ANN and $p' \notin S$. $\because p' \notin S$, $\therefore p' \notin S_i$. Because S_i is a ordered list of nearest neighbors of query point q_i, we can conclude that for each $q_i \in Q$, $dist(p', q_j) > dist(p, q_j)$. This conclusion is illustrated clearly in Figure 2. Then we can infer that $dist(p', Q) > dist(p, Q)$. This denotes p' is not ANN. It is departure from the assumption, so ANN must be in S. □

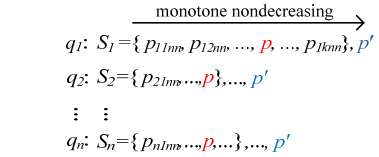


Figure 2. An example of initializing process.

4.2 Pruning Process for *sum* Function

After above mentioned initializing process, the algorithm turns to pruning process. During this process, it will prune data objects which cannot be ANN from set S. In fact, it is a process that lets the size of set S become smaller gradually. When there is only one object in S, then that object is ANN we want to find. It is worth noticing that the pruning strategies for aggregate *sum* function and *max* function are different. Due to the limited space, here we will mainly introduce pruning strategy for *sum* function.

Pruning strategy for *sum* function. Before pruning objects from S, we first select a query point according to some extending strategy to retrieve its next nearest neighbor (p'). Then we compute $dist = \sum_{i=1}^{n} dist(x_i, q_i)$, where

$$x_i = \begin{cases} p' & p' \in S_i; \\ p_{i1nn} & p' \notin S_i; \end{cases} \quad (1)$$

If *dist* > *best_dist*, then we compute the set S_i' for each q_i.

$$S_i' = \begin{cases} \{o | o \in S_i, dist(o, q_i) < dist(p', q_i)\} & p' \in S_i; \\ \Phi & p' \notin S_i; \end{cases} \quad (2)$$

Then let $S' = S_1' \cup \ldots \cup S_n'$. For $\forall p \in S \wedge p \notin S'$, object p can be pruned from S.

Lemma 2. Let p' is the next nearest neighbor of one query point. Let $dist = \sum_{i=1}^{n} dist(x_i, q_i)$, if *dist* > *best_dist*, for $\forall p \in S \wedge p \notin S'$, object p can be pruned from S.

Proof. For each i where $p' \in S_i$, $\because p \notin S'$, $\therefore p \notin S_i'$, then we can get $dist(p, q_i) \geq dist(p', q_i)$ in terms of definition of S_i'. For each i where $p' \notin S_i$, we have $dist(p, q_i) \geq dist(p_{i1nn}, q_i)$. Because $dist = \sum_{i=1 \wedge p' \in S_i}^{n} dist(p', q_i) + \sum_{i=1 \wedge p' \notin S_i}^{n} dist(p_{i1nn}, q_i)$, $dist(p, Q) = \sum_{i=1}^{n} dist(p, q_i) \geq dist$. $\because dist > best\_dist$, $\therefore dist(p, Q) > best\_dist$. Hence p cannot be ANN, and can be pruned. □

If *dist* < *best_dist*, we just need put that query point q_i's next nearest neighbor p' into its extended set S_i, and then judge whether p' is accessed by all query points or not. If this is true, we need to update the *best_dist* by the value of $dist(p', Q)$ and change current best candidate of ANN to be p'. Then we will continue this process until there is only one object in S. Algorithm 1 shows the pseudo-code of our ANN query processing algorithm.

Algorithm 1. Voronoi-Based_ANNQuery(*NVD*, *P*, *Q*)

1: H = new heap; $S = \emptyset$; $S_i = \emptyset$; *best_dist* = ∞; $i = 1$;
2: Create a new heap entry E;
3: while ($i \leq n$)
4: compute q_i's first nearest neighbor (*nn*) p_{i1nn};
5: $S_i = \{p_{i1nn}\}$; $E.qid = q_i$; $E.dist = dist(p_{i1nn}, q_i)$;
6: insert(H, E); $i = i + 1$;
7: while ($\bigcap_{i=1}^{n} S_i = \emptyset$)
8: E = removeHead(H);
9: $q_{current} = E.qid$; compute $q_{current}$'s next nn p';
10: $S_i = S_i + \{p'\}$; $E.dist = dist(p', q_{current})$;
11: insert(H, E);
12: $S = \bigcup_{i=1}^{n} S_i$; $\{p\} = \bigcap_{i=1}^{n} S_i$; *best_dist* = $dist(p, Q)$;
13: while ($|S| > 1$)
14: E = removeHead(H);
15: $q_{current} = E.qid$; compute $q_{current}$'s next nn p';
16: $dist = \sum_{i=1 \wedge p' \in S_i}^{n} dist(p', q_i) + \sum_{i=1 \wedge p' \notin S_i}^{n} dist(p_{i1nn}, q_i)$;
17: if (*dist* > *best_dist*) then
18: compute S_1', S_2',…, S_n';
19: $S = S \cap (\bigcup_{i=1}^{n} S_i')$; $S_{current} = S_{current} + \{p'\}$;
20: else
21: $S_{current} = S_{current} + \{p'\}$;
22: if ($p' \in \bigcap_{i=1}^{n} S_i$) then
23: $S = S - \{p\}$; $p = p'$; *best_dist* = $dist(p, Q)$;
24: $E.qid = q_{current}$; $E.dist = dist(p', q_{current})$;
25: insert(H, E);
26: Return ANN = last object in S; aggregate distance = *best_dist*;

The first part of Algorithm 1 is the initializing process (line 1-12). For each query point q_i, the algorithm first finds the first nearest neighbor and put their first nearest neighbor into their extended set S_i. In addition, each query point is inserted into a *min heap* (H) with the weight $dist(p_{i1nn}, q_i)$ (lines 3-6). Then we compute the next nearest neighbor (for example p') of the top element in H and add it into corresponding query point q_i's extended set S_i. If p' doesn't belong to the intersection of extended set of all query points, it needs to update the weight of query point q_i in H as $dist(p', q_i)$ (line 7-11). Until there is an interesting data object p which is accessed by all query points, then the algorithm compute the aggregate distance $dist(p, Q)$(line 12).

The next part is the pruning process which is the core of algorithm (lines 13-25). The pruning strategy is stated as above. Line 16 computes the value of *dist*. If the next nearest neighbor p' of top element in H belongs to some extended set S_i, we use the actual distance value from p' to each query point q_i to compute the value of $dist(p', q_i)$, otherwise we use the distance from query point q_i to its first nearest neighbor, i.e. $dist(p_{i1nn}, q_i)$. The value of *dist* we compute in this way is the lower bound of aggregate distance from p' to all query points. If *dist* > *best_dist*, some data objects that can't be ANN can be pruned by intersecting the sets S and S' (line 19). If *dist* ≤ *best_dist* and p' is also a common object, Line 23 updates the current best candidate ANN and *best_dist*. Line 24-25 updates the weight of query point $q_{current}$ in H.

4.3 Extending Strategy

There are three kinds of extending strategies when query points extend their search space by gradually computing their nearest neighbors.

Minimum distance first extending strategy. This strategy is based on the intuition that the location of ANN is more likely close to the centroid of query points. Every time we extend the query point whose distance from it to its latest extended neighbor is minimum. This leads all query point extend to centroid fairly. Hence it can avoid some useless extending.

Maximum distance first extending strategy. This strategy gives priority to the query point whose distance from it to its latest extended neighbor is maximum when extending. Because this strategy will extend one query point all the time until it visits all data objects, it will not be applied in our approach and the following experiments.

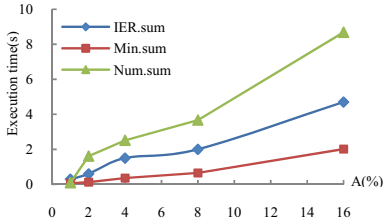
Number-based extending strategy. This strategy is to extend all query points circularly. In fact, there must be a specific optimal extending order for query points. However, we cannot know this optimal extending order in advance. So this strategy might cause some useless search space extending operations, which can cause few or no objects pruning.

5. EXPERIMENT EVALUATION

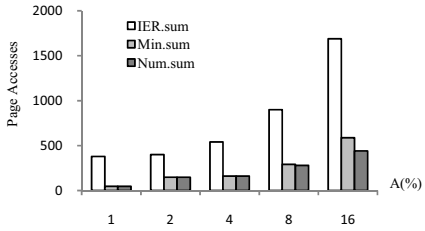
In this section, we evaluate the efficiency of the proposed algorithm. We conduct several experiments for different kinds of extending strategy and compare the performance with *IER* algorithm [3]. We use the real road network (San Francisco road map [7]) as data sets in our experiments. We uniformly generated interesting data objects on the network edges. The query points in Q are generated randomly on edges connected sub-network covering A percent of the network edges. For each experiment setting, we averaged the results of the algorithm over 10 queries. The experiments were executed on a PC with a Pentium(R) CPU of 2.0GHz.We used an LRU memory buffer of 1Mb and the page size was set to 4Kb.

Figure 3 shows for function *sum* the ANN query processing performance of our algorithm and *IER* on response time and page

accesses. There are 8 query points in these experiments. As expected, we can find the cost of three kinds of algorithm increase with A since the search space will be larger when A is bigger.

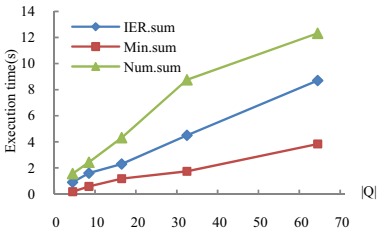


(a) Response time for *sum* function

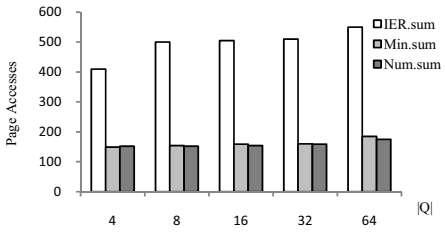


(b) Page accesses for *sum* function

Figure 3. Cost as a function of query area *A*.



(a) Response time for *sum* function



(b) Page accesses for *sum* function

Figure 4. Cost as a function of query points number |*Q*|.

However, as shown in Figure 3(a) our algorithm with *minimum distance first* extending strategy (*Min.sum*) is better than *IER* on response time. The reason is that although *IER* uses R-tree index, but it still need explore the network and compute the aggregate distance, while our algorithm need not expand the network and just search the look-up tables. However, the response time of our algorithm with *number-based extending* strategy (*Num.sum*) is larger because there are many invalid extensions which barely prune interesting data objects in S. Figure 3(b) shows the performance of page accesses. Our algorithm outperforms *IER*. The reason is that *IER* need explore the network and can retrieve only a small number of edges at each step, whereas our algorithm retrieves the pre-computed values from disk in one step. Especially when the value of A is small, the memory buffer is large enough to store the pre-computed values.

Figure 4 shows the effect of the number of query points ($|Q|$) on the query processing performance. We can find that the response time increases with the increment of $|Q|$, because more query points need to be considered when processing ANN query. In addition, we can observe that the increment of $|Q|$ has little influence on the disk I/O performance of our algorithm.

6. CONCLUSION

In this paper, we have studied the aggregate nearest neighbor queries in road networks and proposed a novel approach to solve ANN problem in road networks. Our approach is based on computing kNN of query points in network Voronoi diagram. In addition, we have discussed three kinds of extending strategies. Our experimental results show that our approach outperforms existing approaches on both response time and page accesses when we apply the *minimum distance first* extending strategy.

7. ACKNOWLEDGMENTS

This research is supported in part by the National Natural Science Foundation of China (NSFC) under grant 61073001.

8. REFERENCES

[1] Dimitris Papadias, Qiongmao Shen, Yufei Tao, and Kyriakos Mouratidis. 2004. Group Nearest Neighbor Queries. *In proceedings of the 20th International Conference on Data Engineering* (Boston, USA, March 30 - April 2, 2004). ICDE '04. 301-312.

[2] Dimitris Papadias, Yufei Tao, Kyriakos Mouratidis, and Chun Kit Hui. 2005. Aggregate nearest neighbor queries in spatial databases. *ACM Trans. Database Syst. (TODS) 30(2)*. 529-576.

[3] Man Lung Yiu, Nikos Mamoulis, and Dimitris Papadias. 2005. Aggregate Nearest Neighbor Queries in Road Networks. *IEEE Trans. Knowl. Data Eng*. 17, 6, 820-833.

[4] Mohammad R. Kolahdouzan, and Cyrus Shahabi. 2004. Voronoi-Based K Nearest Neighbor Search for Spatial Network Databases. *In Proceedings of the Thirtieth international conference on Very large data bases* (Toronto, Canada, August 31 - September 3, 2004). VLDB '04. 840-851.

[5] Dimitris Papadias, Jun Zhang, Nikos Mamoulis, and Yufei Tao. 2003. Query Processing in Spatial Network Databases. *In Proceedings of the Thirtieth international conference on Very large data bases* (Berlin, Germany, September 9-12, 2003). VLDB '03. 802-813.

[6] S. Louis Hakimi, Martine Labbé, and Edward F. Schmeichel. 1992. The Voronoi Partition of a Network and Its Implications in Location Theory. *INFORMS Journal on Computing 4(4)*. 412-417.

[7] Real Datasets for Spatial Databases: Road Networks and Points of Interest - San Francisco Road Network, http://www.cs.fsu.edu/~lifeifei/SpatialDataset.htm