Efficient Aggregate Farthest Neighbour Query Processing on Road Networks

Haozhou Wang, Kai Zheng, Han Su, Jiping Wang, Shazia Sadiq, and Xiaofang Zhou

School of ITEE, The University of Queensland,
St. Lucia, Brisbane, QLD 4072, Australia
{h.wang16,kevinz,h.su1,j.wnag28,shazia,zxf}@uq.edu.au

Abstract. This paper addresses the problem of searching the k aggregate farthest neighbours (AkFN query in short) on road networks. Given a query point set, AkFN is aimed at finding the top-k points from a dataset with the largest aggregate network distance. The challenge of the AkFN query on the road network is how to reduce the number of network distance evaluation which is an expensive operation. In our work, we propose a three-phase solution, including clustering points in dataset, network distance bound pre-computing and searching. By organizing the objects into compact clusters and pre-calculating the network distance bound from clusters to a set of reference points, we can effectively prune a large fraction of clusters without probing each individual point inside. Finally, we demonstrate the efficiency of our proposed approaches by extensive experiments on a real Point- of-Interest (POI) dataset.

1 Introduction

As one of the most important types of spatial query, efficient nearest neighbour query processing has been investigated extensively[1,2,3]. This type of query to find k nearest neighbors (kNN) form a given point becomes a fundamental operation in spatial databases, leading to a number of variations. As much as the interest in finding kNN objects, there are a large number of real-life applications which are interested in finding farthest neighbours (FN) [4]. Another important type of kNN variations is the so-called aggregate nearest neighbour (ANN) query [5,2]. The difference between ANN query and NN query is that an ANN query takes multiple query points into account and returns a point from the dataset that minimizes the aggregated distance from the point to all given query points using a user-specified aggregate function (e.g. *max*, *sum*). An example of ANN query with *sum* function is for a number of people to find a meeting place that can minimize their total traveling distance.

In an analogous way, aggregate farthest neighbor (AFN) query can also be defined as an extension to the FN query: for a given set of query points Q and a user-specified aggregate function, find a point p from a data set P such that the aggregate distance from p to all the points in Q is the *largest*. AkFN can be defined as a general case of AFN to find k such points which have larger

H. Wang and M.A. Sharaf (Eds.): ADC 2014, LNCS 8506, pp. 13–25, 2014.
© Springer International Publishing Switzerland 2014

aggregate distances than any other points in P. To illustrate its usefulness, consider a business franchise planning to open a new store. In order to reduce the mutual influences between the new and existing branches to maximise the overall profit, it is desirable for the location of the new store to be far away from all existing stores. By including the locations of all existing stores in Q, all the available locations from a real estate database as P and max as the aggregate function, an AkFN query can find the best candidate locations to choose from.

Despite of its importance for many applications, AkFN query has not been well studied. Just like ANN query processing is a non-trivial extension to NN query processing, AkFN query processing is quite different form kFN query processing and demands new processing strategies. To the best of our knowledge, there exists only one piece of work on AkFN query processing [6]; it, however, considers a simpler case to use Euclidean distance (i.e., in a free space). In this paper, we will investigate AkFN in the context of road networks. The motivation for us to consider road networks is that, in most real applications the movement of people and vehicles is constrained by a underlying road network. Albeit more complex in distance calculation, it is more reasonable and accurate to use network distances rather than Euclidean distances. This is because, in reality, road network contains some properties such as bridges and one way street, which makes the distance shortest path between two points in road network is longer than its Euclidean distance. Therefore, the incorporation of road networks can raise serious efficiency issues for processing AkFN query. The reasons can be two-fold. First, there is still no effective way to index a large number of objects in a road network. Classical hierarchical spatial access methods (e.g., R-tree [7] based) cannot work since they are designed for Euclidean space. Second, network distance evaluation is much more expensive than Euclidean distance evaluation since it involves online shortest path computation.

Our paper aims to propose efficient solutions for answering the AkFN query in road networks. More specifically, we firstly organize the objects in the whole dataset into clusters by apply a network-based hierarchical clustering method. Then we define a set of reference points across the entire space and pre-compute the maximum and minimum network distances between each pair of cluster and reference point. Lastly, we design an efficient search algorithm to spot the most promising clusters that may contain the results and prune the rest of them by leveraging the pre-computed information. It is worth noting that, since the clusters are hierarchical, we can achieve good trade-off between the number of clusters and pruning effect. In summary, we make the following major contributions in this paper: 1) We are the first to investigate the aggregate farthest neighbour query in the context of road networks. 2) We propose efficient solutions for processing the AkFN query by pre-computing and pruning at cluster level. 3) We conduct extensive experiments based on real POI dataset to verify the efficiency of our proposals.

The rest of this paper is organized as follows. In Section 2 we will introduce necessary preliminaries and formally define the AkFN query. We detail our proposed query processing algorithms in Section 3. Section 4 presents the

experimental results for validating the efficiency of our algorithms, followed by a brief literature review on related work in Section 5. We conclude the paper in Section 6.

2 Problem Definition

In this section we will introduce all necessary preliminary concepts and formulate the AkFN query. We summarize the major symbols and notations used throughout the paper in Table 1 for convenience of reference.

Table 1. Table of Notations

Notation	Definition
(v_i, v_j)	A road segment with two road segment nodes v_i and v_j
p_i	A point in the points dataset P
q_i	A query point in the query points set Q
$d_n(p_i, p_j)$	The network distance between p_i and p_j
$d_{agg}(p_i, Q)$	The aggregate distance from a point p_i to Q
r_i	A reference point in the reference points set R
$dr(p_i, q_i)$	The shortest path distance from p_i to q_i via q_i's reference point
C_i	A sub-cluster in hierarchical cluster structure C
$dc(C_i, r_i)$	The maximum network distance from C_i to r_i
$dr(C_i, q_i)$	The network distance from C_i to q_i via q_i's reference point
$dr_{agg}(p_i, Q)$	The aggregate distance from p_i to Q via relate reference points
$dr_{agg}(C_i, Q)$	The aggregate distance from C_i to Q via relate reference points

Definition 1 (Road Network and Network Distance). *A road network G is modeled as a weighted indirect graph $G = (V, E)$, where V is a set of road intersection , and E is a set of road segment. The network distance d_n between p_a and p_b, where p_a and p_b are two points on G, is calculated as the sum of the distance of the road segments along the shortest path between p_a and p_b.*

Notice that we use the term "network distance" and "shortest path distance" interchangeably.

Definition 2 (Aggregate Network Distance). *Given a point p, a query point set Q and road network G, the aggregate network distance between p and Q is $d_{agg}(p, Q) = f_{q \in Q}\ d_n(p, q)$, where f is a pre-defined aggregate function (e.g., sum, mean, min, max, etc).*

In this paper we only consider two types of aggregate functions, namely *sum* and *min*, as they are most applicable in a farthest neighbor query. Now we are in a position to formally define the query.

Definition 3 (AkFN query). *Given a dataset P and a query point set Q, the aggregate k farthest neighbor (AkFN) query retrieves a set S of k points from P that have the largest aggregate network distance with Q, i.e.,*

$$d_{agg}(p, Q) \geq d_{agg}(p', Q), \quad \forall p \in S, \forall p' \in P - S$$

Consider Figure 1 as an example, where q_1, q_2 is the query set Q and $p_1, p_2...p_8$ is the candidate dataset P, that we want to find a point from P, such that its minimum distance to Q is maximized. This is a special case of the AkFN query with $k = 1$ and min aggregate function. By enumerating the locations in P and simple calculation, it is easy to get that p_7 is the best location suiting for the request.

3 Query Processing Algorithm

The most straightforward approach to answer an AkFN query is to exhaustively search all the points in dataset, calculate the aggregate distances from each point to the query point set, and finally obtain the top k results. However, this method, called *exhaustive search* algorithm, has serious efficiency issues especially on road network, since the exhaustively search algorithm need to search the whole dataset P while P is usually very large (e.g., more than 100k points) in practice.Consequently, evaluating the aggregate network distance between all points in the dataset and the query "on-the-fly" can be extremely time consuming.

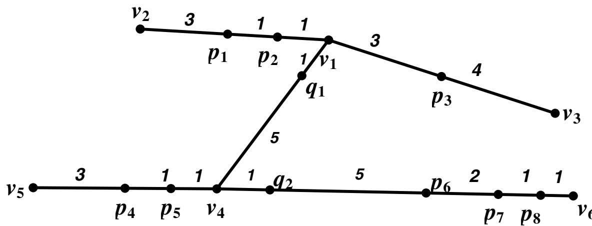
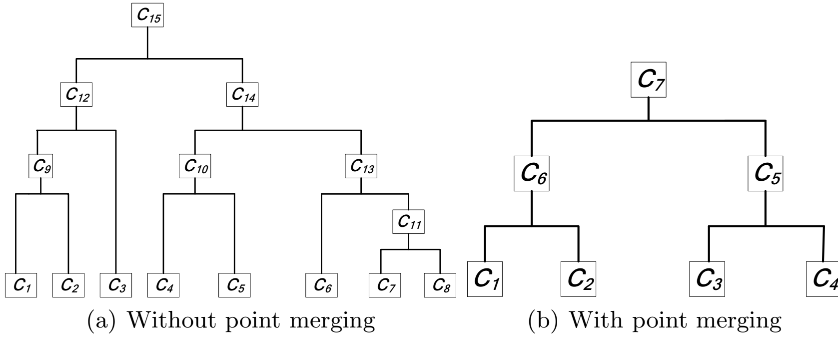


Fig. 1. Example of road network

To improve the efficiency of query processing, we propose an advanced approach with two carefully designed search algorithms, which leverage the power of hierarchical cluster and pre-computed network distance bound of each cluster to reference points to reduce the search space extensively. The query processing consist of three steps: clustering the points of the dataset, pre-computing the network distance bound and searching. The first step clusters the points of the dataset in hierarchical structure by applying the Linkage Hierarchical Clustering algorithm [8], a network-distance-based clustering method. In the second step, we uniformly define a set of *reference points*, which are mapped to road segment nodes, and pre-compute the network distance bounds between each pair of the clusters, generated in the first step, and reference points. This information can be saved for further use since the numbers of both clusters and reference points are relatively small comparing to the original dataset. The third step is searching the hierarchical structure with two different ways by search algorithms. In the rest of this section, we will describe each step in detail.



(a) Without point merging (b) With point merging

Fig. 2. Example of hierarchical cluster

3.1 Linkage Hierarchical Clustering

Using Euclidean-distance-based index (i.e. R-tree [7]) to prune unnecessary points is unsafe for AkFN query, since two points are close on Euclidean space do not mean they are close on road network. For example, two points are located in each side of a river and a bridge is located far away from these two points, absolutely, their network distance is much longer than their Euclidean distance. Obviously, in order to improve the performance of AkFN, a road-network-based point organizing structure is needed.

Thus, we adopt the Linkage Hierarchical Clustering algorithm [8], a well-known road-network-based clustering method, in our search approaches. The motivation for us to use hierarchical clustering is the trade-off between the number of clusters and the searching performance. Though a smaller number of bigger clusters can reduce the overhead cost incurred by storing and processing these clusters, the pruning effect will also be harmed since the distance bound is too loose to be useful. By organizing the objects into clusters with different levels and sizes, we can control the level from which the search starts and thus achieve a reasonable balance between the cluster number and pruning effect. Fig. 1 gives an example of a point dataset $P = p_1, p_2, \cdots, p_8$ on road network and its hierarchical cluster structure is shown in Fig. 2(a), which each cluster in the bottom level is the point itself in P.

However, consider each point of dataset P as a cluster at initiating time of the above algorithm can result in too many bottom level clusters when the size of P is big, which may cause memory overflow in pre-computing step and inefficient query processing in searching step. In order to reduce memory consumption and speed up the query processing, we merge points of P before applying clustering, if the road network distances between such points are less than a threshold ϵ. Fig. 2(b) shows an example of the hierarchical cluster structure with points merging for previous example (Fig. 1). With $\epsilon = 3$, p_1, p_2 are merged as C_3; p_3 cannot be merged with other points and considered as C_2. Meanwhile, p_4, p_5 and p_6, p_7, p_8 are merged as C_3 and C_4 respectively.

3.2 Pre-computation

In the searching step, we need to calculate distance of shortest road network path between query points and clusters many times. However, the shortest path searching is time consuming and could not support the"on-the-fly" query processing. Hence we deploy a reference point based pre-computing approach to boost the query processing to the real-time level. Inspired by the reference point generating method of [9], we split whole map area into small grids, and then for each grid cell gc we select the road segment node v, where v is in gc and v is the nearest road segment node to the center of gc, as a reference point of the road network. If there is no any road segment node in a grid cell, then no reference point will be selected in that grid cell.

The pre-computing is to calculate the maximum shortest path distance $dc(C, r) = \max_{p \in C}\{d_n(p, r)\}$ between each pair of cluster C of all the hierarchical clusters and point r of reference points set R. Therefore, the space cost of building reference list of pre-computed distance is $O(mn)$, where n is the number of clusters in the bottom level of the hierarchical structure, which is much less the total number of points in the road network; and m is the number of reference points. Meanwhile, the distance between reference points and upper level clusters can be calculated directly by previous information.

When a query point q is given, we find its nearest reference point r^*, that is $r^* = \arg\min_{r \in R}\{d_n(q, r)\}$. Then the $dr(C, q) = dc(C, r^*) + d_n(q, r^*)$ can be the upper bound of the shortest path distance $d_n(C, q)$ according to the triangle inequality. The upper bound can efficiently filter out some obviously impossible points during farthest neighbor search, which will demonstrate in the next section. Meanwhile, it is worthy to note that the more reference points mapped to road network, the more tight upper bound we can have, since the query point q could be much closer to reference point r and leads that $dr(C, q)$ is closer to $d_n(C, q)$. On the other hand, a lot of reference points means a heavy memory usage and may excess the total memory usage due to all reference points are loaded into memory during query processing. Hence, we limit the number of reference points, which suit for system memory or user request, to map in the road network, and we conduct a set of experiments to show the effect about number of reference points.

3.3 Search Algorithm

We propose two advanced search algorithms, namely Flat Search (FS) and Hierarchical Search (HS), which make use of the hierarchical clusters and pre-computed network distance bounds to improve the searching efficiency. FS only searches the bottom hierarchy of these clusters for the result; HS searches the whole hierarchy clusters.

Flat Search. We propose the FS algorithm by using the bottom hierarchical clusters to reduce the search space on road network. When given a query points

set Q, we can quickly find the nearest reference point r_i of q_i ($q_i \in Q$) and the shortest path distance between q_i and r_i.

Firstly, FS initializes an empty list A with a fixed length $|k|$, where k is the number of top answers required by user. The elements of A are the result points so far, and sorted by aggregate distance $d_{agg}(p_c, Q)$. Meanwhile, we assign a parameter kth_so_far which is used to record the smallest $d_{agg}(p_c, Q)$in A to 0 . The value of kth_so_far can be a lower bound of the farthest neighbour query, that any point p' with the aggregate distance $d_{agg}(p_c, Q) < kth_so_far$ will be filtered out. Then, the FS starts at a random cluster C_i in bottom layer of hierarchical cluster, it calculates the aggregate distance from C_i to Q (denote as $dr_{agg}(C_i, Q)$). To calculate $dr_{agg}(C_i, Q)$, we firstly find the r_i, that r_i is the nearest reference point of q_i ($q_i \in Q$), and the shortest path distance $d_n(q_i, r_i)$ between q_i and r_i. Since the shortest path distances $dc(C_i, r_i)$ has already been pre-computed, FS computes the dr_{C_i,q_i}, which equals $dc(C_i, r_i) + d_n(q_i, r_i)$. If $dr_{agg}(C_i, Q)$ is larger than kth_so_far, it means that C_i may have a point which can be a result of the query. Otherwise, this cluster C_i should be pruned and FS selects the next cluster. Then, FS visits every point p_i in C_i, and computes the exact aggregate distance $d_{agg}(p_i, Q)$. If there is a $d_{agg}(p_i, Q)$ that is bigger than kth_so_far, FS insert this point in to A; and update kth_so_far. FS searches all of clusters in the bottom level of hierarchical cluster iteratively, and stops when all the bottom hierarchical clusters have been selected.

Hierarchical Search (HS). Flat search algorithm starts at a random cluster, which means it may have to scan all of the points in the worst case. In addition, calculating aggregate distances between of all clusters and query set are time-consuming, more important is that I/O cost of calculation aggregate distances for clusters and the query set Q is non-trivial and needs to be minimized. Hence, we improved our FS algorithm to implement on hierarchical clusters, which is called Hierarchical Search algorithm to minimize calculation cost of aggregate distances.

The key idea of HS is that HS maintains a priority queue to obtain the candidate set and adapts the best-first search. The elements of PQ are clusters C_i or points p_i and sorted by their $dr_{agg}(C_i, Q)$ or $dr_{agg}(p_i, Q)$ in descending order. In the first step, HS initializes a priority queue PQ. Then HS extracts the top layer cluster of hierarchical cluster; for each cluster C_i, HS calculates their $dr_{agg}(C_i, Q)$ and pushes them into priority queue PQ. During second phase, HS pops elements from PQ iteratively. For each element, HS compares its $dr_{agg}(C_i, Q)$ or $dr_{agg}(p_i, Q)$ with kth_so_far, if the $dr_{agg}(C_i, Q)$ or $dr_{agg}(p_i, Q)$ is larger than kth_so_far, this element is selected as a candidate element, therefore HS goes to next step. In this step, if this candidate element is a cluster that contains only one point (i.e., size of cluster is 1) or is a point, and then HS inspects this point p_i in this candidate cluster to calculate aggregate distance $d_{agg}(p_i, Q)$. If not, HS extracts this cluster to get its child clusters/points set, for each cluster C_i or point p_i in child clusters set, HS calculates $dr_{agg}(C_i, Q)$ or $dr_{agg}(p_i, Q)$ and push these elements to PQ with their aggregate distances as new candidate elements. After this, HS returns to the beginning of the second loop and continue popping the elements until $|A| = k$.

Given an example, at beginning, HS calculates cluster aggregate distance for top level clusters C_6, C_5 and push them to PQ, and elements in PQ are $\{(C_5, 8), (C_6, 4)\}$. Then first element $(C_5, 8)$ is popped, since kth_so_far is smaller than C_6's aggregate distance and the number of points in C_5 are larger than one. Then, HS gets C_5's children clusters, which is C_3 and C_4, and calculates their aggregate distance $dr_{agg}(C_3, Q)$ and $dr_{agg}(C_4, Q)$, push them into PQ. After that, the elements in PQ are $\{(C_4, 8), (C_6, 4), (C_3, 3)\}$. Similar with previous step, C_4 is popped, and elements in PQ are $\{(p_7, 9), (p_8, 8), (p_6, 6), (C_6, 4), (C_3, 3)\}$. Continually, point p_7 is popped, however p_7 is already a point, hence we calculate aggregate distance between p_7 and Q, and we get $d_{min}(p_7, Q) = 8$. After that, kth_so_far is updated to 8 with p_7. Return to previous step, p_8 is popped from PQ. The kth_so_far is not less than $dr_{min}(p_8, Q)$ and $|A| = k$, therefore, HS is stopped and report point p_7 as the result of that query.

4 Experimental Evaluation

In this section, we conduct extensive empirical evaluation based on real world POI dataset to verify the superior of our proposed solution. All the algorithms are implemented in Java and running on a PC with Intel 2.13GHz CPU and 4GB memory.

4.1 Experimental Setup

Road Network. The road network dataset used in our evaluation is Beijing road network, which contains 106,579 road segment nodes and 141,380 road segments.

Point Set. We use Beijing POI dataset in our experiment as point set. This dataset contains more than 500K POIs. We align each POI onto the road network to get its network location.

4.2 Evaluation Approach

We proposes three search algorithms, i.e, *exhaustive search* search, FS and hierarchical search HS. The performance, i.e., IO cost and execution time, of these search algorithms is affected by several parameters, such as number of reference points (denoted as n), number of query points (denoted as m) and number of requested results (denoted as k). Table 2 lists the default value and range of all these issues we used throughout the experiments. It is noteworthy that we adopt the variable-control method in the experiments that in each experiment set only one issue is adjustable and the rest issues are fixed to their default values.

Table 2. Parameter Settings

Parameter	Default value	Range
Number of grid cells	300	25, 50, 100, 200, 300, 400
Number of query points	9	5, 7, 9, 11, 13, 15
Number of requested results	10	5, 10, 15, 20, 25, 30

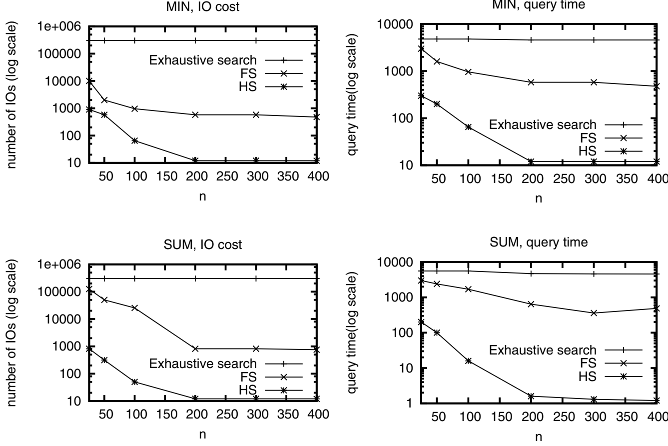


Fig. 3. Effect of number of grid cells

4.3 Evaluation Results

Effect of the Number of Grid Cells In this set of experiments, we evaluate how the performance of the algorithm is affected by number of grid cells. As shown in Figure 3, for each algorithm, the performance to solve *min* and *sum* aggregate function is similar. For both FS and HS algorithm, the IO cost and time cost decrease rapidly with number of grid cells growing from 25 to 200. The reason is that the more grid cells (i.e. more reference points) are deployed on the road network, the more closely query point q is to reference point r, which leads that $dr(p, q)$ is closer to $d_n(p, q)$. The closer $dr(p, q)$ to $d_n(p, q)$, in other words, is a tighter upper bound. Thus the performance of algorithms are improved very much. On the other hand, after deploying more than 200 grid cells, the increasing rate of performance becomes smoothly. This is because once the number of grid cells have reached a certain amount, the influence of adding more reference points is weak.

The last algorithm HS dominates FS under all value of n in both IO cost and query time aspects. This is because HS uses both hierarchical cluster structure and priority queue to prune unnecessary clusters during searching while FS not. Thus, the IO cost of HS is very low since it only need to search few clusters to find the answer of AkFN query, which is verified in this experiment. Meanwhile, this experiment also illustrates that using priority queue to sort the maximum aggregate distances can reduce query time obviously.

Effect of the Number of Query Points. Fig 4 demonstrates how the number of query points affects the effect of these three searching algorithms. We compare the three algorithms by tuning the number of query points from 5 to 15 with the step of 2. The IO cost of exhaustive search algorithm is same with previous experiment, which is still kept at a high level. The query time of exhaustive search algorithm increases rapidly due to calculating the aggregate distance becomes

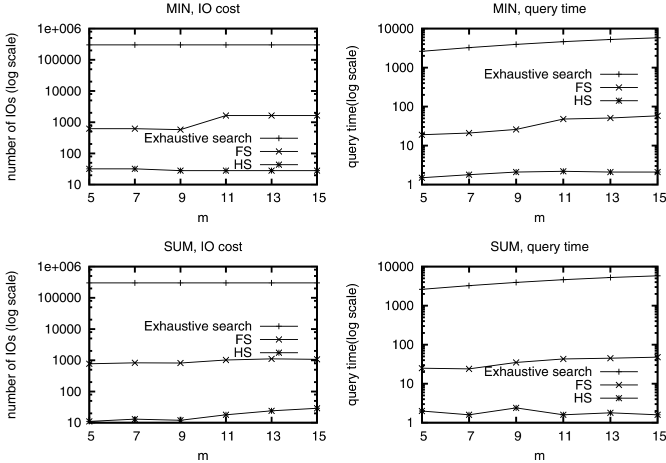


Fig. 4. Effect of number of query points

more complex when the number of query points is bigger. The FS algorithm is stable for *sum* aggregate function, but grows fast on *min* aggregate function. This is because the *min* aggregate function needs to find the maximum minimum aggregate distance while the *sum* aggregate function not, which means the bound of *min* aggregate function is looser than the *sum* aggregate function. HS shows the best performance for both the *min* and the *sum* aggregate functions in both IO cost and query time aspects. The reason is that the clusters in priority queue are sorted by maximum aggregate distance, thus most of unnecessary hierarchical clusters are pruned before HS searching in it. Hence no matter how many query points are given, it will not affect the efficiency of HS very much.

Effect of the Number of Requested Results. In this set of experiments, we evaluate how the performance of the algorithm is affected by number of requested results. The result is shown in Fig 5. We compare the IO cost and the query time of these three algorithms by tuning the number of requested results from 5 to 30 with the step of 5. The IO cost and query time of exhaustive search algorithm is similar with last two experiments, since it need to search the whole dataset. The performance of FS is stable except IO cost in *min* aggregate function, the reason is that the *min* aggregate function needs to find the maximum minimum aggregate distance. The IO cost of HS for both *min* and *sum* aggregate functions increases quickly, since HS needs to search more clusters in hierarchical structure to get top-k results, but its IO cost increases stable with the growing of the number of the requested points. However, the query time of HS is very stable with different k value since the HS maintains the hierarchical cluster structure and the priority queue. Hence HS can quickly get top k records from that priority queue during searching the hierarchical cluster. Finally, HS still outperforms other two algorithms in this experiment set.

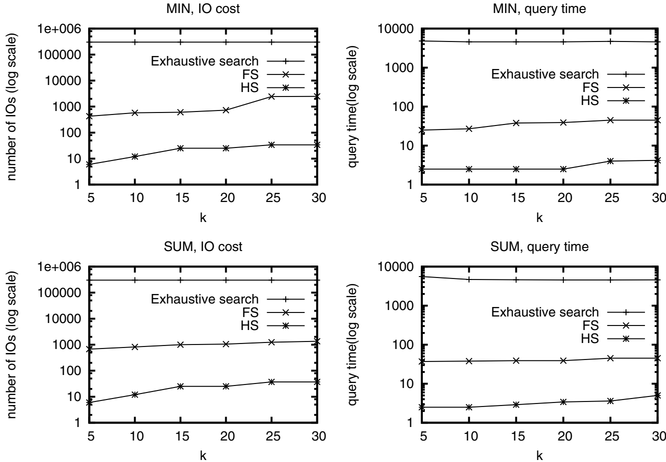


Fig. 5. Effect of number of requested results

5 Related Work

The problem investigated in our paper is combined with aggregate function and farthest neighbor search. We review the previous work related to these two categories in this section.

5.1 Aggregate Nearest Neighbor (ANN) Query

Papadias et al. [3] studied earlier version of ANN query, which is called GNN query. Three algorithms, which are called MQM, SPM and MBM, are proposed in their work to solve GNN query. Papadias et al. extended their work [5] and applied these algorithms to ANN query for solving *min* and *max* aggregate functions. There are several different types of GNN query have been recently studied in [10]. Yiu et al. studied the ANN query in road networks [2] from Papadias et al [1]'s work. They proposed three algorithms, which are called IER, TA and CE, to process *sum* and *min* aggregate functions with all data objects are index by R-tree.

All the above work focus on NN query, which are not applicable to our aggregate farthest neighbor query settings, where our query target to the farthest neighbor on the road network.

5.2 Farthest Neighbour (FN) Query

Yao et al. defined reverse farthest neighbour (RFN) query [11] problem. They proposed furthest Voronoi diagram based algorithms, which are called progressive farthest cell (PFC) algorithm and convex hull farthest cell (CHFC) algorithm to process RFN query with R-tree indexing. Moreover Tran et al. [12]

studied top-k RFN query by using Network Voronoi Diagram (NVD),However, these works only focus on the reverse farthest neighbour problem, where are not applicable to solve AkFN query.

Gao et al. study aggregate farthest neighbour query [6] in Euclidean space, and they the minimum bounding (MB) algorithm and best first (BF) algorithm, which all use R-tree [7] based indexing method. The main idea of their algorithms is using the maximum distance between query set and R-tree node as an upper bound to prune unnecessary nodes. However, it is different with our work, since our AkFN query is processed on the road network.

6 Conclusion

In this work, we investigate the AkFN problem on road network with min and sum aggregate functions. We have proposed two algorithms, flat search and hierarchical search, to associate with hierarchical clustering and pre-computing methods. Finally, the experimental results show that the flat search algorithm and the hierarchical search algorithm boost the searching efficiency compared with naive algorithm.

Acknowledgement. This research is partially supported by Natural Science Foundation of China (Grant No.61232006), and the Australian Research Council (Grants No. DP110103423, No. DP120102829 and No. DE140100215).

References

1. Papadias, D., Zhang, J., Mamoulis, N., Tao, Y.: Query processing in spatial network databases. In: PVLDB. VLD 2003, pp. 802–813 (2003)
2. Yiu, M., Mamoulis, N., Papadias, D.: Aggregate nearest neighbor queries in road networks. TKDE 17(6), 820–833 (2005)
3. Papadias, D., Shen, Q., Tao, Y., Mouratidis, K.: Group nearest neighbor queries. In: ICDE, pp. 301–312 (2004)
4. Cheong, O., Su Shin, C., Vigneron, A.: Computing farthest neighbors on a convex polytope. Theoretical Computer Science 296(1), 47–58 (2003)
5. 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)
6. Gao, Y., Shou, L., Chen, K., Chen, G.: Aggregate farthest-neighbor queries over spatial data. In: Yu, J.X., Kim, M.H., Unland, R. (eds.) DASFAA 2011, Part II. LNCS, vol. 6588, pp. 149–163. Springer, Heidelberg (2011)
7. Guttman, A.: R-trees: A dynamic index structure for spatial searching. In: SIGMOD 1984, pp. 47–57 (1984)
8. Yiu, M.L., Mamoulis, N.: Clustering objects on a spatial network. In: SIGMOD 2004, pp. 443–454 (2004)
9. Jagadish, H.V., Ooi, B.C., Tan, K.L., Yu, C., Zhang, R.: idistance: An adaptive b+-tree based indexing method for nearest neighbor search. ACM Trans. Database Syst. 30(2), 364–397 (2005)

10. Xu, H., Li, Z., Lu, Y., Deng, K., Zhou, X.: Group visible nearest neighbor queries in spatial databases. In: Chen, L., Tang, C., Yang, J., Gao, Y. (eds.) WAIM 2010. LNCS, vol. 6184, pp. 333–344. Springer, Heidelberg (2010)
11. Yao, B., Li, F., Kumar, P.: Reverse furthest neighbors in spatial databases. In: ICDE, pp. 664–675 (2009)
12. Tran, Q.T., Taniar, D., Safar, M.: Reverse k nearest neighbor and reverse farthest neighbor search on spatial networks. In: Hameurlain, A., Küng, J., Wagner, R. (eds.) Trans. on Large-Scale Data- & Knowl.-Cent. Syst. I. LNCS, vol. 5740, pp. 353–372. Springer, Heidelberg (2009)