Approximate ANN on road networks

Saranya Sadasivam
Computer science and Software engineering
Auburn University
Auburn, Alabama, USA
szs0092@auburn.edu

Abstract— **Aggregate nearest neighbor queries return a point or a set of points that satisfy a distance constraint from a set of query points. For example, consider a company that wants to know which warehouse (data points) would yield the minimum total distance trucks from all the branches have to travel in order to meet. A fast but approximate algorithm called approximate aggregate nearest neighbors (AANN) is proposed so as to process such queries on road networks based on pre-computed data. By focusing on reducing the amount of pre-computed data stored and efficient ways to use them during query time, the algorithm aims to be computationally faster at the cost of being approximate. Experiments on three road network data sets demonstrate the impact of input parameters on the query processing time. It was observed that the query processing time was reduced by up to 10 times while the minimum deviation from the ground truth was approximately 12%.**

Index Terms—**Spatial database, aggregate nearest neighbor**

I. Introduction

Sometimes it is necessary to find a place that is equidistant or at a minimum net distance from two or more places. For instance, imagine a group of friends who live in different areas of a city and plan to meet at a common place. They would each like to do this by travelling equal distances. Therefore, the solution to this problem is a place on the road network that is equidistant from all the friends' houses. Similarly, imagine another situation where a company wants to distribute its goods from a warehouse to two other branches in minimum time possible. For this, it needs to identify a place at which the warehouse truck will meet with the branch trucks leading to minimum delivery time. The solution to this problem is a place that has minimal net distance from the warehouse and branches

Such queries that are spatial in nature are called Aggregate Nearest Neighbor queries. They are a special class of spatial queries that involve finding the nearest neighbor based on multiple query points. They are expected to return a data point or a set of data points (points of interest) that satisfy some net distance constraints. Mathematically, given a set P of data points, a set Q of query points, and an aggregate function *f()* (e.g., minimum sum, equidistant) an Aggregate Nearest Neighbor (ANN) query retrieves the object p in P, such that *f()* is satisfied.

In this study, an algorithm is proposed which answers aggregate nearest neighbor queries faster by using an assortment of optimization techniques. Although the result is approximate, the time taken to process a query is considerably reduced. Of the many optimization techniques, the technique of pre-computation is used. The algorithm pre-computes and stores relevant data that may be required at query time. Since multiple queries use the same pre-computed data, there is reuse of information which conserves space and time. Many algorithms have already been proposed which are based on pre-computation [3, 4, 5, 6]. A common observation in these studies is that pre-computation comes with a high cost with respect to computational speed and storage requirements. This is due to a vast amount of data that must be stored as well as iteratively accessed and retrieved from the disk to the main memory. These factors significantly affects the performance of query processing on spatial network.

However, by accessing the network distances of a subset of network points on the road network, the amount of data storage requirement can be greatly reduced. By selectively storing only some network distance data and by computing query results based on such pre-computed data, the space and time complexity of the proposed algorithm can be significantly reduced. This is the motivation to use pre-computation on a subset of the data, in the proposed algorithm. The algorithm, like a few other [7, 8] also uses approximation in terms of network distance to reduce the query processing time.

The proposed algorithm can be applied to only answer a query containing three query points. The rationale behind this assumption is that the geometric properties of the triangle formed by the query points can be exploited to make the algorithm faster. For instance, given three points that form a triangle, a Fermat point is a point on the triangle whose net distance is the least. Therefore, once such a point is found on a triangle, assuming that the given road network graph is approximately uniform, the result to ANN queries can be found near the Fermat point. This is the motivation to initially compute the Fermat point or the circumcenter of the triangle formed by the query points and then proceed to find the nearest neighbor from this point in the proposed algorithm.

Another optimization technique used is a grid based structure for spatial division. The grid is used to store network points that fall within the bounds of each grid cell. By using this technique network points present in a large spatial region can be stored and accessed with ease and efficiency. For instance, imagine a road network which covers a large expanse of region with no data point. Such empty spaces that are of no particular value only increase the processing time. For such regions, the proposed algorithm does not store any pre-computed data, thereby saving on storage space and processing time.

II. Related work

Many algorithms have been proposed which process aggregate nearest neighbor queries using Euclidean distance [1, 2]. Papadias et al [1] have widely experimented and proposed

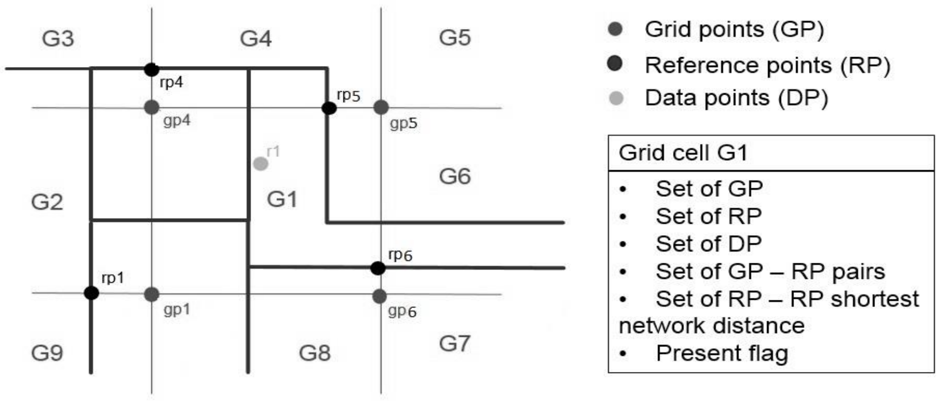


Fig. 1. Pre-computation based on rectangular grids

various algorithms that used efficient spatial data structures such as R Trees and exploited geometric properties. Two such algorithms are the MQM (Multiple Query Method) and the SPM (Single Point Method). In MQM, nearest neighbor search is done from all the query points. As the neighbors are discovered, their aggregate distance to all the query points are calculated and compared with the aggregate distance of the already discovered neighbors, so as to find the neighbor with minimum aggregate distance. In this method, however, a lot of neighbors need to be discovered and checked to reach the optimal point. A better algorithm is the SPM, in which geometric properties of the convex hull formed by the query points is exploited to find a centroid. Once found, the nearest neighbors are discovered from the centroid, instead of the query points. This makes the search for the optimal point faster as it will be located near the centroid.

Although such algorithms cannot be applied on road networks where object movements are constrained on the network, they motivate the use of Euclidean distance on road networks to guide the search for the desired points. For instance, the proposed algorithm adapts the SPM (Single Point Method) so as to find a point on road networks which may yield more probable ANN query results.

The research of Yiu et al [2], is also prominent when it comes to ANN queries on road networks. They introduced and solved ANN query processing in the context of large road networks. They proposed an algorithm that uses Euclidean distance to guide the search for data points, while computing the shortest network distance from every data point found, in parallel. They have explored the efficiency of their algorithms in the presence of data structures that materialize shortest path distances between network nodes. But, dealing with the raw spatial network data still results in a large storage overhead.

III. Problem definition

The problem definition is as follows: Given a set P of data points, a set Q of three query points, an aggregate function *f()* and a network graph G, the proposed algorithm aims to find the object p in P, that satisfies *f()*, as fast as possible and with less storage overhead.

The proposed algorithm comes with some assumptions. Firstly, only three query points can be processed. Secondly, it assumes that the road network is bi-directional. Thirdly, all the data points and query points are assumed to be on the road network. Except the first, the rest of the assumptions go along with real life road networks. For applications with more than three query points, the algorithm can be adapted with minor changes to it.

IV. Algorithm design

The proposed algorithm, AANN, addresses the shortcomings of existing pre-computation techniques. AANN addresses the issue of large query processing time by using a grid based approach to segregate network points on the road network. The second issue of higher storage requirements is tackled by storing information related to only a subset of network points. Finally, the issue of large query processing time is addressed by using simple geometric properties that lead to an optimal result efficiently. The description of the three techniques are described further below.

A. Grid based pre-computation

To make the processing of an aggregate nearest neighbor query faster, AANN uses the technique of grid based pre-computation. During the pre-computation phase, shortest paths between selected network points (called as reference points) are computed and associated with the grid cell they lie in. By using rectangular grids of certain resolution as shown in fig. 3 (a), the shortest paths can be stored in the corresponding grid cell. During query time, these grid cells can be traversed through by changing the row and column indices and the pre-computed shortest paths can be easily accessed.

To reduce the amount of pre-computed data stored and accessed, the number of network points based on which pre-computation is done, is kept to a fixed number per grid cell. AANN chooses four reference points that is paired up to each grid point of the cell. Therefore, during pre-computation, shortest paths between these reference points are computed. Fig.1 shows the grid based pre-computation approach that is taken by the proposed algorithm. As shown, for a road network, grid cell G1 has four grid points, four reference points, the grid point-reference point association, a set of data points located within the grid cell, the pre-computed shortest paths between reference points and a present flag that indicates if a data point is present in the cell.

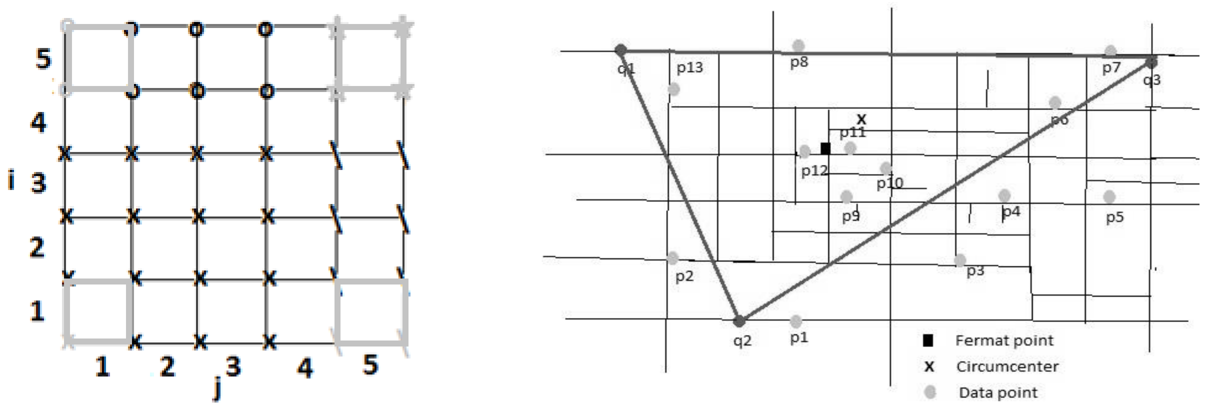


Fig. 3. (a) In the 5X5 grid, one, two or four grid point (in gray) is associated with each grid cell based on their location in the grid so as to avoid redundant computations. Grid cell [2, 2] will have lower left corner grid point (shown by ‘x’) associated with it while [3, 5] will have lower and upper left grid points (shown by ‘o’) associated with it. Similarly, [5, 5] will have all four grid points (b) Intermediate points of the triangle (convex hull) formed by the query points

By associating four reference points to each grid cell and then computing shortest paths between them, redundant computations occur. This can be addressed by associating specific number of grid points per grid cell according to their location in the grid as shown in fig 3 (a). For grid cells that are not right most (j = 5) or top most (i = 5), the grid point at lower left corner is associated with the grid cell (shown by ‘x’). For the right most, the two grid points at the bottom are chosen while for the top most, the left lower and upper grid points are chosen. Finally, the cell in the top right (i = 5, j = 5) will have all four grid points. The pre-computation phase, after such an arrangement yields to fewer computations that can later be replicated.

The reference points are computed as the network points that are closest to the grid points. It is evident that by using approximation, the algorithm only has to store a fixed set of reference points per grid cell. Also, depending on the resolution of the grids, the amount of approximation can be controlled. Fig. 2 shows the algorithm to pre-compute data.

```
Algorithm PreCompute (R, D, r, c)
Input: R is the road network adjacency list, D is set of data
points and r, c are number of rows and columns in grid
1. Find minimum (min) and maximum (max) bounds of R
2. Compute grid cell size based on r, c, max and min
3. Distribute and associate network points from R to grid cells
4. Distribute and associate data points from D to grid cells
5. For each grid cell,
6.     Associate appropriate grid points
7.     If data points are associated with grid cell then,
8.         Present flag is set to number of data points
9.     Else, set Present flag to 0
10. For each grid point in a grid cell,
11.     Find nearest reference point based on Euclidean distance
12.     Store reference point in the grid cell
13.     Associate reference point to grid point
14. For each pair of reference points,
15.     Compute shortest path using A* algorithm
16      Associate and store paths to grid cell
17.     Associate paths to starting reference point
18. For each grid cell, replicate grid points and reference points
    derived from neighboring grid cells.
```

Fig. 2. Algorithm to pre-compute and store data

B. *Intermediate point computation*

Given three query points, the algorithm first finds an intermediate point which can be expected to be close to the final solution. When the distance constraint of the aggregate nearest neighbor query is minimum net distance, the Fermat point of the triangle formed by the three query points is computed. When the distance constraint is equal distances to the query points, the circumcenter of the triangle is found. Fig. 3 shows the triangle formed by joining the query points and the two intermediate points i.e. Fermat point and circumcenter. Depending on the distance constraints, the algorithm to calculate the corresponding point is used.

Let us assume the distance constraint is minimum net distance. Therefore, the point to be found is the Fermat point of the triangle. This point can be calculated based on two cases. In the case where an angle of the triangle is greater than or equal to 120^o, the Fermat point is that respective vertex. Otherwise, an equilateral triangle is constructed on each of two arbitrarily chosen sides of the triangle. A line is drawn from each new vertex to the opposite vertex of the original triangle. The point at which the two lines intersect is the Fermat point. Once the intermediate point is computed, the nearest data point from it is found.

C. *Usage of pre-computed data*

Once the intermediate point is found, the grid cell into which this point falls, can be easily found based on the intermediate point coordinates and the grid resolution. Pre-computed data from the grid cell is accessed to find if any data point belongs to it. This is done by checking the present flag as shown in the query processing algorithm in fig. 4.

If a set of data points are found, the one with minimal Euclidean distance to intermediate point is chosen. The shortest path to this data point from all query points is then computed in three parts. Firstly, shortest path from the query point to nearest reference point of the grid cell it belongs to is computed. This step is done using the raw network. Secondly, from this reference point, shortest path to one of the four reference points

```
Algorithm ProcessQuery (Q, G, choice)
Input: Q is the set of query points, G is pre-computed array of grid cells representing the grid, choice is the choice of distance constraint the optimal point should adhere to.
1. Find intermediate point ip based on choice
2. Find the grid cell $i$, which contains ip
3. If $present_i > 0$ then,
4.  Find nearest data point dp from ip based on Euclidean
    distance
5.  Go to step 16
6. Else, traverse through the grid in a clockwise manner
7. For each grid level $l$,
8.     For each grid cell $j$ in $l$,
9.       If $present_j > 0$ then,
10.          Continue traversing one more level $(l + 1)$
11.          Add data points to priority queue dplist
12. Remove the minimum element dp from dplist
13. For each query point qp in Q,
14.     Using A* algorithm find,
15.         Shortest path sp1 from qp to nearest reference
            point rp of the current grid cell using raw road network
16.         Shortest path sp2 from rp1 to one of four reference
            points rp2 of the grid cell that contains dp using G
17.         Shortest path sp3 from rp2 to dp using raw
            road network.
18.     Concatenate sp1, sp2, sp3 as shortest path for query q
19. Return dp and shortest path for all three queries
```

Fig. 2. Algorithm for processing query

of the grid cell that contains the data point is computed. This step is done using only pre-computed data and is a grid based traversal. Thirdly, shortest path from the reference point found in the second step to the data point is found. This is done on the raw network. As a reference point to reference point traversal is used to find the shortest path, a major chunk of computation is already done and efficiently stored within the grid. This significantly contributes to speeding up of the algorithm.

When the current grid cell does not have any data point, a clock wise traversal is done on the grid. This is done by manipulating the grid cell indices and checking the value of present flag of the grid cell in parallel. Since only one value needs to be checked to prune search space, the algorithm is fast. The clockwise traversal will occur until a grid cell is found with a nonzero present flag value. At this point, one more grid level traversal occurs so as to account for all data points within a distance range. Similar to the earlier case, the data point with the least Euclidean distance to the intermediate point is chosen and shortest path from all the query points to the data point is computed. Finally, the data point and the shortest paths are returned.

V. Experimental Results

This section contains a report on a systematic empirical study conducted to evaluate the performance of AANN as compared to ANN. All experiments were performed on a Dell Precision T3400 PC with Intel® Core ™ 2 Duo CPU at 2.83 GHz, with 4 GB RAM and running on Windows XP. The algorithm was implemented in Java and JavaGeom library was used for computing the intermediate points.

A. Experimental setup

Experiments were performed to compute the aggregate nearest neighbor with minimum net distance and find the average computation times and average path distances for both ANN and AANN. Along with these, experiments to find AANN performance in terms of normalized path distance difference, speed up and pre-computation time were also performed. As the AANN results depend on the data set that it is applied to, three different real world road networks of cities were chosen for the experiments (available at http:www.cs.fsu.edu/~lifeifei/SpatialDataset.htm). These three road networks are California road network (21,047 nodes and 21,692 edges), city of San Joaquin County (TG) road network (18,262 nodes and 23,873 edges), and city of Oldenburg (OL) road network (6,104 nodes and 7,034 edges).

B. Experimental analysis

A common observation during the empirical study was that the shortest path computation dominated the query processing time. Therefore, the experiments focus on finding the shortest path from the given three query points to a data point that is located closest to the derived Fermat point.

For the experiments, a set of thousand queries containing three query points each were randomly generated and were given as inputs to both, the ANN and AANN algorithms. The query points were chosen randomly from the set of network points on the road network. Also given as inputs where grid sizes ranging from 10X10 to 100X100. The grid could be rectangular but a square grid is used for simplicity and ease of experimentation.

To understand the comparative study of the algorithms, the metric that is described below, is used. The metrics are: 1) average computation time for AANN as $\overline{AANN_c}$, and for ANN as $\overline{ANN_c}$, 2) average speed up introduced by AANN as $\overline{AANN_{sp}}$, 3) average path distance computed by AANN as $\overline{AANN_{pd}}$ and ANN as $\overline{ANN_{pd}}$, and 4) the average normalized shortest path distance as $\overline{N_{pd}}$. The metrics are defined as follows:

$$\overline{AANN_c} = \frac{Total\ time\ taken\ by\ AANN\ to\ process\ all\ queries}{Number\ of\ queries\ processed}$$

$$\overline{ANN_c} = \frac{Total\ time\ taken\ by\ ANN\ to\ process\ all\ queries}{Number\ of\ queries\ processed}$$

$$\overline{AANN_{sp}} = Avg\left(\frac{Total\ time\ taken\ by\ ANN\ for\ query\ i}{Total\ time\ taken\ by\ AANN\ for\ query\ i}\right)$$

$$\overline{AANN_{pd}} = \frac{Total\ path\ distance\ of\ AANN\ for\ all\ queries}{Number\ of\ queries\ processed}$$

$$\overline{ANN_{pd}} = \frac{Total\ path\ distance\ of\ ANN\ for\ all\ queries}{Number\ of\ queries\ processed}$$

$$\overline{N_{pd}} = Avg\left(\frac{d_i^{AANN} - d_i^{ANN}}{d_i^{ANN}}\right)$$

where d_i is the shortest path distance for query i.

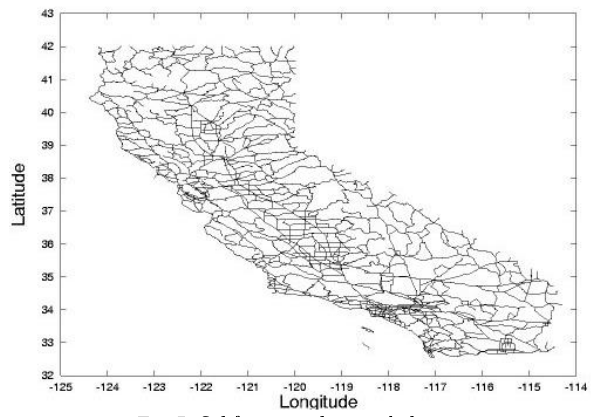


Fig. 5. California road network dataset

C. *California road network*

The California road network graph, as seen in Fig. 5, is not distributed evenly throughout the region bounding it. The road network stretches diagonally through the region bounding it. Therefore, for a query point lying near the borders of the road network, a large amount of search space can be pruned out while searching for shortest paths. This can be attributed to the fact that the search will encounter grid cells which may not contain data points in them. As only the present flag needs to be checked, a large amount of area can be ignored.

The experimental results while computing aggregate nearest neighbor queries on California road network dataset is shown in fig.6. In fig.6 (a), the average computation time of AANN as compared against ANN is shown. Clearly, the AANN outperforms ANN by about 2.5 seconds. For a grid size of 30X30, the average computation time is the least at 0.2 seconds as compared to the other grid sizes. The variation in performance among different grid sizes is due to fact that more raw network traversal needs to be performed when the grid cells are large. On the other hand, smaller grid cells lead to an increase in the amount of pre-computed data retrieved. Due to these reasons, grid sizes 10X10 and 100X100 require approximately same amount of time. It is also due to the dependence of pre-computed data to the underlying network. For a particular grid size, the pre-computed data contains favorable paths that lead to the destination faster. This is why 30X30 yields quicker output.

Fig. 6 (b) shows the average sum of the three shortest path distances computed from the three query points to a chosen data point. As shown in the figure, the lower grid size that gives raise to larger grid cells, result in shortest paths that are more deviated from the actual result as compared to shortest paths computed by larger grid sizes that give raise to smaller grid cells. In the pre-computation phase, larger grid cells store the shortest paths between reference points that are farther apart as compared to the reference points of a smaller grid cell. Due to this, the shortest paths computed by the larger grid cells will be longer than the shortest path distance computed by the smaller cells.

AANN speeds up the process of finding the shortest paths between two points as shown in fig. 7 (a). It can be observed that grid size 30X30 provides the maximum speed up of more than 16 times. Furthermore, it can be seen that even though the speed up is not as high as 30X30, the other grid sizes still give a good

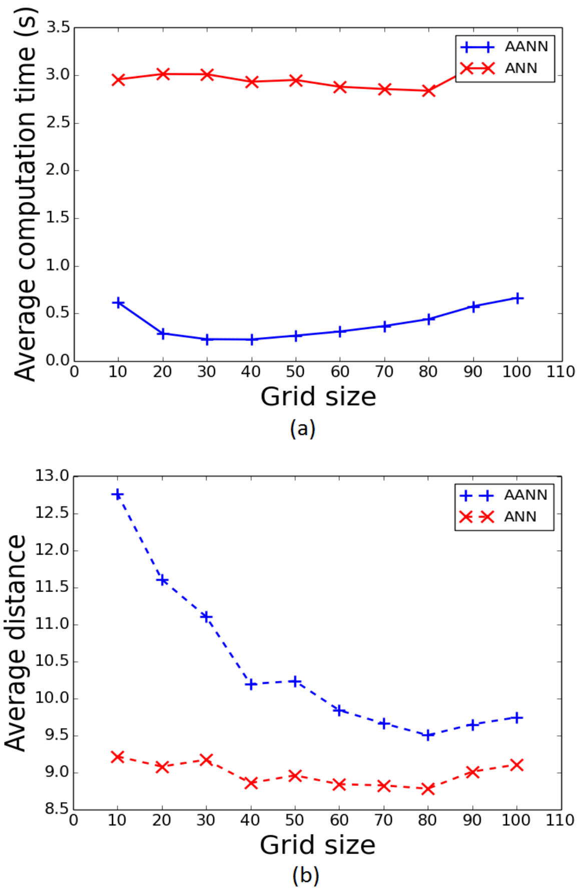


Fig. 6. Experimental results with California's road network dataset. (a) This shows how the average computation times of ANN and AANN compares against each other when grid size is varied. (b) This shows how varying grid sizes affect the average path distances of the resultant shortest path computed by ANN and AAN

amount of speed up factor. At a minimum, the speed up is little over 4 times. The variation in performance again can be attributed to the dependence of pre-computed data to the underlying road network, as discussed earlier.

Although AANN greatly reduces the amount of data stored and retrieved from the memory, it comes with a tradeoff of requiring a relatively large pre-computation time. A shown in fig. 7 (b), the pre- computation time grows linearly with grid size but stays relatively constant from 70X70 onwards. This is also relative to the data set processed.

Fig. 8 shows the normalized path distance difference between ANN and AANN resultant paths. The figure is in agreement with the notion stated earlier that smaller the grid cells in AANN, larger is the deviation of computed shortest paths from ground truth results provided by ANN. From the experimental results above, it can be seen that various grid sizes of AANN provide different benefits. While 30X30 is gives a speed up of 16 times and allows faster pre-computations, it gives rise to 20% longer shortest paths. However, using a 100X100 grid size, gives rise to 4 times speed up and shortest paths varying just 7.5% from ground truth, but at a large pre-computation cost. Therefore, AANN provides varied amount of approximation which can be configured to fit the purpose it is used for.

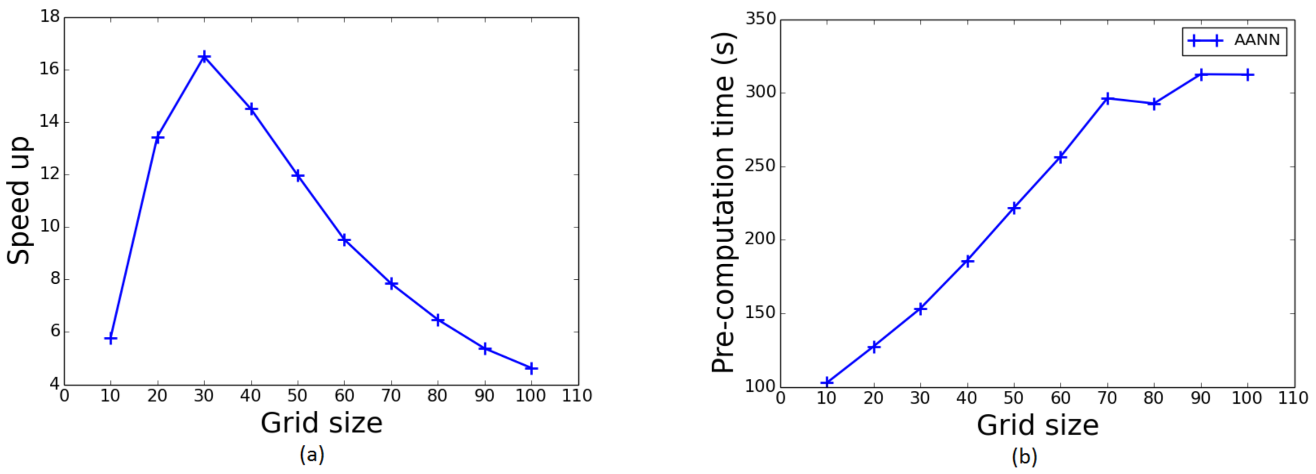


Fig. 7. Experimental results with California's road network dataset. (a) Speed up introduced by AANN is shown. (b) The effect of varying grid size on time required to pre-compute and store the shortest paths between reference points of all the grid cells is shown

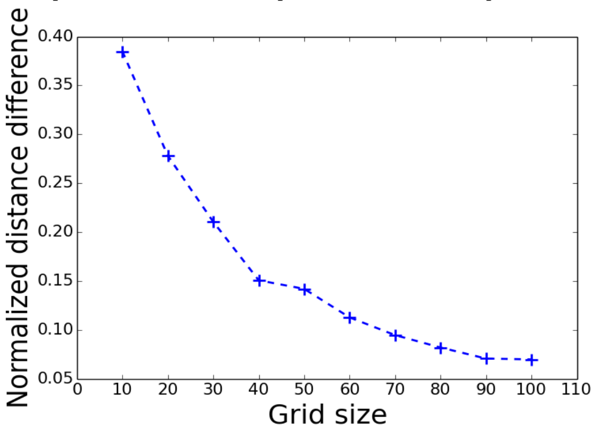


Fig. 8. Experimental results with California's road network dataset showing normalized path deviation (difference) of AANN from ANN results.

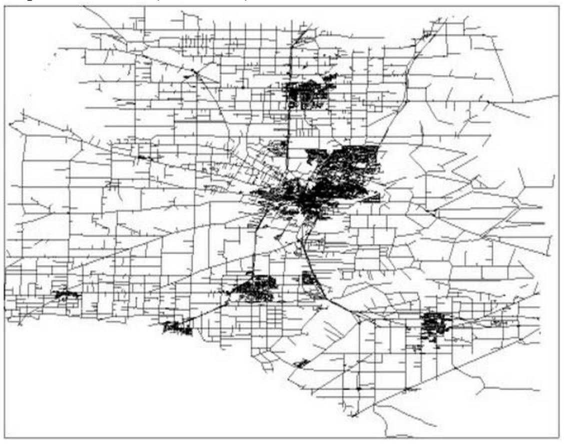


Fig. 9. City of San Joaquin County (TG) road network

D. *City of San Joaquin County (TG) road network*

As seen from fig. 9, unlike California, the San Joaquin County (TG) road network is fairly distributed within the region that bounds it. It is slightly smaller than the California dataset in terms of number of nodes and edges. Similar experiments were performed with this dataset and the results are described below. Figure 10 (a) shows that, similar to the California dataset, the average computation time taken by AANN is approximately 1.5 seconds lesser than ANN. However, smaller grid sizes require 0.5 seconds more than the California data. This is due to the lesser empty area in the region covering the road network. Since a smaller grid size leads to larger grid cells, more network points need to be traversed during the raw network traversal from a query point to nearest reference point of the grid it falls into. Therefore, 10X10 takes a longer time to process a query. Furthermore, a good range of grid sizes process a query faster as compared to the California data set. This is due to the road network being more distributed within the region.

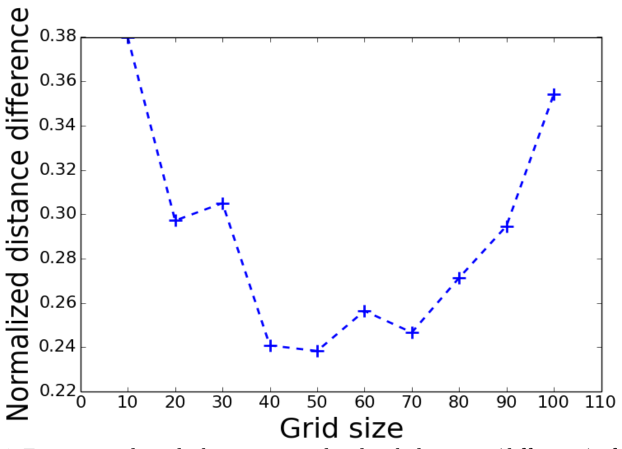
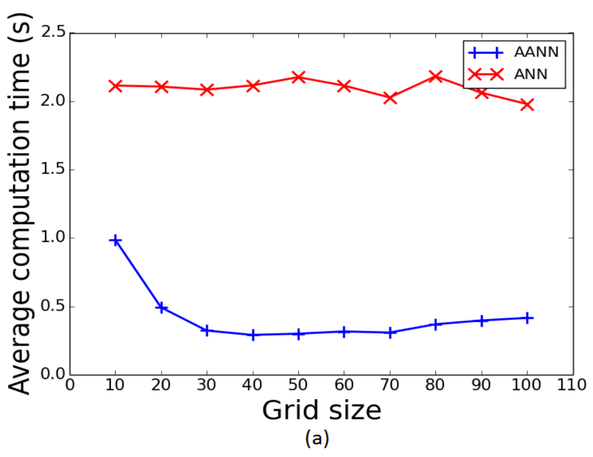


Fig.11. Experimental result showing normalized path deviation (difference) of AANN from ANN results when using TG dataset.

The average path distance of ANN and AANN in Fig 10 (b) does not converge within the grid size range. However, the normalized path distance difference in fig. 11 shows that for a few grid sizes the path distances produced by AANN is as low as 24% longer than the actual distance. As seen in fig. 12 (a), the maximum speed up provided by AANN is about 8.5 times. This is half the speed up provided by AANN on the California dataset. This is due to the reduced number of cases when a grid cell with no data point is encountered. However, similar to the California dataset results, there is a gradual decrease in the speed up as the grid size is increased after the peak value. Furthermore, a minimum speed up of 3 is observed when the grid size is 10X10.



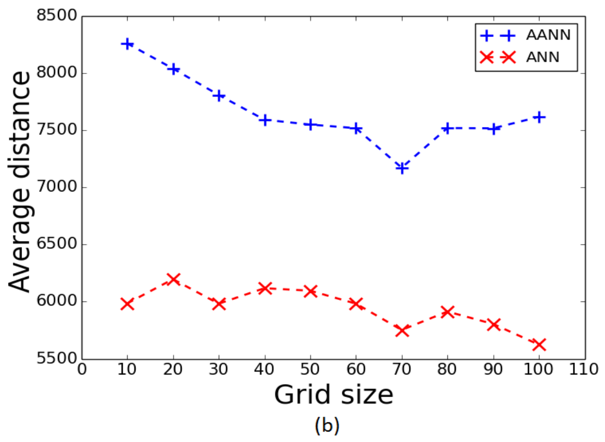


Fig. 10. Experimental results with TG road network dataset. (a) This shows how the average computation times of ANN and AANN compares against each other when grid size is varied (b) This shows how varying grid sizes affect the average path distances of the resultant shortest path computed by ANN and AANN.

Similar to the results of the California dataset, the pre-computation time as observed in fig. 12, is generally increasing with increase in grid size. This is highly dependent on the underlying road network. From 10X10 to 80X80, the pre-computation time increases due to increase in number of grid cells requiring pre-computation. But after 80X80, there is marked dip in the graph due to grid cells being too small for any pre-computation .i.e. there may not be any network point near the grid point that is eligible to be one of the four reference points of the grid cell.

E. City of Oldenburg (OL) road network

The Oldenburg road network is also fairly distributed over the region bounding it. However, it has only one third of the nodes and edges as compared to the other datasets. The scalability of AANN is evident as it provides adequate speedup with minimal approximation. The experimental results are described in fig 13, 14 and 15.

Fig. 13 (a) shows the average computation time taken by AANN and ANN when working on OL dataset. Like the other two datasets the computation time for AANN is significantly smaller than ANN. However, as the grid size is increased, the average computation time taken by AANN increases. This is due to the road network structure of the dataset. Since the dataset

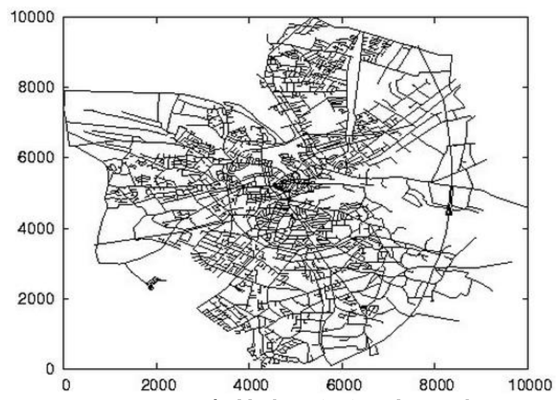
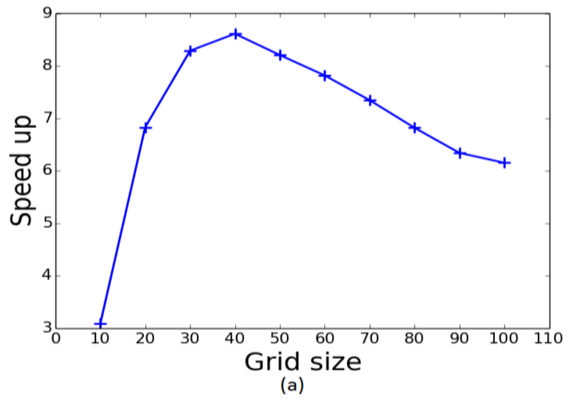


Fig. 16. City of Oldenburg (OL) road network



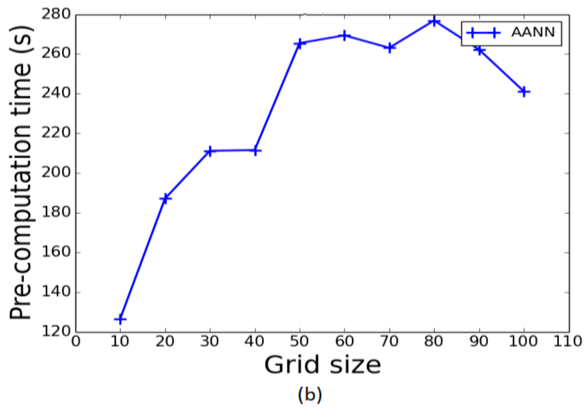


Fig.12. Experimental results with TG road network dataset. (a) Speed up introduced by AANN is shown. (b) The effect of varying grid size on time required to pre-compute and store the shortest paths between reference points of all the grid cells is shown.

has road networks that are both densely and sparsely structured, for larger grid sizes, the resultant grid cells are too small to capture any pre-computed data. Therefore, more grid cells need to be traversed to compute the shortest paths.

In fig. 13 (b), a trend similar to the California dataset result is seen where for smaller grid sizes, there is a larger deviation from ground truth value while for larger grid sizes, and there is generally a constant amount of deviation from the actual value. This is due to the increase in the amount of blank spaces in the region which is prominent in the road network structure of both the datasets. Another trend that can be noticed in fig. 13 (b) is the similarity in the position of dips and peaks as the grid sizes get larger. This clearly supports the notion that there is a constant amount of approximation provided by AANN. Fig. 14 also is in accordance with the observation stated above. With increase in grid size the results are more accurate as compared to smaller grid sizes. The minimum deviation is given by 80X80, which computes shortest paths that are 18% longer than ground truth.

In fig. 15 (a), the speed up provided by AANN in computing the query results is shown. Similar to the other two datasets, the results for this dataset exhibits a peak speed up at lower grid size values and thereafter gradually decreases with increase in grid size. For a grid size of 20X20, the maximum speed up is achieved as 7 times the time taken by ANN. The pre-computation time taken by AANN is shown in fig.15 (b).

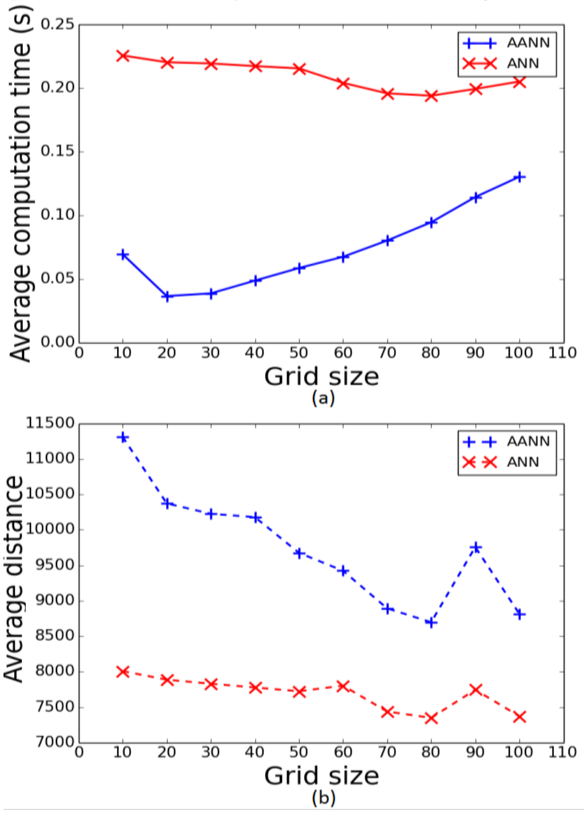


Fig.13. Experimental results with OL road network dataset. (a) This shows how the average computation times of ANN and AANN compares against each other when grid size is varied (b) This shows how varying grid sizes affect the average path distances of the resultant shortest path computed by ANN and AANN.

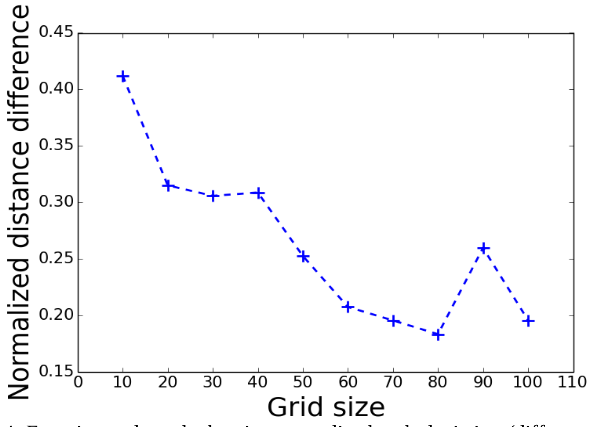


Fig.14. Experimental result showing normalized path deviation (difference) of AANN from ANN results when using OL dataset.

Similar to the results produced by the other two datasets, there is a gradual increase in the pre-computation time required as the grid size is increased and once the maximum is attained it starts to decrease. However, unlike the other two data sets, the peak occurs at a much smaller grid size.

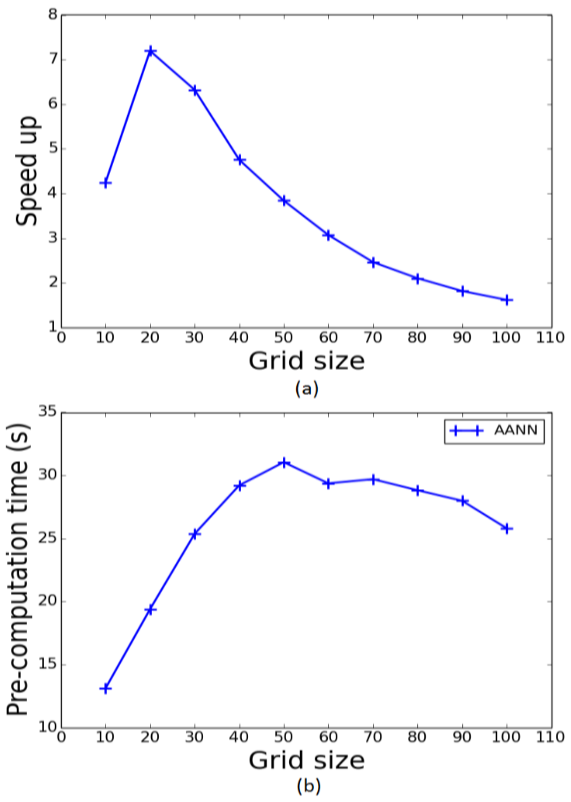


Fig.15. Experimental results with OL road network dataset. (a) Speed up introduced by AANN is shown. (b) Pre-computation time required by AANN is shown.

VI. Conclusion

For future work, the AANN algorithm can be expanded to answer queries with more than three query points. Furthermore, it can also be modified to handle datasets larger than the main memory for disk based solutions.

In conclusion, this paper presents the AANN algorithm that employs various optimization techniques so as to efficiently compute aggregate nearest neighbor queries. By applying it to different datasets and noticing similar trends, AANN is proved to be scalable for different datasets. As observed in the results, although they are dependent on the dataset used, the speed up introduced by AANN can go up to 16 times in certain cases and has a minimum value of 2 times. The approximation ranges from 40% to as low as 7.5%. Therefore, grid sizes can be configured to give rise to different benefits as per requirement.

References

[1] D. Papadias, Y. Tao, K. Mouratidis, and C. K. Hui. Aggregate Nearest Neighbor Queries in Spatial Databases. In ACM Trans. Database Syst., 2005.

[2] M. L. Yiu, N. Mamoulis, and D. Papadias. Aggregate Nearest Neighbor Queries in Road Networks. In TKDE, 2005.

[3] X. Huang, C. S. Jensen, H. Lu, and S. Saltenis. S-GRID: A versatile approach to efficient query processing in spatial networks. In SSTD, 2007.

[4] M. R. Kolahdouzan and C. Shahabi. Voronoi-based K nearest neighbor search for spatial network databases. In VLDB, 2004.

[5] X. Huang, C. S. Jensen, and S. Saltenis. The islands approach to nearest neighbor querying in spatial networks. In SSTD, 2005.

[6] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao. Query processing in spatial network databases. In VLDB, 2003.

[7] Y. Tao, J. Zhang, D. Papadias, and N. Mamoulis. An efficient cost model for optimization of nearest neighbor search in low and medium dimensional spaces. In IEEE Trans. Knowl. Data Eng., 2004.

[8] Sommer, C. Approximate shortest path and distance queries in networks. Ph.D. thesis, The University of Tokyo, 2010.

[9] Luo, Yanmin, et al. Efficient methods in finding aggregate nearest neighbor by projection-based filtering, 2007.