VII Simpósio Brasileiro de Geoinformática, Campos do Jordão, Brasil, 20-23 novembro 2005, INPE, p. 356-367.

PINE based RNN queries in Road Networks

Maytham Safar and Ahmad Al-Saleh[1]

[1]Computer Engineering Department – Kuwait University
Kuwait

maytham@eng.kuniv.edu.kw, abo_othman_@hotmail.com

Abstract. *A common query type in spatial networks is to find the Reverse Nearest Neighbors "RNN" of a given query object. Due to the difficulties and the expense of the required calculations to answer such a query, almost all approaches in the literature use Euclidean distances and not the real network distance. In this work, we propose an approach that is based on a real network distance computation to answer the RNN query. In addition, we distinguish between four different types of queries that could rise by using two types of objects such as a road intersection (e.g., a location of a vehicle) or an interest point (e.g., a restaurant.) Our approach is based on manipulating the network Voronoi diagram properties and applying a progressive incremental network expansion for finding the polygon inner network distances required to solve an RNN query.*

1. Introduction

Over the last decade, due to the rapid developments in information technology (IT), particularly communication technologies, a new breed of information systems has appeared such as mobile information systems. Mobility is perhaps the most important market and technological trend within information and communication technology. Mobile information systems will have to supply and adopt services that go beyond traditional web-based systems, and hence they come with new challenges for researchers, developers and users.

One of the well-known applications that depend on mobility is the car navigation system, which allows drivers to receive navigation instructions without taking their eyes off the road. Using a Global Positioning System (GPS) in the car navigation system enables the driver to perform a wide manner of queries, from locating the car position, to finding a route from A to B, or dynamically selecting the best route in real time. One of the interesting queries used in such systems is reverse nearest neighbors (RNN) queries. This type of query is defined as: given a set of spatial objects (or points of interest, e.g., restaurants), and a query point (e.g., vehicles' location), find the restaurants that consider that vehicle as their nearest neighbor. Another query business owners ask when deciding to open a new grocery store is "where is the best location for the grocery store?". The RNN query can restructure this question as "How many buildings consider this possible location as the nearest grocery store?". In this case each location should start RNN query and the results are then compared for a better decision.

With spatial network databases (SNDB), objects are restricted to move on pre-defined paths (e.g., roads) that are specified by an underlying network. This means that the shortest network distance between objects (e.g., the vehicle and the restaurants) depend

on the connectivity of the network rather than the objects' location. In [Safar (2005)] we showed that the majority of the existing work on nearest neighbor (NN) queries are based on either computing the distance between a query and the objects on-line, or utilizing index structures. The solution proposed by the first group is based on the fact that the current algorithms for computing the distance between a query object q and an object O in a network will automatically lead to the computation of the distance between q and the objects that are (relatively) closer to q than O. The advantage of these approaches is that they explore the objects that are closer to q and compute their distances to q progressively. However, the main disadvantage of these approaches is that they perform poorly when the objects are not densely distributed in the network since then they require a large portion of the network to be retrieved for distance computation. The second group of approaches is designed and optimized for metric or vector spatial index structures (e.g., m-tree and r-tree, respectively). The approaches that are based on metric index structures require pre-computation of the distances between the objects and grouping of the objects based on their distances to some reference object (this is more intelligent as compared to a naive approach that pre-computes and stores distances between all the object-pairs in the network). These solutions filter a small subset of a possibly large number of objects as the candidates for the closest neighbors of q, and require a refinement step to compute the actual distance between q and the candidates to find the actual nearest neighbors of q. The main drawback of applying these approaches on SNDB is that they do not offer any solution as how to efficiently compute the distances between q and the candidates. Moreover, applying an approach similar to the first group to perform the refinement step in order to compute the distance between q and the candidates will render these approaches, which traverse index structures to provide a candidate set, redundant since the network expansion approach does not require any candidate set to start with. In addition to this drawback, approaches that are based on vector index structures are only appropriate for spaces where the distance between objects is only a function of their spatial attributes (e.g., Euclidean distance) and cannot properly approximate the distances in a network.

Taken into consideration that Mobile devices are usually limited on memory resources and have lower computational power, in [Safar (2005)] we proposed a novel approach that reduces the problem of distance computation in a very large network, into the problem of distance computation in a number of much smaller networks plus some online "local" network expansion. The main idea behind that approach, termed Progressive Incremental Network Expansion (PINE), is to first partition a large network into smaller/more manageable regions. We achieved this by generating a network Voronoi diagram over the points of interest. Each cell of this Voronoi diagram is centered by one object (e.g., a restaurant) and contains the nodes (e.g., vehicles) that are closest to that object in network distance (and not the Euclidian distance). Next, we pre-compute the inter distances for each cell. That is, for each cell, we pre-compute the distances across the border points of the adjacent cells. This will reduce the pre-computation time and space by localizing the computation to cells and a handful of neighbor-cell node-pairs. Now, to find the k nearest-neighbors of a query object q, we first find the first nearest neighbor by simply locating the Voronoi cell that contains q. This can be easily achieved by utilizing a spatial index (e.g., R-tree) that is generated for the Voronoi cells. Then, starting from the query point q we perform network expansion on two different scales simultaneously to: 1) compute the distance from q to its first nearest neighbor (its Voronoi cell center point), and 2) explore the objects that are close

to q (centers of surrounding Voronoi cells) and compute their distances to q during the expansion.

At the first scale, a network expansion similar to Incremental Network Expansion (INE) [Papadias (2003)] is performed inside the Voronoi cell that contains q (VC(q)) starting from q. To this end, we utilize the actual network links (e.g., roads) and nodes (e.g., restaurants, hospitals) to compute the distance from q (e.g., vehicle) to its first nearest neighbor (the generator point of VC(q)) and the border points of VC(q). When we reach a border point of VC(q), we start a second network expansion at the Voronoi polygons scale. Unlike INE and similar to Voronoi-based Network Nearest Neighbor (VN^3) [Kolahdouzan (2004)], the second expansion utilizes the inter-cell pre-computed distances to find the actual network distance from q to the objects in the other Voronoi cells surrounding VC(q). Note that both expansions are performed simultaneously. The first expansion continues until all border points of VC(q) are explored or all kNN are found.

To the best of our knowledge, INE and VN^3 approaches presented in [Papadias (2003), Kolahdouzan (2004)], respectively, are the only other approaches that support the exact kNN queries on spatial network databases. However, VN^3 performance suffers with lower density data sets. The majority of the existing work on RNN queries are also based on computing the Euclidean distances between the query object and the object of interest, thus allowing a minimum computational time and less complex algorithms, while not providing the exact solution. Few of the proposed solutions are based on real network distances [Tao (2004), Yiu (2005)], however, they lack the support for different query types under the same structure, and the performance highly depends on the data distribution and density.

In this paper, we developed algorithms to answer the RNN queries by utilizing the same underlying system that was used by PINE to answer the kNN queries. Thus, our proposed solution is based on network Voronoi diagram and utilizes all the pre-computed distances such as the border-to-border and the generator-to-border distances to minimize the network distance computation time (see [Kolahdouzan (2004)], and [Safar (2005)] for further details.) The solution also utilizes the spatial index of the network Voronoi diagram (NVD) to check the polygons for interest data points. Leaving the only unknown network distance from the generator to any interest point inside the polygon. To overcome this, a progressive incremental network expansion (PINE) [Safar (2005)], which is based on Dijkstra's algorithm, is used for finding the inner network distance in the polygons. The network expansion occurs around the query point and expands from there to find the nearest neighbor.

Furthermore, we identified four different RNN query types that use an interest point and a vehicle (e.g., cars as the moving objects and restaurants as the static objects.) The different RNN queries are as follows:

- RNN_{Car} (Restaurant): for a given car as a query object, find the restaurants that consider the car as their nearest neighbor car.
- $RNN_{Restaurant}$ (Car): for a given restaurant as a query object, find the cars that consider the restaurant as their nearest neighbor restaurant.

- RNN $_{\text{Restaurant}}$ (Restaurant): for a given restaurant as a query object, find the restaurants that consider the query restaurant as their nearest restaurant neighbor.
- RNN $_{\text{Car}}$ (Car): for a given car as a query object, find the cars that consider the query cars as their nearest car neighbor.

2. RNN Queries

In this section we will define four different combinations of the RNN queries based on the type of the interest point and the type of the query object. Our developed algorithms to answer such queries will depend on the existence of a network Voronoi diagram (NVD) and a set of precomputed data (such as border-to-border and border-to-generator distances). The system described in our previous work [Safar (2005)] creates a set of NVDs, one for each different interest point (e.g., NVD for restaurants, hospitals, …etc.). To find the k nearest neighbors of a vehicle, PINE algorithm was used with the NVD for restaurants. In this paper, we will develop new algorithms to answer RNN queries that utilize the previously created NVDs, precomputed distances, and PINE algorithm. In the following subsections, we will assume that the restaurants are the generator points and we will use the NVD created for restaurants.

2.1. RNN $_{\text{Car}}$ (Car)

For a given car as a query object, find the cars that consider the query car as their nearest car neighbor.

One of the problems in answering such a query lies in the dynamicity of the car and that it moves in a real road network. Another challenge is that both the interest points and the query object are of the same type (i.e., vehicle). Since we have no precomputed NVD that uses the vehicles as interest points, hence, our algorithm will utilize any of the stored NVDs to solve the query (e.g., NVD for restaurants). Using a precomputed NVD helps in reducing the network distance calculation time to answer such a query. The idea is to use PINE expansion mechanism to explore all the cars that are in the vicinity of the query car (candidate cars for the answer of RNN). Once a car is reached, a 1NN query is issued using the original PINE algorithm to check if the query car is the 1NN of the candidate car or not. If yes, then the candidate car is a part of the solution. Note that we need to expand in all directions from the query point and the candidate cars. However, using the results of some lemmas we can reduce the work by stopping the expansion in some directions while expanding only in the directions where we expect to find candidates for the answer of the query.

We will use the following lemma to distinguish between two types of interest data points, one that can be reached using the network expansion and another that can be reached using both the network expansion and NVD border-to-border links.

Lemma 1: Let Q be the query point (i.e., car), R are the generator points of the network Voronoi diagram (i.e., restaurants) and P the interest data point (i.e., other cars). For any data point that lies in the same polygon or the adjacent polygon to the polygon that contains the query point, the network expansion mechanism of PINE can be used to solve RNN.

In other cases, polygons may contain an interest point but are neither adjacent to the polygon containing the query point, nor to polygons containing interest points. Those can be reached by crossing over what we call an "empty polygon" (a polygon containing no interest points), and they may contain a candidate solution. Here, we utilize the border-to-border distances provided by the network Voronoi diagram for the empty polygon from the adjacent edge of the generator polygon to the edge of the polygon that contains cars. We will also use the results of Lemma 2 in [Yiu (2005)].

Lemma 2: "*Let q be a query point, n a graph node and p a data point satisfying d(q,n) > d(p,n). for any point p'≠p whose shortest path to q passes through n, it holds that d(q,p')>d(p,p'), i.e., p'∉RNN(q).*" [Yiu (2005)].

From lemma 2 we can conclude that if the expansion started from Q and reached an intersection point, a border point of NVD, or an interest point B (i.e. car B) passing through an interest point A (i.e. car A) then point A is closer to Q than point B. Thus, for the proposed method we say that B is closer to A than it is to Q. This would limit our expansion in some directions. The expansion will continue in all directions and stop in a direction if either a car or a point that satisfies the properties in lemma 2 is reached.

For example, once a query point is initiated, the algorithm will check the current polygon for interest data points by utilizing the spatial index (e.g. R-tree). Then, the expansion mechanism of PINE is used for finding the interest points close to the query point. During this process, each found interest point will start a heap list with its distance to the query point as the initial entry, for example dist(Q,Car1). Each found interest point starts another network expansion (e.g., using PINE to solve 1NN problem from Car1) to fill up the rest of its heap list. If a new interest data point is found then its distance for example dist(Car1,Car3) is compared with dist(Q,Car1) and the expansion stops in that direction only if dist(Car1,Car3) > dist(Q,Car1). However, if dist(Car1,Car3) < dist(Q,Car1) then the entire expansion "all directions" from Car1 stops because Car3 is considered as 1NN of Car1. Hence, Car1 cannot be reverse nearest neighbor of the query. If however one of the neighboring polygons does not contain any interest data points (empty polygon), then the polygon is skipped to its adjacent polygons, by using the virtual border-to-border links. This can be illustrated by looking at P6 in Figure1. We notice that P6 is an adjacent polygon to the polygon of the query point and it contains no cars, therefore, the distances from b6 to b19, b20, b21, b22 and b23 are utilized to pass this empty polygon to its neighboring polygons. Otherwise, a new network expansion should have started in P6 using the actual road network in that polygon.

Since cars are moving objects, we have incorporated a simple mechanism that adds a time limit to the expansion process. For each nearest candidate car that is being checked, we set a time interval limit. If the time limit is exceeded before the completion of the verification then the current found answer is set to true. For example if the time finished and the heap list contain {(Q,Car1:10),(Car1,car3:12), (Car1,Car7:14)} (where :10, :12, and :14 are the distances) then Car1 is the nearest neighbor to Q. This time limit is used just to make sure that the search does not take a large amount of time to be refined and helps in handling errors.

Algorithm steps:

1- Progressive incremental expansion starts from the query point exploring for RNN candidate cars. The expansion in each path stops once a car is located in that direction.

2- Each candidate starts a heap list with the distance from the query point as the initial value.

3- Each candidate starts PINE expansion to locate its 1NN.

4- PINE will test every new distance once it reaches an intersection point, a car or a polygon border point, if that distance is larger than the initial value of the heap then the expansion stops, else the expansion continuous.

5- For cars that can be reached by passing through an empty polygon, the algorithm uses NVD border-to-border distance to the destination polygon and then starts expanding from there for finding the inner distance to the car.

6- Once a car is located in which the distance from the candidate is shorter than the initial distance then the exploration stops setting the query point as "Not my 1NN". However, if the exploration ended with no shorter distance than the initial one then the query point is the 1NN to the candidate.

For example, consider the NVD of Figure1. From the query point we start an expansion and let us assume that we reach Car1, Car3, b8, b7, b6, b2, b1. Checking P1, P3, P4, P6 we find that the polygons contain no cars, so we directly get the border-to-border distance and check their adjacent polygons for cars. This procedure can be expanded further if one or more of the adjacent cells do not contain cars, but we stop here for simplicity or we could assign a time limit for this expansion, which may give an estimate result. Now, exploring through polygon P6 could lead us to P14, which contains Car6. Once the border of P14 is reached the network expansion b22 is started to find the network distance to Car6. On the other hand, instantaneously, another network expansion starts in P2 from b1 finding Car3 and then stops. Lets say for simplicity that dist(Q,Car1:12), dist(Q,Car3:20) and dist(Q,Car6:35) are the results of this candidate interest point locating stage with their distances from the query point. Although Car5 can be reached from b2-b18 and then expanding in P7, we will not look into it because it's a similar case to Car3. Now, after all candidates are found, each car starts a heap list with the network distance to the query point as the initial entry and executes 1NN query using PINE. The search for 1NN performs the same steps again, starting with PINE finding all the interest points and border points, however, this time the expansion distance is limited by the initial value of the heap. If the expansion goes a longer distance without finding any interest points, then it stops in that direction. Therefore, in this example Car5 will never be reached because b13 distance from Car3 is larger than 20. Car3 continues the expansion to find Car4:22. Since there was neither an intersection point nor a border point after a distance of 20 that could be added to the heap, we were not able to tell earlier that Car4 is further away from the query point. The result for Car3 now clearly states that the query point is the nearest neighbor 1NN. For Car1, PINE immediately finds that Car2 is the nearest neighbor, thus the expansion stops in that direction and the result states that the query point is not the 1NN. Car6, however, starts PINE to find the distances to the polygons border points b21, b22, b23,

b24... etc. From there, using the border-to-border distance from NVD it searches for cars. The search in any direction stops when the distance to any border from Car6 exceeds the initial heap value. For this example the search will stop with no cars found in a shorter distance stating that the query car is not the 1NN. The search for RNN_{Car}(Car) is now finished with Car3 as the reverse nearest neighbor of the query point.

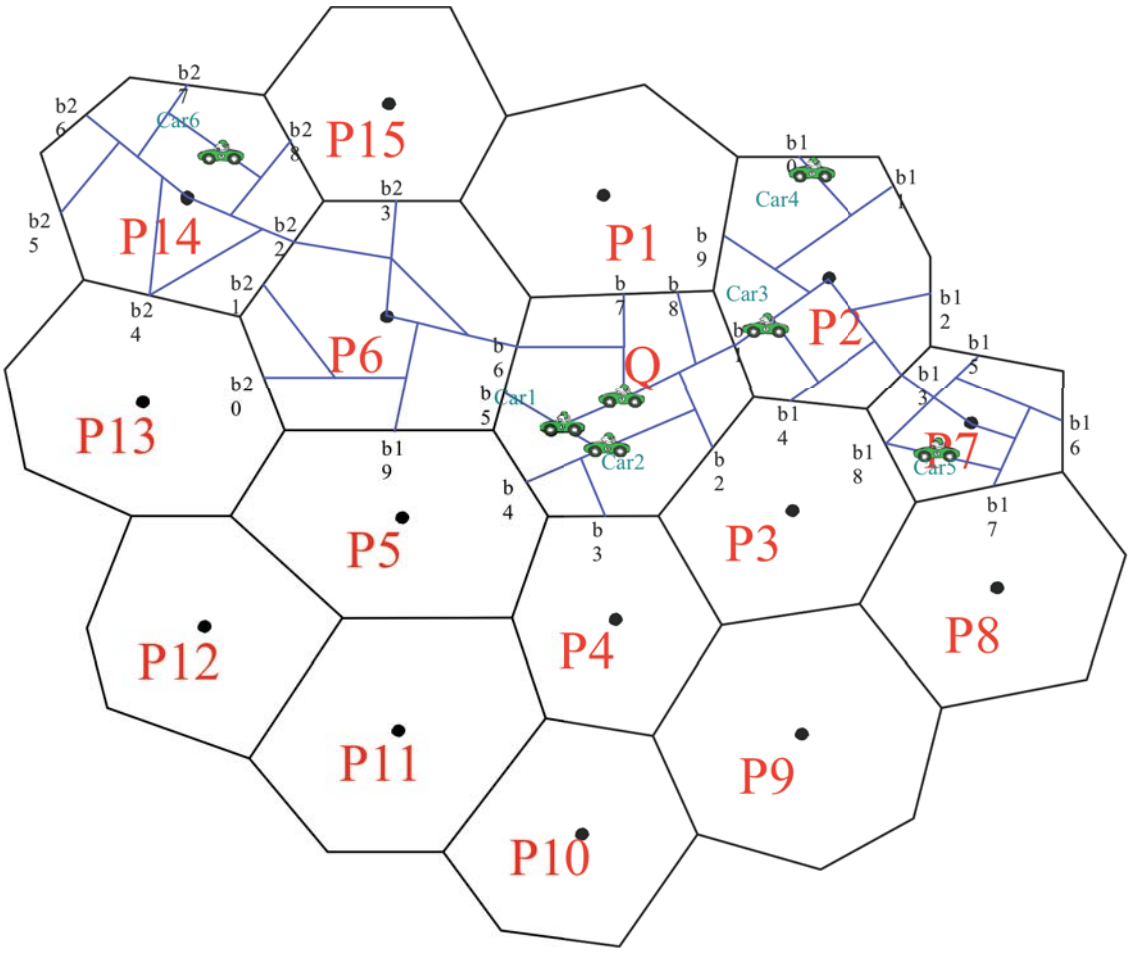

Figure 1. RNN $_{Car}$(Car)

2.2. RNN $_{Car}$ (Restaurant)

For a given car as a query object, find the restaurants that consider the car as their nearest neighbor car.

For this query type, the interest points (i.e., restaurants) already have an NVD created for them that could be used to answer the RNN query. The steps for this algorithm are similar to that used for RNN $_{Car}$ (Car), with one exception that will be mentioned next. The algorithm starts with network expansion to explore the possible restaurants that can be reached by the query car including the generator point of the polygon and the border points to neighboring polygons. However, if any of the adjacent polygons contains cars then it's eliminated from the candidate list and no expansion is performed from that polygon. It is unnecessary to go into a polygon that contains cars,

because the generator points of those polygons consider those cars as their nearest neighbors. Furthermore, from the properties of the NVD we conclude that no other polygon after that polygon could contain an answer (see [Safar (2005)] for further details). This is because, if a car is reached then all restaurants that can be reached through this path can consider this car as the nearest neighbor. For the polygons that do not contain cars we use NVD border-to-generator links to do further expansion.

Algorithm steps:

1- Progressive Incremental Expansion starts from the query point exploring for RNN candidate restaurants. The expansion in each path stops once a car or a border point of a polygon that contains cars is reached.

2- The reached candidate polygons will be checked for cars. If the polygon contains cars then it is not a candidate restaurant.

3- NVD border-to-generator point distance is used to reach the candidate generator.

4- Once the restaurants are reached then heap lists are initiated with the distance from the query car to the candidate as the first entry or the initial distance.

5- The candidates then start 1NN using PINE.

6- The distance from the generator point-to-border of the neighboring cells are found using the NVD. PINE then starts from the border point of the neighboring cells exploring for cars.

7- PINE will test every new distance once it reaches an intersection point, car or polygon border, if that distance is larger than the initial value of the heap then expansion stops in that direction, else the exploration continuous.

8- Once a car is located whose distance to the candidate is shorter than the initial distance then the exploration stops setting the query point as “Not my 1NN”. However, if exploration ended, by either expanding to a larger distance than the initial distance in the heap or by exceeding the time limit, with no shorter distance than the initial then the query point is the 1NN to the candidate.

For example, consider the NVD of Figure 2. The network expansion will start at Q “Car1” looking for interest points “Restaurant” and border points. It reaches P1 and stops in that direction. The expansion will also reach b1, b2, b4, b5, b6, b7, and b8. The reached polygons are now checked for cars by utilizing the spatial index; the result will state that P6 is the only polygon that can be considered as a candidate because it does not contain any car. The distance to P6 can be found using NVD border-to-generator distance. Once P6 is reached, a heap list is created with dist(Q,P6) as the initial entry. P6 will then start 1NN search using PINE, and since there are no cars in P6, we use NVD property to find the generator-to-border distance to all neighboring polygons. The reached polygons are checked for cars and the result found is that P5 and P15 contain cars. Now, network expansion starts from b19 and b22 to find the distances to Car3 and Car6. The expansion finds that dist(P6,Car3) and dist(P6,Car6) > dist(Q,P6), thus the search stops and we can conclude that Q is the nearest neighbor for P6. However, since P14 is an empty polygon and can be reached passing through another empty polygon then it is considered as a candidate for RNN, and can be treated the same way as P6 but

this time we need dist(b6,b20), dist(b20,P14) and to start PINE at P14. However, for simplicity we assume that dist(P14,car6) < dist(Q,P14), that makes it not the RNN for Q.

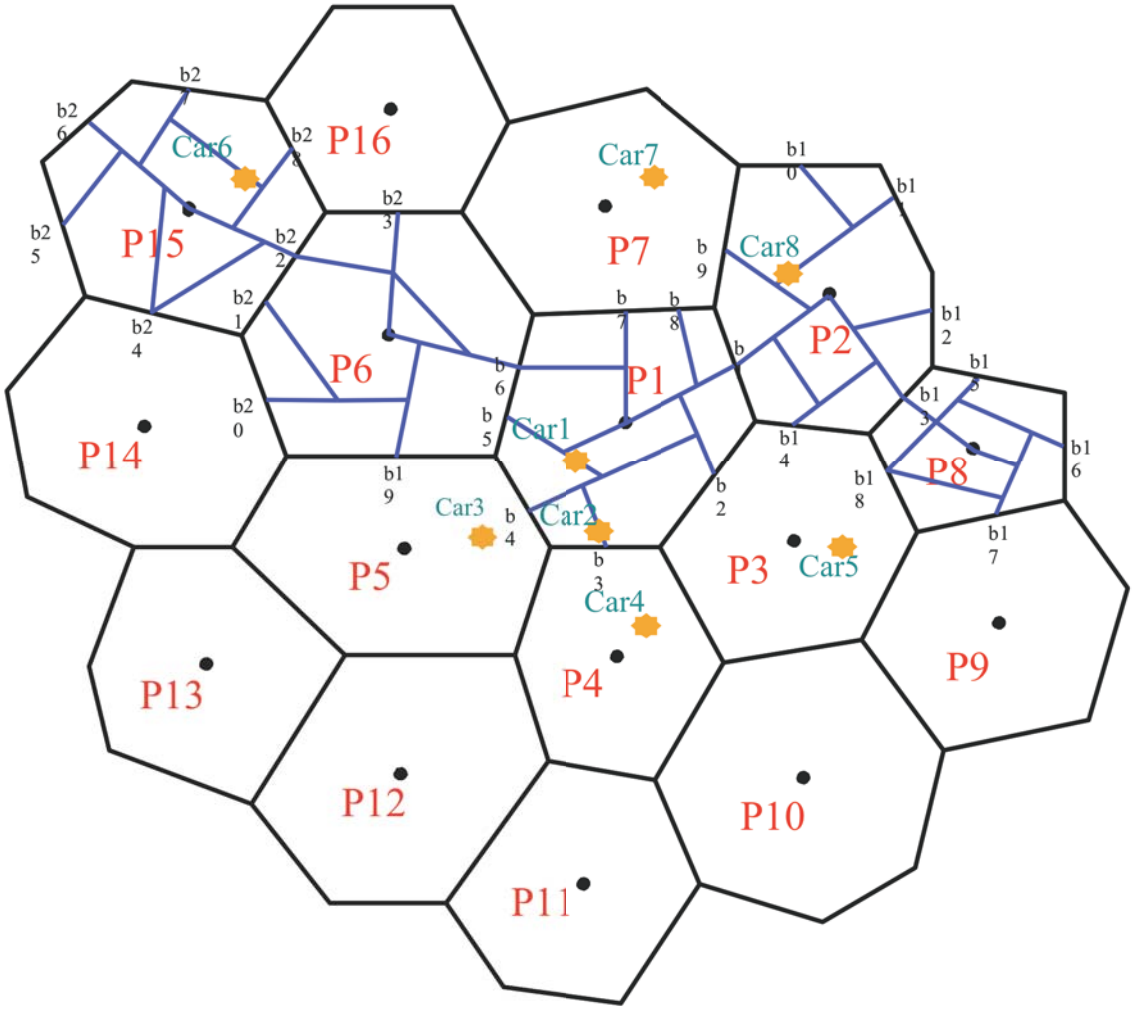

Figure 2. RNN $_{Car}$ (Restaurant)

2.3. RNN $_{Restaurant}$ (Car)

For a given restaurant as a query object, find the cars that consider the restaurant as their nearest neighbor restaurant.

This query type can be solved using only the NVD properties. Since every object in the polygon considers the generator point as the nearest neighbor then the result of the query should be a set of cars that belong to the polygon.

Algorithm steps:

1- If there are cars in the query polygon, then those cars are considered as the answer to the RNN query.

2- If however there are no cars in the query polygon then there is no result for the query (empty set solution.)

2.4. RNN $_{\text{Restaurant}}$ (Restaurant)

For a given restaurant as a query object, find the restaurants that consider the query restaurant as their nearest neighbor.

This query type does not need any inner network distance calculations, since we already have precomputed the NVD for the restaurants. All the required information/data was computed and stored while generating the networkVoronoi diagram. Restaurants are the generator points of the Voronoi diagram and thus all distances from the generator points to borders are known. The candidate restaurants for RNN belong to the set of the query adjacent polygons RNN ∈{Query Adjacent Polygons} (see [Safar (2005), Kolahdouzan (2004)] for details). The query first starts by using NVD to find the distances from the generator of the polygon (in this case it is also "Q") to all border points (i.e. b1, b2, b3 ... etc) and then the distances from those border points to adjacent generators. For example, in Figure 3 we need to find dist(Q,b1) + dist(b1,P2).

Once the neighboring generator points are reached, the algorithm starts a heap list with the dist(Q, Adjacent Restaurant) as the initial distance. The distances between all candidates are first measured (i.e. dist(P2,b14) + dist(b14,P3)) using NVD generator-to-border and vise versa, thus eliminating the repetition of the calculation among them (P2 to P3 = P2 to P3). To cut down the calculations even further, all distances between the polygons are compared to the shortest distance between the Query and its adjacent restaurants 1NN. If a path is found that is shorter than the Q-to-1NN then both restaurants are canceled because they might be considered as the nearest neighbors to each other, or in other words they are closer to each other than the query restaurant.

The new candidate restaurants are then set as query points and they start searching for their 1NN using PINE. Every new found distance from the intersection point, border points and generator points are tested and compared to the first entry in the heap. If the distance is larger, then it is heaped as the second entry. However, if the distance is shorter, then the search stops in that direction and we set the polygon as NOT the RNN to the query point.

Let us take for example Figure 3. The query point is P1 and thus the solution to the query belongs to the set of {P2, P3, P4, P5, P6, P7}. First we execute 1NN query using NVD to find the distance from P1 to b1, b2, b3, b4, b5, b6, b7 and b8 and the distances from those borders to the adjacent generator points. However, if there are two paths to one generator point, for example dist(P1,b7) + dist(b7,P7) and dist(P1,b8) + dist(P8,P7), then the longer distance should be eliminated. At this stage let's assume that P3 was found as the 1NN to Q and dist(Q,P3:10). Next the distances between the candidate neighbors should be calculated for example dist(P2,P7:18), dist(P2,P3:14), dist(P3,P4:8) .. etc. Now, we check the found distances, if two polygons are closer to each other than the query to its 1NN then we say that those two polygons do not belong to the set of candidates RNN. In this example we found that R3 and R4 are closer to each other than to the query point, therefore the RNN candidate set is now {P2, P5, P6, P7}. Each candidate restaurant starts a heap list with the distance from the query point as initial value and executes 1NN query using the NVD generator-to-border distances. If

a distance is found that is shorter than the initial value in its heap then the search stops for that polygon and announces that this candidate generator point is not the RNN to the query restaurant. If however, the search ended with the distance from the query point less than the initial value then the candidate generator point is an RNN to Q.

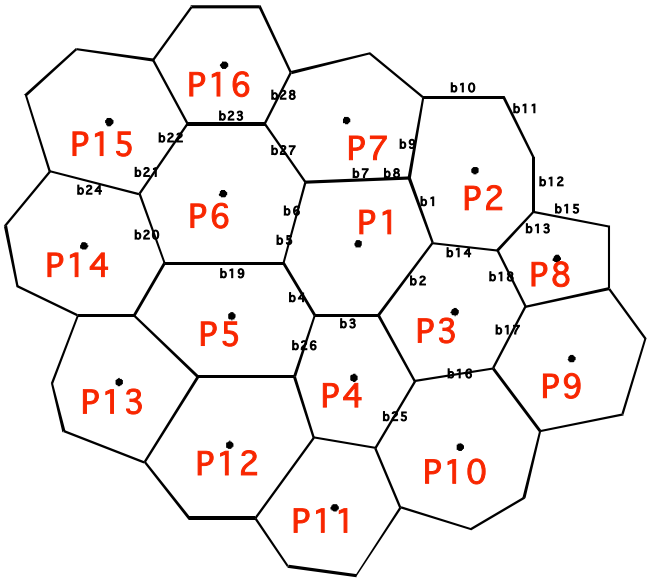

Figure 3. RNN $_{Restaurant}$ (Restaurant)

3. Conclusion and Future Work

This paper presented a comprehensive approach for RNN query in a road network using real distances. The paper presented four different RNN queries and algorithms to solve them. The algorithms are novel and are not based on Euclidean distance. The use of network expansion mechanism of PINE side by side with the NVD has many benefits as the paper explained. It helps in reducing the number of computations required to answer a RNN query. This paper shows the road to several interesting and practical directions for future work on the reverse nearest neighbor query. Many works are redirecting the use of such queries from a scientific method to a real commercial application in several fields like telecommunication and location based services. We also plan to extend the algorithms mentioned in this paper to solve continuous RNN and group RNN queries.

4. Acknowledgments

This research was funded by Research Administration at Kuwait University (Project No EO02/04).

References

Corral, A., Manolopoulos, Y., Theodoridis, Y., and Vassilakopoulos, M. (2000) "Closest pair queries in spatial databases". In Proceedings of ACM SIGMOD International Conference on Management of Data, Dallas, USA.

Kolahdouzan, M., Shahabi, C. (2004) "Voronoi-Based K Nearest Neighbor Search for Spatial Network Databases". In Proceedings of VLDB.

Kollios, G., Gunopulos, D., and Tsotras, V.J. (1999) "Nearest Neighbor Queries in a Mobile Environment". In the Proceedings of the International Workshop on Spatio-Temporal Database Management, pp. 119–134.

Okabe, A., Boots, B., Sugihara, K., and Nok Chiu, S. (2000) "Spatial Tessellations, Concepts and Applications of Voronoi Diagrams, John Wiley and Sons Ltd., 2nd edition, ISBN 0-471-98635-6.

Papadias, D., Zhang, J., Mamoulis, N., Tao, Y. (2003) "Query Processing in Spatial Network Databases". In Proceedings of VLDB: 802-813.

Rigaux, P., Scholl, M., and Voisard, A. (2002) "Spatial Databases with Applications to GIS", Morgan Kaufmann.

Safar, M. (2005) "K Nearest Neighbor Search in Navigation Systems". To appear in the Journal of Mobile Information Systems (MIS), IOS Press.

Song, Z., and Roussopoulos, N. (2001) "K-Nearest Neighbor Search for Moving Query Point". In the Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases.

Tao, Y., Papadias, D., and Shen, Q. (2002) "Continuous Nearest Neighbor Search". In the Proceedings of the Very Large Data Bases Conference (VLDB), Hong Kong, China.

Tao, Y., Papadias, D., Lian, X. (2004) "Reverse kNN Search in Arbitrary Dimensionality". Proceedings of 30th Very Large Data Bases (VLDB), pp. 744-755, Toronto, Canada, August 29-September 3.

Yiu, M. L., Mamoulis, N., Papadias, D., Tao, Y. (2005) " Reverse Nearest Neighbor in Large Graphs". ICDE 2005: 186-187.

Yu, C., Ooi, B.C., Tan, K.L., and Jagadish, H.V. (2001) "Indexing the Distance: An Efficient Method to KNN Processing". In the Proceedings of the Very Large Data Bases Conference (VLDB).