Voronoi Diagrams for Uncertain Objects in Road Networks

Muhammed Mas-ud Hussain Abu Saleh Md. Noman Dr. Muhammad Eunus Ali

Abstract—A wide range of GPS-enabled *Location-Based Services* such as road-side assistance, mobile-based tracking, highway patrol, location-aware advertisement etc. are being frequently enjoyed by most people around the world. These services range from satisfying distance queries, shortest path queries to information queries (e.g. finding the nearest hospital). Exciting developments in mobile computing, *GIS* and *Location-Based Services* have resulted in growing research interests in processing of such queries on spatial networks in recent years. For reasons such as mobility of objects, privacy concern of users, inaccuracy and noise of data capturing machines etc., location of some objects can not be determined exactly on spatial networks. These objects, better known as uncertain objects, provide fascinating challenge to researchers focusing on query processing in road networks. In this paper, our main concentration is on processing *Nearest Neighbor* queries for uncertain objects. In particular, we try to overcome the problem of constructing a probabilistically accurate *Voronoi Diagram* for uncertain objects efficiently. We offer an innovative algorithm which uses a greedy approach and does necessary probabilistic calculations to achieve this goal. Some pruning conditions are imposed and few lemma are invented to make this algorithm work faster and produce desired *Voronoi Diagram* in quick time. Experimental results show that our algorithm is about six to ten times faster than existing approaches and also reduces computational overhead significantly. Our algorithm does the best that can be expected in case of queries related to uncertain objects: always provide a theoretically accurate answer in quick time.

Index Terms—Nearest Neighbor, Uncertain Objects, Road Networks, Spatial Database.

1 Introduction

As GPS-enabled mobile services have become vastly popular in recent years, *Location-Based Services(LBS)* are being enjoyed more often and by most people around the globe. Variety of *LBS*, such as road-side assistance, mobile-based tracking, highway patrol, location-aware advertisement etc. are frequently consulted by users. These services need to provide sufficiently correct answers to different types of queries, ranging from finding shortest path between two points(e.g. source and destination) to searching for nearest data objects around a query point. These developments in mobile computing and GIS have resulted in growing research interests in processing of such queries efficiently in road networks. In this paper, our primary focus is on finding nearest uncertain objects for a given query point. In particular, we try to overcome the problem of constructing a theoretically accurate and optimized *Voronoi Diagram* for uncertain objects. We offer an algorithm *Voronoi Diagram for Uncertain Objects (VDUO)* which uses a greedy approach with probabilistic evaluation to achieve this goal.

Nearest Neighbor(NN) query is one of the important class of queries in spatial database research field. Given a set of objects and distance metric, an *NN* query in road networks can be defined as finding closest object p to a query point q. A road network in general is interpreted as a weighted graph, with nodes and edges corresponding to road intersection points and road segments respectively. Weights, which are assigned to every edge of a road network graph, represents the distance or time required to travel between two nodes. One approach to perform *NN* query processing is to compute the distance between a query point q and objects by incrementally expanding the search area. This can be achieved by implementing traditional algorithms (e.g Dijkstra, Breadth-First-Search) for distance computation. The motivation behind this approach is that only distances of objects closer to q need to be computed. But, poor performance in the road networks where objects are not uniformly or densely distributed is a major disadvantage. Another group of algorithms in solving *NN* queries is based on different kinds of spatial index structures (e.g. R-Tree, M-Tree, VoR-Tree). These algorithms offer a relatively small subset of a large number of objects as the possible nearest objects of q. Although this subset of possible answers can be obtained in quick time, another refinement step including distance compution for these possible nereast objects is needed to determine actual nearest object p. These algorithms also fail in cases where weights of road segments depend mostly on time-based functions instead of Euclidean distances.

Hence, in recent years, Voronoi diagrams have been proved to be extremely efficient in exploring an *NN* search region. The Voronoi diagram is a computational geometry based partitioning method which represents a special kind of decomposition of a given space. It is determined by distances to a specified set of objects or points in the space. As can be seen in Figure 1, it mainly partitions the data space into disjoint Voronoi cells, so that every point

- *M. Shell is with the Department of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, 30332.*
  E-mail: see http://www.michaelshell.org/contact.html
- *J. Doe and J. Doe are with Anonymous University.*

within a Voronoi cell boundary is closer to one particular object. Thus given a Voronoi diagram of a road network with respect to a set of objects, the task of finding nearest object of a query point *q* is as simple as finding the Voronoi cell within which *q* resides. Various techniques exist for constructing Voronoi diagram for objects whose location is known and fixed (such as hospitals, shopping malls etc.). But, due to various issues (e.g. mobility of objects, privacy concern of users, inaccuracy of data capturing machines), an object's location may not be certainly known to the system. This type of objects are called *Uncertain Objects*. Our motivation behind the work in this paper is that very few methods have been invented to form Voronoi diagram for uncertain objects which show desired amount of efficiency and accuracy.

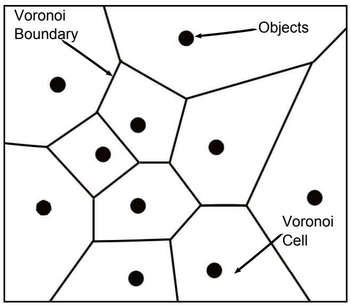


Fig. 1. A Voronoi diagram for a set of objects

Scenario: *First, a person may desire to find the nearest available taxi on the road from his/her location and call for service. Second, a user can issue a query to the central database server to see the location of nearest police cars for emergency assistance. Third, another scenario can be of an application, which uses satellite images to retrieve location of objects*. In the first case, taxi is a moving object with variable speed and thus current exact location of a taxi may not be available to the system. In case of police cars, police force may decide not to provide exact location of their patrolling vehicles for security concerns . Instead, they may provide a region around each patrolling vehicle, within which it will certainly be available. The final scenario too is an example of uncertain database, as due to noisy transmissions and network delay, location data of objects obtained from satellite images may not be accurate. Therefore, uncertain objects are a common feature in wide range of applications relating to *LBS*. The problem with uncertain objects is that representing them as an exact single point in the road network will give inaccurate results. Because, exact point of the road network at which an uncertain object is currently residing, can not be determined. In our proposed algorithm *VDUO*, instead of a point, we treat an uncertain object as a region in the road network within which it will certainly reside. As we are dealing with road networks, an uncertain boundary region for an object will consist of road segments only. We term boundary points of every uncertain region on the road segments as object nodes.

To construct the Voronoi diagram for uncertain objects, we opt for a similar approach as in [5] applied for certain objects. Nearest object for every node is determined at first. If we know nearest object of two end nodes of an edge, we can then find out the exact Voronoi boundary point(if any) on the edge by using splitting point method [5]. As location of an uncertain object is not known exactly, it is impossible to determine the true nearest object for each node. Instead we calculate most probable nearest uncertain object for every node and then by using splitting point method, we can construct probabilistically accurate Voronoi Diagram for uncertain objects. At the begining, *VDUO* treats each uncertain object as an extended object. Extended objects are one kind of certain objects which occupy larger than a point area in the road network. So, at first step, we consider each uncertain object as an extended object having area same as the area of its uncertain region and then find nearest object for all general nodes of the network. A tentative Voronoi diagram can be constructed at this point which will work quite accurately for uncertain objects as well. But *VDUO* provides some probability measures and a refinement procedure to develop a more probabilistically accurate Voronoi diagram. An uncertain object is actually a point in the road network, which can be at any location within its uncertain region. We assume that probability distribution in the uncertain region of an object is uniform. It means that, at a specific time, the object being at any point within the uncertain region is equally possible. For each node in the road network, along with one nearest object found considering all uncertain objects as extended ones, *VDUO* also keeps track of a list of probable nearest objects that has some probability of being nearest neighbor of that node. Based on our assumption of uniform probability, then probability calculations are done on these candidate uncertain objects using basic probability equations to determine the most probable nearest uncertain object for each node.

[5] shows building *Shortest Path Tree(SPT)* with respect to each certain, static object can be used to calculate nearest neighbor for every node. But, for uncertain objects, using *SPT* is a very costly operation and performs horribly in real life data. Because each uncertain object has two or more object nodes associated with it and performing *SPT* for all those object nodes individually is highly inefficient. We offer a novel approach in solving this problem by devising a greedy algorithm. *VDUO* examines one edge at a time and updates nearest neighbor data for concerning nodes accordingly. The trick here is to take the lowest cost unexamined edge available each time. The process continues until theoretically correct nearest neighbor data for all nodes are obtained. *VDUO* uses few invented lemma and pruning conditions to make this process work faster. These modifications along with the naive greedy approach greatly reduces the number of edges processed, thus increases efficiency and decreases processing time.

Our proposed algorithm *VDUO* for constructing Voronoi diagram reduces computational overhead significantly and ensures that probabilistically accurate responses to all kinds of *NN* queries generated by users relating to uncertain objects. *VDUO* can be utilized to construct *Local Network Voronoi Diagram (LNVD)* if proper partitioning method is provided. *LNVD* constructs Voronoi diagram locally, based on a single query point so that pre-computation of whole Voronoi diagram need not be done frequently and thus save processing time. *VDUO* will also produce exciting results where pre-computation of Voronoi diagram is needed, as this algorithm needs much less processing time than other existing ones. We conducted extensive experimental study in a real road network to analyze performance of our proposed algorithm. Experimental data show that our algorithm *VDUO* is about six to ten times faster than both naive and *SPT* approach for moderate number of uncertain objects in the road network. Experiments also illustrate that a significant percentage of edges (around 30-40% of total edges on average) are omitted from computation of Voronoi diagram, which results in computational overhead reduction. It is impossible to always determine an absolutely accurate answer in case queries related to uncertain objects. Our algorithm does the best that can be expected in this circumstances: always provide a theoretically accurate answer in quick time.

In summary, the contributions of this paper are as follows:

- We formulate the problem of constructing Voronoi diagram for uncertain objects in road network.
- We propose a technique to interpret uncertain objects in road network and provide an algorithm *VDUO* to construct Voronoi diagram for them. Thus ensuring efficient *NN* query processing related to uncertain objects. Our algorithm uses a greedy approach and probabilistic calculation to perform this task.
- We impose some pruning conditions and prove few lemma to make *VDUO* more compact and faster.
- We conduct a complete experimental evaluation to figure out effectiveness of our proposed algorithm and how it outperforms all the existing ones.

The rest of the paper is organized as follows. Section 2 examines and describes briefly works related to ours, while Section 3 defines the problem setup and preliminaries: here all the definitions and prerequisites of the areas covered are discussed in brief. Section 4 focuses on our proposed approach by devising pruning conditions and related lemma and also provides an illustrative example of the process. Section 5 describes our algorithm *VDUO* in details. Section **??** discusses experimental setup and analyzes experimental results. Finally, Section 6 demonstrates conclusive remarks and our future work plan.

2 Related Works

[2] stuided $k-NN$ monitoring in road networks, where the distance between a query and a data object is determined by the length of the shortest path connecting them. They proposed two methods that can handle arbitrary object and query moving patterns. The first one maintains the query results by processing only updates that may invalidate the current NN sets. The second method reduces the processing time. It groups together the queries that fall in the path between two consecutive intersections in the network, and produces their results by monitoring the NN sets of these intersections. [3] defined three types of proximity relations to model continuous spatio-temporal queries among sets of moving objects in road networks which evaluated large number of continuous queries (many sets of moving objects) simultaneously.

[4] proposed a new structure called $VoR-Tree$ which merged R-Trees with Voronoi diagram to get benifited. R-Tree was used to find out the NN-search region effciently and Voronoi diagram then covers the search region to find out the result. [5] pointed out that the Network Voronoi Diagram (NVD) requires access to all data objects for preprocessing which might not be suitable for real time large datasets. They proposed a method called the *Local Network Voronoi Diagram (LNVD)* to compute query answers for a small area around the query point which requires no precomputation and as a result reduces data access and computation cost. In fact, this process locally computed a NVD which requires neither offline-construction nor repetitive graph traversal operations.

Several works are performed on Voronoi-based K-NN search for Spatial Network Databases [6]. The main idea behind these approaches is to first partition a large network in to smaller/more manageable regions. This can be achieved by generating a first-order network Voronoi diagram over the points of interest. Each cell of this Voronoi diagram is centered by one object (e.g., a gas-station) and contains the nodes that are closest to that object in network distance (and not the Euclidian distance). Next, pre-compute the intra and inter distances for each cell. That is, for each cell, the distances between all the edges (or border points) of the cell to its center are computed. Distances only across the border points of the adjacent cells are pre-computed. Now, to find the k nearest-neighbors of a query object q, first find the 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. Utilizing the intra-cell pre-computed distances, the distance from q to the borders of the Voronoi cell of each candidate are found, and finally the inter-cell pre-computed distances are used to compute the actual network distance from q to each candidate. [7] proposed two approaches for spatial-network k-NN queries: the *incremental network expansion (INE)* and *incremental Euclidean restriction (IER)*.

In recent years, query processing on an uncertain database has become a buzzword due to its wide range of applications. In case of location based services or biological database, where location of users is not specific, query processing for nearest neighbour becomes challenging. For processing moving nearest neighbour queries on uncertain data, Probabilistic Moving Nearest Neighbor (PMNN) queries, the *Probabilistic Voronoi Diagram (PVD)* was proposed [8]. A *PMNN query* finds the most probable nearest neighbor of a moving query point continuously. To process PMNN queries efficiently, two techniques were provided, a pre-computation approach and an incremental approach. An algorithm was proposed for evaluating *PMNN* queries for pre-computed *PVD*. Since it requires computation every time there is an update in database, it is not suitable for queries which are confined into small space in database. Rather an incremental approach was introduced which retrieves a probabilistic safe region for an uncertain object.

When object locations are uncertain clustering is a little tough because well known clustering algorithms deal with data having known locations. [9] concentrated on the problem of clustering objects with location uncertainty. Rather than a single point in space, an object is represented by a *probability density function (pdf)* over the space. They proposed pruning techniques that are based on Voronoi diagrams to reduce the number of expected distance calculation.

Very few work has been done on uncertain data using Voronoi diagram. In [10], George Beskales proposed to use a Voronoi diagram over 2D points where each query is an uncertain region rather than a query point. Also new general uncertainty model was introduced allowing data and query uncertainty.

[11] introduced uncertainty in Voronoi diagrams and named it *UV-diagram(Uncertain Voronoi diagram)*. The UV-diagram is prepared from UV-cells where each cell is associated with a set of objects. Each UV-partition P is associated with a set S of one or more objects. For any point q located inside P, S is the set of answer objects of q (each object in S has a non-zero probability for being the nearest neighbor of q). But the construction cost of such diagrams is exponential. In another work of uncertain Voronoi diagrams [12], the fuzzy Voronoi diagram as an extension of the Voronoi diagram was introduced. It assumed Voronoi sites to be fuzzy points and then defined the Voronoi diagram for this kind of sites, then provided an algorithm for computing this diagram based on Fortunes algorithm which costs $O(nlogn)$ time. Also the fuzzy Voronoi diagram for a set of fuzzy circles, rather than fuzzy points, of the same radius were introduced.

Where locations of objects are continuously changing, query result can produce errors. In [13] probabilistic query evaluation is studied where each query result is incorporated with probabilistic evaluations including optimizations. Also, [14] worked on existentially uncertain spatial data rather than locationally uncertain data and defined two query types namely, thresholding and ranking queries.

The distance between two uncertain objects can be expressed by probability density function. In [15], the idea to use probabilistic distance functions between positional uncertain objects in order to assign probability values to query results, which reflected the trustability of the result, was adopted.

[16] defined a *Probabilistic Group Nearest Neighbor (PGNN)* query which retrieves a set of uncertain objects such that their probability of being *GNN* is greater than a user-specified probability threshold. They proposed the efficient pruning methods: spatial pruning and probabilistic pruning, to reduce the *PGNN* search space.

In [17], kao et al. studied clustering of uncertain data using Voronoi diagram and R-tree index. They considered each object to be confined into a finite region and proposed a pruning technique based on Voronoi diagrams which reduced the number of calculations. The R-tree indexing reduced the pruning cost largely.

[18] discussed an algorithm for processing *k-Range Nearest Neighbor (kRNN)* query in road networks that finds the k nearest neighbors of every point on the road segments within a given query region. They introduced a shared execution approach for each boundary point of the query and also a parameter to control the storage required for pre-computation.

[19] focussed on representing the uncertain location of the objects moving along road networks as time-dependent probability distribution functions. Based upon an indexing mechanism: *UTH (Uncertain Trajectories Hierarchy)*, an efficient algorithm for processing spatio-temporal range queries was proposed. They discussed a model for uncertain motion along road network. Also an effective indexing structure as well as efficient processing algorithms for both snap-shot and continuous probabilistic range queries were developed.

Intensive research on uncertainty in moving objects has been performed in many papers before. Trajectories of moving point objects were concentrated in [20]. The authors presented new types of spatio-temporal queries. To access the data they introduced two methods: the *Spatio-Temporal R-tree (STR-tree)* and the *Trajectory-Bundle tree (TB-tree)*. In another work [21], the positioning of an moving object was proposed. The paper illustrated how queries regarding uncertainty can be answered. They considered the spatial zone of the objects position during two consecutive sampling is an ellipse.

[22] developed three concepts to process queries regarding uncertain objects (e.g. taxi cabs) and precise objects (e.g. location of gas station) keeping the thing in mind that the location of query issuer may also be imprecise. Incorporating the uncertainty of query issuers location by computational geometry method query expansion was described. Query-Data duality and probability threshold constraint were used to process queries depending on its type.

[23] dealt with NN and CNN(Continuous Nearest Neighbour) queries for moving object database. The authors proposed UNICONS (Unique Continuous Search Algorithm), which incorporates the precomputed NN lists in using Dijkstra's algorithm for NN queries to avoid unnecessary disk I/Os. For CNN queries, the advantage of UNICONS is that only two NN queries are performed between adjacent nodes, independent of the density of objects, which reduces the number of disk I/Os.

[24] presented the interaction between objects in terms of operation and the geometry of the uncertain trajectories of them when their movements are constrained to networks with imprecision. They introduced some data types, algebra and operations to support uncertainty. Besides, an indexing

technique was developed to enhance the speed of query processing.

The answers to continuous NN-queries in spatio-temporal settings are time parameterized in the sense that the objects constituting the answer vary over time. Incorporating uncertainty in the model yields additional attributes that affect the semantics of the answer to this type of queries. Some of the previous work [25] addressed the problem of processing continuous Nearest Neighbor (NN) queries for moving objects trajectories when the exact position of a given object at a particular time instant is not known, but is bounded by an uncertainty region. They identified a simple transformation of a view over the uncertain trajectories and proposed an interval tree called *IPAC-NN tree*.

[26] discussed *Top-k query* processing in uncertain database. Top-k processing in uncertain databases is semantically and computationally different from traditional top-k processing. The interplay between score and uncertainty makes traditional techniques inapplicable. They introduced new probabilistic formulations for top-k queries and constructed a framework integrating space navigation algorithms.

The range nearest neighbor (NN) query is an important query type in location-based services, as it can be applied to the case that a NN query has a spatial region, instead of a location point, as the query location. [27] proposed a new approximate range NN query processing algorithm that enables the user to tune a trade off between query response time and the quality of query answers. Their proposed algorithm allows the user to specify an approximation tolerance level k, where we return an answer set A such that each object in A is one of the k nearest objects of every point in the query region. The larger the value of k, the smaller the answer set returned by a database server. Thus, the approximation tolerance level is a tuning parameter that trades off between query response time and the quality of answers. They designed an incomplete pyramid structure as an access method for efficiently retrieving a set of Voronoi cells that intersects a given query region from a Voronoi diagram for the proposed query processing algorithm.

Suppose, the precise locations of n sites (n points in the plane) are known and yet we would like to determine, for every point in the plane, the closest site to that point. If we know the approximate location of each site, say, that the i^{th} site lies in a subset D_i of the plane, then we might be able to answer this question perhaps not for every point but for many points in the plane. Our goal is to find, for each site i, the set of points that are guaranteed to be closer to that site than to any other. In other words, no matter where each site lies (as long as the j^{th} site is in D_j for every j) the closest site to the point is always site i. For some points, we cannot guarantee a closest site. In [28], first, formally the partition of the plane into cells of guaranteed closest points and the neutral zone and state some properties of this partition are defined. Then they considered the special case when the uncertain regions (i.e. the subsets D_i) are discs and showed that the complexity of the partition in this case is linear in the number, n, of sites, and that it can be calculated in $O(nlogn)$ time.

3 Preliminaries and Problem Setup

For security, privacy issues or random mobility, position of an uncertain object can not be exactly retrieved. In case of extended objects, they can occupy a large area in the road network. So, instead of being a point in the road network, positions of these objects are rather treated as a region. For uncertain objects, this region consists of an area within which the object will most probably remain for a specific time. As we are dealing with road networks, an object's uncertain or extended region will be bounded within roads only. Road networks are considered as a graph where nodes are the intersection of roads and each road is an edge. So, the region related to these objects must be within the edges of a road network graph as in Figure 2. The area of this specific region around each object can vary from applications to applications and also depends on certain properties of the objects.

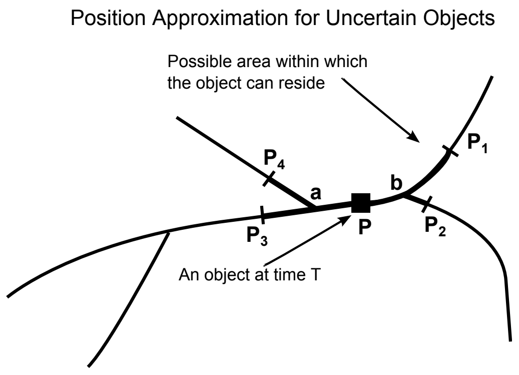

Position Approximation for Uncertain Objects
Possible area within which the object can reside
P_1
P_2
P_3
P_4
a
b
P
An object at time T

Fig. 2. Object position approximation for uncertain objects

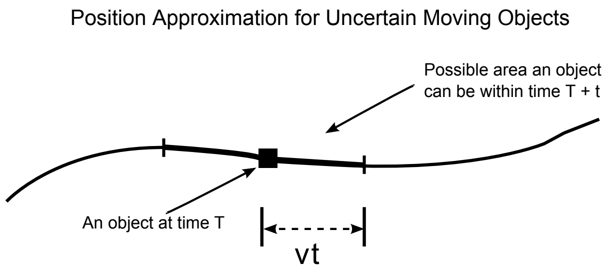
As an example, for moving uncertain objects an average speed from historical data of the objects can be computed. Let, a time interval t, precomputed average speed or average cost per unit time v. At time T, an object's position is retrieved. Approximated position of the moving object within time $T+t$ can be like in Figure 3. Within time $T+t$, the moving object may go to any direction, assuming direction of movement is unknown. So, from the current position of the object P in Figure 3, it can be anywhere within vt cost radius in every direction. v may vary edge-wise.

A road network consists of edges and nodes which represents roads and their intersection points respectively. For understanding of *VDEUO*, clear idea of terms such as intersection nodes, object nodes and object super node is important.

Definition 3.1: (Intersection Nodes). In a road network, intersection nodes are the intersection points of two or more edges that represent roads.

In Figure 4, a, b, g are intersection nodes.

Definition 3.2: (Object Nodes). Given an uncertain object P, the boundary points of its uncertain region on



Position Approximation for Uncertain Moving Objects
Possible area an object can be within time T + t
An object at time T
vt

Fig. 3. Object position approximation for moving objects within time $T+t$

edges of the road network are termed as object nodes. An object node is associated with one particular object. An uncertain object P may have any number of object nodes.

In Figure 4, P_1, P_2, P_3, P_4 are object nodes for object P.

Definition 3.3: (OBJECT SUPER NODES). Given an uncertain object P, the collection of all of it's object nodes are termed as an object super node. It is a collection of object nodes. An uncertain object P can have only one object super node.

In Figure 4, collection of $\{P_1, P_2, P_3, P_4\}$ is termed as an object super node collectively.

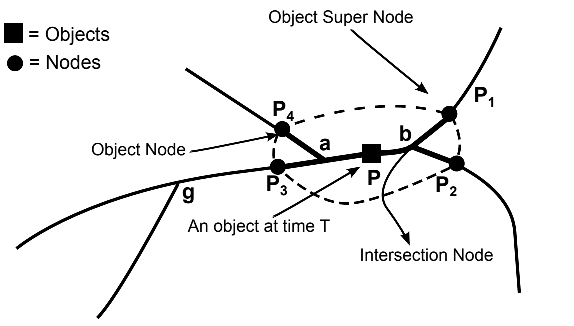


Fig. 4. Object Node, Object Super Node and Intersection Nodes

3.0.0.1 : In our case, we are provided with a road network graph $G(V,E)$ with a set of nodes V(intersection point of roads) and set of edges E where each edge may have a cost associated with it indicating the cost which will be incurred if that edge/path is chosen (e.g., the time required to pass that entire edge). There are some extended/uncertain objects(e.g., ATM booth) residing on the network. Our main goal is to construct a Voronoi diagram using splitting point idea discussed in Section 4. After that, whenever a query is issued we need to determine which voronoi cell the query issuer belong to and respond with corresponding cell object.

3.0.0.2 : In mathematics, a Voronoi diagram is a special kind of decomposition or partition of a given space, e.g. a metric space, determined by distances to a specified family of points or objects (subsets) in the space. To each such an object or point, one can relate a corresponding Voronoi cell, alternatively called Dirichlet cell, representing the set of all points in the given space whose distance to the given object is less than or equals to their distance to the other objects.

In case of a road network, a Network Voronoi Diagram or *NVD* means constructing a Voronoi diagram for a given road network graph with respect to a set of objects in the network. In the simplest case of static or certain objects , a finite set of points or objects $P = p_1, \ldots, p_n$ are given in the road network plane. In this case each site p_i is simply a point, and its corresponding Voronoi cell V_i consisting of all points whose distance to p_i is not greater than their distance to any other site. It is a very useful mechanism to process NN queries. Because, then just finding in which Voronoi cell the query point is in will be enough to determine NN object. Thus real time searching or processing NN query is much faster if *NVD* is used.

However, there are some drawbacks of preprocessing technique such as *NVD*. It needs access to all data objects and network nodes, which means that it is not always suitable for large datasets in many real life cases. Local Network Voronoi Diagram or *LNVD* tries to mitigate some of these drawbacks by constructing Voronoi diagram for a small area of the whole network around the query point. It prevents repetitive distance evaluation over the same or similar sets of nodes. If used with effective and clever partioning, it can be a part of more efficient NN query processings.

4 OUR APPROACH

In this section, we first set up definitions and ideas related to our approach. We first propose a method for extended objects only, without considering uncertainity of objects. This idea is then extended for uncertain objects using probability evaluation. Steps of *VDEUO* are shown with a simple example in this section.

4.1 Constructing Network Voronoi Diagram for Extended Objects

4.1.1 Handling of Extended Objects

A static certain object in a road network is a simple point, whose position is certain and in most cases unchanged. The main challenge of constructing *NVD* for uncertain objects in a road network is that the uncertain objects are not as simple as a point. In fact, they must be considered as a region. At first to simplify our thinking, we consider only extended objects of a road network. Extended objects are those objects whose position is certain, but their size is larger than a point. An extended object can occupy a large region in the road network. Consider the following : " *A person needs to find nearest university from different areas of a city* ". In this case, the portion of the road network where an university resides can be considered as an extended object.

In our proposed algorithm, we assume that an extended object has a large area associated with it and every extended object can be considered as an object super node. Thus there are several object nodes for an extended object which represents boundary points of the object in the road network. So, for every extended object in the road network, at first these object nodes are determined. The whole object super node region is then treated like a simple point in our approach while constructing *NVD*.

An approach [5] to building Voronoi diagram for certain objects is to first determining nearest object of all intersection nodes of the road network and then applying splitting point theory to specify Voronoi cell boundaries. To determine nearest object of all intersection nodes, Shortest Path Tree or *SPT* can be constructed for each certain object, which is a simple point of the road network and then nearest object from each intersection node can be obtained from available data. But when dealing with uncertain or extended objects in this approach of building *SPT*, *SPT* must be constructed for each object nodes. As an object can have any number of object nodes and in general the number is more than three, when there are large number of uncertain objects in the road network, this approach becomes highly inefficient and impractical.

To handle this problem, we propose a new algorithm in our *VDEUO* approach. Suppose, $G(V,E)$ is a road network graph, where V represents the set of vertices or nodes and E represents the set of edges. Initially we start from a null graph G' with a set of nodes V and a null set of edges. Initially, all intersection nodes have null as their nearest object. Then one by one edges from set E is added to G' in increasing order of edge costs. Each time during addition of an edge, node status or attributes of its end nodes are updated accordingly. Adjacency list of all intersection nodes is also built gradually with addition of edges. Certain constraints and pruning conditions are applied in our algorithm, so that less number of edges need processing. Our algorithm keeps size of graph G' as minimal as possible. Processing data available from graph G', we can determine nearest object of all intersection nodes. The full process and algorithm is described in the following sections with a simple example.

4.1.2 An Example of Constructing NVD for Extended Objects

Let, $G(V,E)$ is a road network graph. Let, set of nodes V = $\{a,b,c,d,e,f,g,h\}$ and set of edges E has 16 members initially. The road network G is shown in Figure 5. Figure 5 shows all the members of set E with corresponding edge costs.

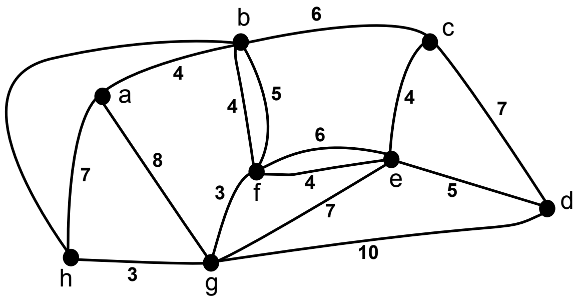
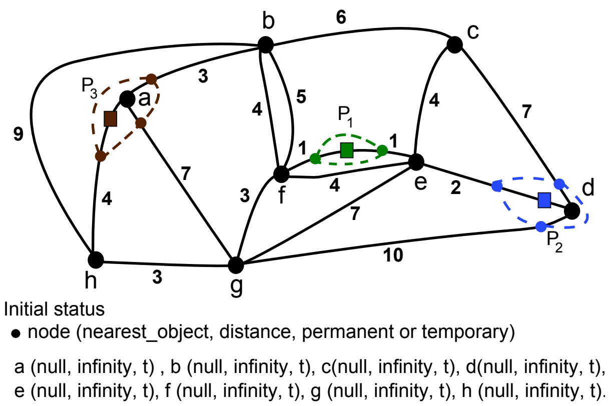


Fig. 5. A road network $G(V,E)$

A set of extended objects $P = \{P_1,P_2,P_3\}$ remain in road network G. Our main objective while constructing the Voronoi diagram is to find ‘ *nearest object* ’ for each intersection node of the road network. At first using our idea of Section 3, object nodes and object super node are created for each object. The graph is shown in Figure 6 showing positions of all objects, object nodes and object supernodes. A node’s status is represented using three attributes: *node name (nearest object, distance to nearest object, temporary or permanent)*. For the third attribute, t is used to represent temporary and p is used to denote permanent. Initially for all intersection nodes, their status is $(null, infinity, t)$.



Initial status
● node (nearest_object, distance, permanent or temporary)
a (null, infinity, t) , b (null, infinity, t), c(null, infinity, t), d(null, infinity, t), e (null, infinity, t), f (null, infinity, t), g (null, infinity, t), h (null, infinity, t).

Fig. 6. Initial condition of the network

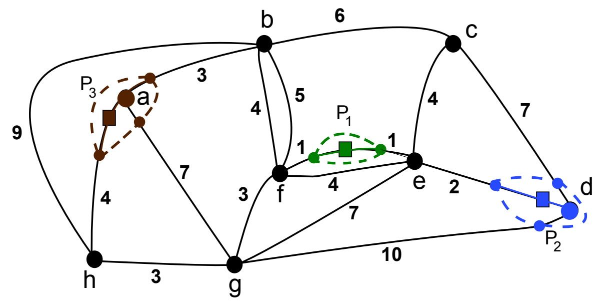
The first step to construct a *NVD* for uncertain and moving extended objects using algorithm provided by *VDEUO* after all object supernodes are defined, is to insert all edges of the road network in a priority queue Q where they are sorted based on edge costs. Edges with lower costs have higher priority and will be popped earlier. Edges between object nodes and internal edges within an object super node are not inserted in Q. Initially adjacency list of all intersection nodes is empty. The object associated with an object node is assigned as its nearest object and distance to nearest object is set to 0. For intersection nodes within an object super node, following Lemma can be derived. Here, nearest object and distance to nearest object of an intersection node within an uncertain object’s region is set to that object and 0 respectively.

Lemma 4.1: For any internal node n, which has edges to more than one object node of a particular object P, then n is within the super node of P. So, $n_{nearestObject} = P$ and $n_{dist} = 0$.

Implying lemma 4.1, we get the status of all the intersection nodes as in Figure 7. We can see, status of node a and d is changed using lemma 4.1. Their status is $(P_3, 0, p)$ and $(P_2, 0, p)$. Except for nodes a and d, all other nodes status remain as $(null, infinity, t)$.

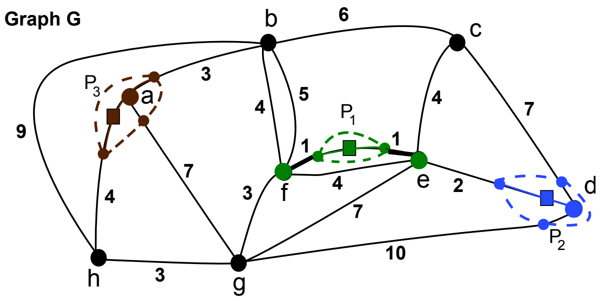
Now we have a null graph G' consisting of object nodes and intersection nodes of the road network G, but we have not considered any edges yet.

Then we have to pop the top element e of priority queue Q. e is the edge with lowest cost amongst unprocessed edges of road network G at any time. e can also be termed as the edge with highest cost of graph G'. After adding e to graph G', we must check if the addition of e



After aplying lemma 4.1 -
● node (nearest_object, distance, permanent or temporary)
a (P_3, 0, **p**) , b (null, infinity, t), c(null, infinity, t), d(P_2, 0, **p**),
e (null, infinity, t), f (null, infinity, t), g (null, infinity, t), h (null, infinity, t).

Fig. 7. Condition after applying Lemma 4.1



After popping edges P_1f(1), P_1e(1) and applying lemma 4.2 - 4.3 :-
● node (nearest_object, distance, permanent or temporary)
a (P_3, 0, **p**) , b (null, infinity, t), c(null, infinity, t), d(P_2, 0, **p**),
e (P_1 , 1, **p**), f (P_1 , 1, **p**), g (null, infinity, t), h (null, infinity, t).

Fig. 8. Status after popping edges $P_1f(1), P_1e(1)$ and applying Lemma 4.2- 4.3

updates the status of e's end nodes which are not marked as permanent. Each of the end nodes is then added to each other's adjacency list if they are not yet in each other's list. In our example, one of the edges P_1f and P_1e is popped at first, as both of them have cost 1. The other one is popped in next iteration. If addition of any edge updates a node n's status, status of nodes in adjacency list of n must also be checked for updates. We should also track the edge with largest cost in grpah G' throughout the algorithm and term its cost as $Cost_{largestEdgePopped}$. On basis of $Cost_{largestEdgePopped}$, two more lemmas can be obtained which are useful for marking intersection nodes status or attributes as permanent.

Lemma 4.2: If for any node n, $n_{dist} < Cost_{largestEdgePopped}$ holds, then the node n's data can be marked as permanent and n can be ommitted from further processing of the graph.

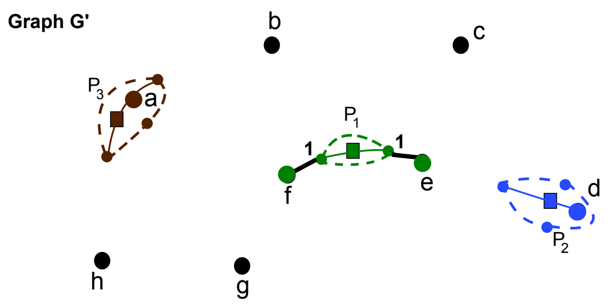
Lemma 4.3: If for any node n, $n_{dist} = Cost_{largestEdgePopped}$ holds and n does not have any edge to an object node having cost larger than or equals to $Cost_{largestEdgePopped}$, then the node n's data can be marked as permanent and n can be ommitted from further processing of the graph.

All nodes satisfying Lemma 4.2 and 4.3, are marked as permanent after addition of any edge e. We have to repeat this process of adding an edge to graph G', until all intersection nodes of graph G are marked as permanent.

In our example, after two iterations and popping both the edges P_1f and P_1e, the status of all the nodes are shown in Figure 8. In Figure 8, the road network graph G is shown and popped edges are highlighted. State of graph of G' is illustrated in Figure 9 where only two edges P_1f and P_1e are added till now. Current $Cost_{largestEdgePopped}$ is 1. Adjacency list of nodes e and f are also updated. Here, node e and f are also marked as permanent using Lemma 4.3.

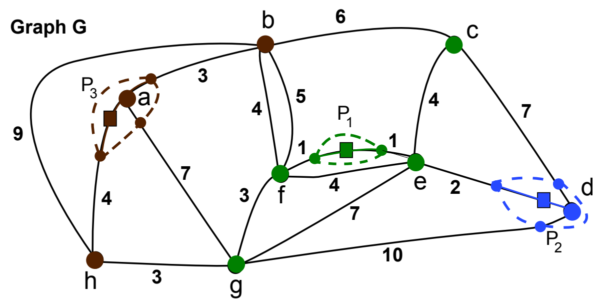
After adding edges $P_1f, P_1e, P_2e, hg, P_3b, gf, ce, bf, hP_3, ef, bf$ (in order) to graph G', all nodes are marked as permanent. We had to add and consider only 11 edges out of total of 17 edges to the graph G'. In our example, the final node status of all the nodes are shown in Figure 10.

After finding nearest object for all intersection nodes,



After popping edges P_1f(1), P_1e(1) and applying Lemma 4.1.2 - 4.1.3 :-
● node (nearest_object, distance, permanent or temporary)
a (P_3, 0, **p**) , b (null, infinity, t), c(null, infinity, t), d(P_2, 0, **p**),
e (P_1 , 1, **p**), f (P_1 , 1, **p**), g (null, infinity, t), h (null, infinity, t).

Fig. 9. Status of Graph G' after popping edges $P_1f(1), P_1e(1)$ and applying Lemma 4.2- 4.3



After 11 iterations of step 4, 5, 6, 7 :-
● node (nearest_object, distance, permanent or temporary)
a (P_3, 0, **p**) , b (P_3, 3, **p**), c(P_1, 5, **p**), d(P_2, 0, **p**),
e (P_1 , 1, **p**), f (P_1 , 1, **p**), g (P_1, 4, **p**), h (P_3, 4, **p**).

Fig. 10. Final node status

Voronoi diagram for the road network of extended objects can be constructed using splitting point idea [5], [3]. If two end nodes of an edge in graph G have same nearest object P then the whole edge lies in the voronoi cell of P. If two end nodes a and b of edge ab have different nearest objects P_1 and P_2 then a splitting point s' on edge ab is found using following equation :

$$d_{cost}(P_1,a)+d_{cost}(a,s') = d_{cost}(s',b)+d_{cost}(P_2,b) \quad (1)$$

An example is shown in Figure 11, where a node a has status$(P_1,5,p)$ and another node b has status $(P_2,7,p)$ cost of edge $d(ab) = 4$. Using eq. 1, we can see $s' = (6/2) = 3$.

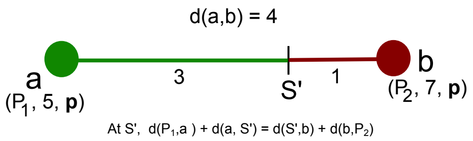


Fig. 11. Spliiting point determination

After performing all of these steps, a Voronoi diagram like Figure 12 will be obtained for extended objects.

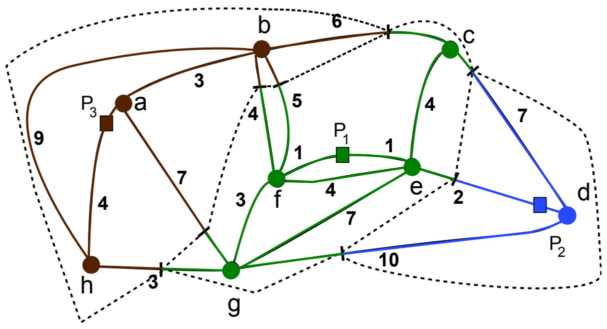


Fig. 12. Voronoi diagram for the example road network

4.1.3 Multiple nearest objects for a node

It is possible that two or more objects have same minimum distance from a particular node. Let, any node a has a set of k nearest objects $P_{nearest_a} = P_1, P_2, \ldots, P_k$. While using splitting point method for an edge $e_i(ab)$, which has node a as one of its end points and any node b as the other, then the type of nearest objects of b has diverse effects on how the splitting point method may be used. This is discussed for all types of nearest objects of b.

1) If b has one nearest object and it is not in $P_{nearest_a}$, then splitting is done as the principle of section 4.1.2. One portion of edge e_i will have multiple nearest objects and work like mutual Voronoi cell in the Voronoi diagram.
2) If b has one nearest object and it is one of P_i from $P_{nearest_a}$, then two possibilities occur. If edge e_i lies on the shortest path from a to P_i, then edge e_i will have no spliting point and the whole edge wil be within Voronoi cell of object P_i. Otherwise, we must perform splitting point operation like situation 1.
3) If b has multiple nearest objects and the set of the nearest objects is not equals to $P_{nearest_a}$, then splitting is done as the principle of section 4.1.2.
4) If b has multiple nearest objects and the set of the nearest objects is equals to $P_{nearest_a}$, edge e_i will have no spliting point and the whole edge wil be within mutual Voronoi cell of $P_{nearest_a}$ in the Voronoi diagram.

Thus a node can have multiple nearest objects and we show that this exceptional case can be handled accurately. Some Voronoi cells may be owned by more than one object in these cases.

4.1.4 Further Pruning Conditions

We can apply two more conditions or checkings to prune more edges before checking for any update of node status:

1) If a node n is marked permanent, then all edges$e(n,P_{i_j})$ where P_{i_j} is an object node of object P_i, can be eliminated without further consideration.
2) If a node n is marked permanent, then all edges $e(n,n')$ where n' is also marked permanent, can be eliminated without further consideration.

These pruning conditions can reduce the processing effort of the proposed alogorithm significantly. As can be seen in our example from previous section, if these two conditions are applied before processing an edge for updating node status, a further 4 edges can be eliminated before processing which will take the number of edges processing for updating to only 7 out of total 17 edges!

4.2 Constructing Network Voronoi Diagram for Uncertain Objects

4.2.1 Handling of Uncertain Objects

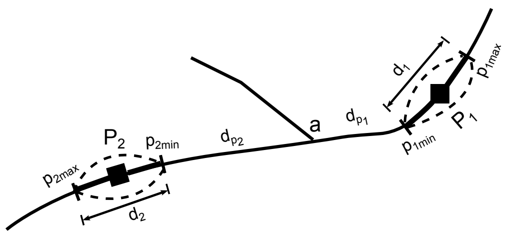


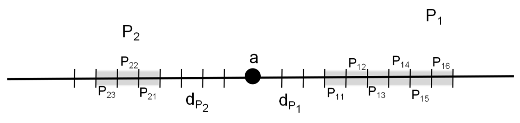
Fig. 13. Reliability issues while tackling uncertain objects

We can relate a region around every uncertain object as its own region. So, like extended objects, uncertain objects also have object nodes and form object super node to specify boundary points. But, an uncertain object is actually a point in the road network, which can be at any point within the specified region. The radius of this uncertain region may vary from object to object. Also, probability of an uncertain object to be at any point within its region may not be uniform.

Our approach proposed for extended objects is used for constructing Voronoi diagram for uncertain objects as well with some modification. So, we must determine nearest object for each node of the raod network like we did in Section 4.1. Results obtained using the previous approach will work mostly fine for uncertain objects as well. But, some of the results will not be accurate as it was for extended object.

As an example, let us look at Figure 13. For simplicity, we consider two objects, P_1 and P_2 with their uncertain region diameter d_1 and d_2 respectively, are shown. d_1 and d_2 may not be equal. If we want to find nearest object for node a, our approach for extended objects will return P_1 as the result as $d_{p_1} < d_{p_2}$. But, for uncertain objects, an object can be at any point within its region. So, it is possible that P_1 is at position P_{1max} and P_2 is at position P_{2min}. Here, $d_{P_2} < (d_{P_1} + d_1)$. In this case, obviously our result from extended approach will fail us miserably as actual nearest object for node a is P_2. So, we can conlude in Lemma 4.4 that nearest object computation for a node is reliable only if maximum distance of the computed nearest object is less than minimum distance of all other objects from that node.

Lemma 4.4: For any node n and its nearest uncertain object P computed from extended approach, the result is reliable if for all other objects P_i, $maxdist(n,P) \leq mindist(n,P_i)$. Here, $maxdist(n,P) = mindist(n,P) + d$, where d is the diameter of the uncertain region of any object P.



here, mindist(a, P_2) = d_{p2} = 4, mindist(a, P_1) = d_{p1} = 3. d_2=3 and d_1=6
So, maxdist(a, $P_{2)}$ = mindist(a, P_2) + d_2 = 4+3 = 7.
maxdist(a, P_1) = mindist(a, P_1) + d_1 = 3+6 = 9.

Fig. 14. Probabilistic calculation

There can be more than one objects P_i, for whom $maxdist(n,P) > mindist(n,P_i)$. To resolve this unreliability to an extent and improve accuracy, probabilistic calculation must be done for all objects involved. A simple example of this calculation can be given from the Figure 14. two objects P_1 and P_2 are shown near a node a. Here, $mindist(a,P_1) = d_{P_1} = 3$ and $mindist(a,P_2) = d_{P_2} = 4$. As $d_{p_1} < d_{p_2}$, P_1 is the nearest object of node a if we were dealing with extended objects. Diameter of uncertain region of P_1 and P_2 is d_1 and d_2 respectively. Here, $d_1 = 6$ and $d_2 = 3$. We can calculate maxdist of any object from a node using eq. 2 :

$$maxdist(a,P_i) = mindist(a,P_i) + d_i \qquad (2)$$

So, $maxdist(a,P_1) = 3+6 = 9$, $mindist(a,P_2) = 4$ and $mindist(a,P_2) < maxdist(a,P_1)$. From Lemma 4.4, we can see that P_1 is not certainly the nearest object of node a. Here, $maxdist(a,P_2) = 4+3 = 7$.

In Figure 15, for simplicity, we divide the uncertain regions of the objects in discrete sub-regions and assume that both objects are on the same long road and they dont enocounter any internal nodes in their region. We also assume that the probability of an object being in one of these sub-regions is uniform for all sub-regions. As an example, P_1 is divided into 6 regions $P_{11}, P_{12}, P_{13}, P_{14}, P_{15}, P_{16}$ respectively and $Probability(P_1 in P_{1i}) = \frac{1}{6}$ for all P_{1i}.

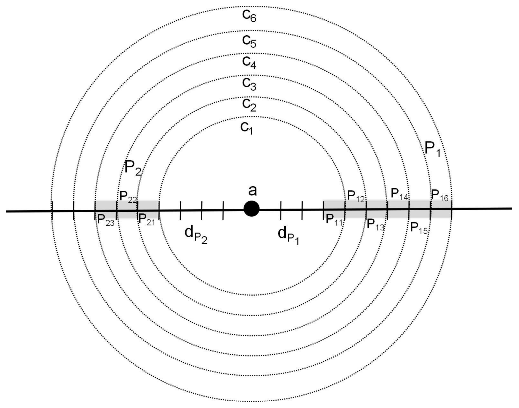


Fig. 15. Probabilistic calculation for P_1

Calculation of probability of P_1 is nearest object of a can be done by calculating the probability for each sub-region. From Figure 15, probaility of P_1 is nearest object of a when P_1 is in sub-region P_{11}, is equals to the probability of P_1 in P_{11} multiplied by the probability of P_2 not within the circle c_1. Similarly, probaility of P_1 is nearest object of a when P_1 is in sub-region P_{12}, is equals to the probability of P_1 in P_{12} multiplied by the probability of P_2 not within the circle c_2. So, we can do such calculation for all sub-region P_{11}, P_{12}, P_{13}, P_{14}, P_{15}, P_{16} using circles $c_1, c_2, c_3, c_4, c_5, c_6$. If we sum all of these proabilities, we will get the actual probability of P_1 being nearest object of node a. So,

$Probability(a.NearestObject = P_1) = \frac{1}{6}.\frac{3}{3} + \frac{1}{6}.\frac{2}{3} + \frac{1}{6}.\frac{1}{3} + \frac{0}{3}.\frac{1}{6} + \frac{1}{6}.\frac{0}{3} + \frac{1}{6}.\frac{0}{3} = \frac{1}{6} + \frac{1}{9} + \frac{1}{18} = \frac{1}{3}$

For the object P_2, it is divided into 3 sub-regions. Here, similar calculation done on object P_1 can be applied which will yield following result:

$Probability(a.NearestObject = P_2) = \frac{1}{3}.\frac{4}{6} + \frac{1}{3}.\frac{3}{6} + \frac{1}{3}.\frac{2}{6} = \frac{2}{9} + \frac{1}{6} + \frac{2}{18} = \frac{1}{2}$

Although $mindist(a,P_2)$ is more than $mindist(a,P_1)$, probability of P_2 being nearest object of node a is more.

In real world applications, dividing regions into discrete unit sub-regions are mostly inappropriate. We must consider the region as a whole or continuously most of the times. Let, a set of k uncertain objects P=$\{P_1, P_2, \ldots, P_k\}$ can be considered as a contender for being the nearest object of a node a. $f(P_1)$, $f(P_2)$, $\ldots$, $f(P_k)$ are probability functions for uncertain objects $P_1, P_2, \ldots, P_k$ respectively. Eq. 3 is used to compute probability for a particular object P_i.

$$Probability(a.NearestObject = P_i) = \int_{u \in P_i} (f(P_i).\Pi^k_{i \neq j}(\int_{v \in P_j} (1 - f(P_j \qquad (3)$$

In our approach *VDEUO*, the object in P with maximum probaility value will be returned as the nearest object of node a. If more than one object evaluate same probability value, then multiple objects will be returned as the nearest object of node a. While constructing Voronoi diagram, we can use principles of Section 4.1.3 in case of multiple nearest objects for a node.

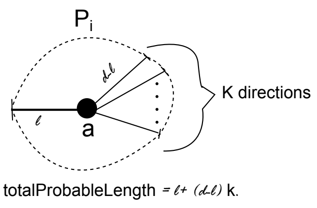


Fig. 16. total probable length computation for P_1

If probability is uniform through the whole uncertain region of any object P_i, then the probability function, $f(P_i) = \frac{1}{totalProbableLength}$. The value of $totalProbableLength$ can be calculated from diameter d. If the uncertain region of P_i does not co-incide with any intersection nodes, then simply $totalProbableLength = d$. Let, uncertain region of P_i intersects with an intersection node a after moving l distance and there are k more edges from a in k directions like in Figure 16. If no other intersection node is encountered in the uncertain region of P_i, then $totalProbableLength = l + (d - l) * k$. So, $totalProbableLength$ can be computed using this equation recursively.

4.2.2 Pruning Conditions for Uniform Probability

If probability of an object to be at any point of its object super node or region is uniform, then further pruning conditions can be applied. These conditions will eliminate the need of probabilistic calculation in certain cases. Let two or more objects are in consideration to be the nearest object of a node. If all of these objects have same $totalProbableLength$ value, then the result obtained from the extended objects algorithm will be sufficient enough for uncertain objects as well. Because, in this case, our approach from Section 4.2.1 will give the same result as the approach from Section 4.1.2 always. Also, if the object that is returned from the extended objects approach as result has lowest value of $totalProbableLength$ among all other candidate objects, then the result is sufficient enough for uncertain objects as well. So, we can conclude to two pruning conditions stated below:

1) For any node *n*, let k objects have possibility of being nearest object of *n*. If probability of an object to be at any point of its object super node or region and radius of all k object super nodes involved is uniform, then no probabilistic calculation is needed and result obtained from the approach for extended objects is sufficient.
2) For any node *n*, let k objects have possibility of being nearest object of *n* and P_i is returned as the nearest object from our approach for extended objects. If P_i has the lowest value of $totalProbableLength$ among these k objects, then no P_i probabilistic calculation is needed and P_i can be returned as the nearest object of node *n*.

These two conditions are useful to speed up the *VDEUO* algorithm significantly.

4.2.3 Modification of Approach

We must make some modifications in *VDEUO* for extended objects to make it work more efficiently for uncertain objects as well. To handle uncertain objects properly, we should keep track of all objects that can possibly be the nearest object for a particular node. Using extended object's concept, for every node n_i, we will get the object P_i with minimum value of $mindist(n_i, P_i)$. To handle uncertain objects, we must also check that if there are any other objects P_j, for which $mindist(n_i, P_j) < maxdist(n_i, P_i)$. So, a node can be made permanent only when $Cost_{largestEdgePopped} \geq maxdist(n_i, P_i)$. We can calculate $maxdist(n_i, P_i)$ using equation 2. For extended objects, condition for making a node parmanent was $Cost_{largestEdgePopped} > mindist(n_i, P_i)$. So, we can modify Lemma 4.2 and construct Lemma 4.5.

Lemma 4.5: For any node n and uncertain object(s) P_i which has minimum $mindist(n, P_i)$, if $Cost_{largestEdgePopped} \geq maxdist(n, P_i)$ holds, node n can be marked as permanent and n can be omitted from further processing of the graph.

We represent a node's status with four attributes as *node name (nearest object, distance to nearest object, probable nearest objects, temporary or permanent)*. Initially *nearest object* attribute will hold the object with minimum *mindist* value. Finally it will hold the value of the most probable nearest object of the node. *probable nearest objects* is a list of a datastructure which contains objects which can be the nearest object of the node except for the one having minimum *mindist* and their corresponding distance form the node. If *probable nearest objects* is null, then *VDEUO*'s approach for extended objects is proved to be sufficient and current value of *nearest object* is the nearest object of that node. No further calculation is needed then. Otherwise probabilistic calculation and pruning conditions checking method must be applied over a node to determine which one of these objects are most probable. Thus a new process for probabilistic evaluation is needed which will determine nearest uncertain object for every node. Then applying splitting point method from Section 4.1.2 will give us a Voronoi diagram of the road network.

While dealing with extended objects, for any intersection node a having P_i as its nearest object, $Dist_{NearestObject}$ is set to $mindist(a, P_i)$ value. This $Dist_{NearestObject}$ value is later used for constructing Voronoi diagram. But, while computing $Dist_{NearestObject}$ considering uncertain objects, distance to the most probable point $Dist_{ProbablePoint}$ in the super node of the uncertain object is added to the $mindist(a, P_i)$ value shown in eq. 4.

$$a.Dist_{NearestObject} \leftarrow mindist(a, P_i) + Dist_{ProbablePoint} \quad (4)$$

In case of uniform probaility function, $Dist_{ProbablePoint}$ is equals to $(NearestObject.d_i)/2$.

If multiple nearest objects are found for a particular node, then $Dist_{NearestObject}$ must be computed individually for each nearest object. Then average single value of these $Dist_{NearestObject}$ values of the nearest objects will used as distance to nearest object while using splitting point method and constructing Voronoi diagram.

If any intersection node a falls in the region of a super node of object P_i, then $mindist(a,P_i)$ is set to zero. Status of node a is marked as *permanent*. Then from eq. 4 $Dist_{NearestObject}$ is equals to $Dist_{ProbablePoint}$ only. If more than one intersection nodes fall in the supernode region of any object P_i, all these nodes' status are marked as *permanent*. If any node a falls in more than one super node regions, then it has multiple nearest objects and is dealt accordingly.

5 Algorithm

We explain the detail of our generic algorithm for constructing a Voronoi diagram in this section. At first, the algorithm is shown for extended objects in a road network. Then this algorithm is enhanced to handle uncertainity of uncertain objects.

5.1 Algorithm for Extended Objects

Algorithm 1 shows steps of Finding-Nearest-Objects-of-Nodes-Extended process for finding nearest extended objects for each intersection node. This is the main algorithm of *VDEUO*. Algorithm 2 shows steps of Position-Approximation-Extended process to create object nodes around each extended object. Algorithm 3 shows the steps of the Update-Check-Extended process which checks whether a node can have a smaller distance to nearest object using an edge given as parameter and updates the intersection node's attributes accordingly.

After successfully implementing Algorithm 1, using splitting point idea from Section 4.1, constructing *NVD* is easily possible. The algorithm takes the following parameters as input : A set of edges E=$\{e_1,e_2,\dots,e_n\}$, a set of internal nodes N=$\{n_1,n_2,\dots,n_m\}$, and a set of extended objects P=$\{P_1,P_2,\dots,P_l\}$ of the road network G. The alogrithm finds nearest object for each internal node n_i in N.

The algorithm outputs a list N', which contains information of nearest objects, distance to nearest objects for each internal node. This algorithm also uses Algorithm 2 and Algorithm 3.

We summarize the notations used in this whole algorithm section as follows:

- G: a road network which as a set of E edges and N nodes.
- P: a set of extended or uncertain objects in the road network G.
- N': a list, which contains for each intersection node: its nearest object, distance to nearest object and whether the node is marked permanent or temporary.
- Q_p: a priority queue, which contains all the edges of E and edges with lower cost will have higher priority.
- $NoOfPermanentNodes$: it holds the value of number of internal nodes marked permanent currently. Initially it is 0. When it is equal to the total number of internal nodes, algorithm 1 terminates.
- $node.adjacencyList$: Initially it is empty for all nodes. Whenever an edge is popped, both end points are added to each other's adjacency list and thus it is build up. A pointer to the edge that connects to the adjacent node is also kept in the list.
- $Enqueue(Q_p, e_i, e_i.cost)$: pushes an edge e_i to a priority queue Q_p, where its position depends on $e_i.cost$.
- $LargestEdgePopped$: it holds the cost of the largest cost edge popped form Q_p currently.
- $Dequeue(Q_p)$: pops the top element of priority queue Q_p.
- $e_i.Endnode$: one of the endnode of the edge e_i.
- $n'_{n_i}.NearestObject$: nearest object(s) for node n_i based on the edges popped till now.
- $n'_{n_i}.Dist_{nearestObj}$: minimum distance to nearest object(s).
- $n'_{n_i}.isPermanent$: it represents a *Yes* or *No* value. If a node is marked permanent, it is omitted from further considertaions.
- $n_i.type$: type of the particular node n_i. A node can be an intersection node or object node.
- $n_i.adjacencyList.\text{ADD}(n_o, e_i)$: adds a node n_o and corresponding edge e_i to the adjacencyList of node n_i.
- $e_i.cost$: cost of the edge e_i.
- $n'_{n_i}.NearestObject.\text{ADD}(P_i)$: adds an object P_i to a node n_i's Nearest Object list in case of more than one nearest objects.

The algorithm first initializes various variables from Lines $1.1-1.7$. In line 1.1, it initialzes N' to be a list of m items. Here, m is the total number of intersection nodes. Then priority Q_p is initilized. Initially no node is marked permanent. So, at stpe 1.3, $NoOfPermanentNodes$ is set to zero. Then for all intersection nodes, respective adjacency list is initialized as an empty list. In lines $1.6-1.7$, each element of N' is set to it's initial values. Initially, every intersection node's nearest object is set to null and distance to nearest object is set to ∞, so that any other valid value is less than the initial value. All intersection nodes are marked temporary at the start. So their *isPermanent* attribute is set to *No* initially.

In line 1.8, Algorithm 2 is used. Algorithm 2 shows steps of Position-Approximation-Extended process to create object nodes around each extended object and update N, E, N' accordingly. This algorithm takes the following parameters as input: a set of extended objects P, a set of intersection nodes N, a set of edges E of road network G and a list N'. Lines $2.2-2.7$ is done for each object P_i iteratively. At first in line 2.2, k object nodes are created representing boundary points of an object. As new object nodes are created, these newly created object nodes must be added to N and E should also be updated in line 2.3. Edges within an object super node region and edges from two object nodes of same object are not inserted to E. In lines $2.4-2.7$, all intersection nodes n_i, which are within the region of the super node of the object P_i, are made permanent using Lemma 4.1. P_i is set as their nearest object and distance to nearest object is set to zero. Then update N, E and N' is returned from Position-Approximation-Extended process.

Thus Algorithm 1 gets update N, E and N' in line 1.8.

Algorithm 1: FINDING-NEAREST-OBJECTS-OF-NODES-EXTENDED(E,N,P)

```
Input : A set of edges $E$={$e_1,e_2,\dots,e_n$}, a set of
        intersection nodes $N$={$n_1,n_2,\dots,n_m$} and a set of
        extended objects $P$={$P_1,P_2,\dots,P_l$}.
Output: A list $N'$={$n'_1,n'_2,\dots,n'_m$}, where $n'_i$ represents
        {$NearestObject, Dist_{nearestObj}, isPermanent$}
        attributes for any internal node $n_i$.
1.1  Initialize $N'$ to a list of $m$ items {$n'_1,n'_2,\dots,n'_m$};
1.2  Initialize a priority queue $Q_p$;
1.3  $NoOfPermanentNodes \leftarrow 0$;
1.4  for each $n_i$ in $N$ do
1.5      Initialize $n_i.adjacencyList$ to an empty list
1.6  for each $n'_i$ in $N'$ do
1.7      $n'_i$.{$NearestObject, Dist_{nearestObj}, isPermanent$} $\leftarrow$
         {$Null, \infty, No$}
1.8  {$N,N',E$} $\leftarrow$
     POSITION-APPROXIMATION-EXTENDED$(P,N,N',E)$;
1.9  for each $e_i$ in $E$ do
1.10     $Enqueue(Q_p, e_i, e_i.cost)$
1.11 $LargestEdgePopped \leftarrow 0$;
1.12 while $NoOfPermanentNodes < m$ do
1.13     $e_i \leftarrow Dequeue(Q_p)$;
1.14     $n_1 \leftarrow e_i.EndNode1$;
1.15     $n_2 \leftarrow e_i.EndNode2$;
1.16     $LargestEdgePopped \leftarrow e_i.cost$;
1.17     if $((n'_{n_1}.isPermanent = Yes$ AND $n'_{n_2}.isPermanent =$
         $Yes)$ OR $(n'_{n_1}.isPermanent = Yes$ AND $n_2.type =$
         $objectNode)$ OR $(n_1.type =$
         $ObjectNode$ AND $n'_{n_2}.isPermanent = Yes))$ then
1.18         $continue$;
1.19     if $((n'_{n_1}.isPermanent = No$ AND $n_1.type =$
         $intersectionNode)$ then
1.20         $N' \leftarrow$ UPDATE-CHECK-EXTENDED$(n_1, e_i, N')$;
1.21     if $(n'_{n_2}.isPermanent = No$ AND $n_2.type =$
         $intersectionNode)$ then
1.22         $N' \leftarrow$ UPDATE-CHECK-EXTENDED$(n_2, e_i, N')$;
1.23     for each $n'_i$ in $N'$ do
1.24         if $(n'_i.isPermanent = No$ AND $n'_i.Dist_{nearestObj} <$
             $LargestEdgePopped)$ then
1.25             $n'_i.isPermanent \leftarrow Yes$;
1.26             $NoOfPermanentNodes$++;
1.27
1.28 return $N'$;
```

Then all edges of E is inserted into the priority queue Q_p. No edge is popped from the priority queue Q_p, so *LargestEdgePopped* is set to zero in line 1.11. Then in lines $1.12-1.27$, which is the main body of this algorithm, edges are popped from Q_p one by one and processed accordingly. This process ends when all intersection nodes are marked as permanent. At line 1.13, the top element of Q_p is popped. Then two end nodes of that edge e_i is saved to two variables n_1, n_2. *LargestEdgePopped*'s values is set to the current popped edge e_i's cost in line 1.16.

Then two pruning conditions from section 4.1.4 are implemented in lines $1.17-1.18$. As stated, all edges satisfying these conditions are not processed further and the loops continues it's next iteration immediately. In lines $1.19-1.22$, it is checked that if update on nearest object and distance to nearest object attributes of both n_1 and n_2 can be made. For this checking, UPDATE-CHECK-EXTENDED process is used. But n_i must be temporary and an intersection node to be checked in UPDATE-CHECK-EXTENDED process. From lines $1.23-1.27$, all intersetion nodes which are not permanent and has distance to nearest object attribute less than that of *LargestEdgePopped* are marked as permanent and *NoOfPermanentNodes* is incremented each time by one using Lemma 4.2. This algorithm returns N', which contains nearest object list and distance to nearest object attributes of all intersection nodes.

Algorithm 2: POSITION-APPROXIMATION-EXTENDED(P,N,N',E)

```
Input : A set of extended objects $P$={$P_1,P_2,\dots,P_l$}, a set
        of intersection nodes $N$={$n_1,n_2,\dots,n_m$}, a list
        $N'$={$n'_1,n'_2,\dots,n'_m$}, where $n'_i$ represents
        {$NearestObject, Dist_{nearestObj}, isPermanent$}
        attributes for node $n_i$ and a set of edges
        $E$={$e_1,e_2,\dots,e_n$} of the road network $G$.
Output: Updated $N$, $N'$, $E$.
2.1  for each $P_i$ in $P$ do
2.2      Create k object nodes {$P_{i_1},P_{i_2},\dots,P_{i_k}$} specifying object
         boundaries
2.3      Update $N$ and $E$ for newly created object nodes;
2.4      for all intersection node $n_i$ within the area of super
         node {$P_{i_1},P_{i_2},\dots,P_{i_k}$} do
2.5          $n'_{n_i}.Dist_{nearestObj} \leftarrow 0$;
2.6          $n'_{n_i}.NearestObject \leftarrow P_i$;
2.7          $n'_{n_i}.isPermanent = Yes$;
2.8  return $N$, $N'$, $E$;
```

Algorithm 3 shows the steps of the UPDATE-CHECK-EXTENDED process which checks whether a node can have a smaller distance to nearest object using an edge given as parameter and updates the intersection node's attributes accordingly if needed. If a node's attributes are updated, it checks for the possible update of adjacent nonpermanent intersection nodes' attributes by recursion. The algorithm takes the following parameters as input : a node n_i for which updating of attributes will be checked, an edge e_i having n_i as one of it's endnodes and a list N' where n'_i represents {$NearestObject, Dist_{nearestObj}, isPermanent$} attributes for node n_i. The algortihm returns updated N' after checking for all possible updates.

In lines $3.1-3.2$, the other endnode of e_i is assigned to n_o variable and n_o is added to n_i's adjacency list along with the edge e_i as the connector. From lines $3.3-3.8$, values of variables *Dist* and *NearestObject* is set, which will be used later in this algorithm. If n_o is an object node, then there is no corresponding values is found in N' for n_o, as it is a list of m items containing attributes of intersection nodes only. So, the variables *Dist* is assigned 0 and *NearestObject* is assigned as the object of n_o. If n_o is an intersection node, then there is a corresponding value of $Dist_{nearestObj}$ and *NearestObject* list in N' for n_o. These values are assigned to *Dist* and *NearestObject* variables in this case.

From lines $3.9-3.21$, it is checked that whether shorter $Dist_{nearestObj}$ for node n_i can be obtained using edge e_i. If sum of *Dist* and edge cost $e_i.cost$ is less than n_i's current distance to nearest object, then n_i's attributes $Dist_{nearestObj}$ and $NearestObject$ is updated in N'. In second case (line 3.16), if sum of $Dist$ and edge cost $e_i.cost$ is equals to n_i's current distance to nearest object, then n_i is equidistant from two objects and value of $NearestObject$ varibale is added to n_i's $NearestObject$ attribute. In both cases, the process UPDATE-CHECK-EXTENDED is recursively called for all non permanent intersection adjacent nodes of n_i (lines $13.12-14$ and $13.18-13.20$). The adjacency list is build within this process gradually. Thus the process updates intersection node n_i's attributes correctly and efficiently.

Algorithm 3: UPDATE-CHECK-EXTENDED(n_i, e_i, N')

```
Input : A node $n_i$, an edge $e_i$ having $n_i$ as one of it's
        endnonodes and a list $N'=\{n'_1, n'_2, \dots, n'_m\}$, where $n'_i$
        represents
        $\{NearestObject, Dist_{nearestObj}, isPermanent\}$
        attributes for an intersetion node $n_i$.
Output: Updated $N'$.
3.1  $n_o \leftarrow e_i.EndNodeOther$;
3.2  $n_i.adjacencyList$.ADD($n_o, e_i$);
3.3  if $n_o.type = ObjectNode$ then
3.4      $Dist \leftarrow 0$;
3.5      $NearestObject \leftarrow n_o.Object$;
3.6  else
3.7      $Dist \leftarrow n'_{n_i}.Dist_{nearestObj}$;
3.8      $NearestObject \leftarrow n'_{n_o}.NearestObject$;
3.9  if $Dist + e_i.cost < n'_{n_i}.Dist_{nearestObj}$ then
3.10     $n'_{n_i}.Dist_{nearestObj} \leftarrow Dist + e_i.cost$;
3.11     $n'_{n_i}.NearestObject \leftarrow NearestObject$;
3.12     for each $n_k$ in $n_i.adjacencyList$ do
3.13         if $n_k.type = intersectionNodes$ AND
             $N'_{n_k}.isPermanent = No$ then
3.14             $N' \leftarrow$ UPDATE-CHECK-EXTENDED($n_k, e_{ik}, N'$);
3.15
3.16 else if $Dist + e_i.cost =$
     $n'_{n_i}.Dist_{nearestObj}$ AND $NearestObject \neq n'_{n_i}.NearestObject$
     then
3.17     $n'_{n_i}.NearestObject$.ADD($NearestObject$);
3.18     for each $n_k$ in $n_i.adjacencyList$ do
3.19         if $n_k.type = intersectionNodes$ AND
             $n'_{n_k}.isPermanent = No$ then
3.20             $N' \leftarrow$ UPDATE-CHECK-EXTENDED($n_k, e_{ik}, N'$);
3.21
3.22 return $N'$;
```

5.2 Algorithm for Uncertain Objects

Algorithm 4 shows steps of FINDING-NEAREST-OBJECTS-OF-NODES-UNCERTAIN process for finding nearest uncertain objects for each intersection node. This is the main algorithm of *VDEUO* for constructing Voronoi diagram for uncertain objects. The algorithm takes following parameters as input: a set of edges $E=\{e_1, e_2, \dots, e_n\}$, a set of intersection nodes $N=\{n_1, n_2, \dots, n_m\}$ and a set of uncertain objects $P=\{P_1, P_2, \dots, P_l\}$. The alogrithm outputs a list N' which contains information on nearest objects, distance to nearest objects for each internal node. This algorithm also uses Algorithm 5, Algorithm 6 and Algorithm 7. Other than Algorithm 7, all other methods are just a modification of their respective version for extended objects. Algorithm 7 is needed only for dealing with uncertain objects and performs probabilistic calculation when needed.

Alomost all notations used in algorithms for extended objects has same meaning in these sections as well. We summarize additional notations and meanings used in this section as follows:

- $n'_{n_i}.NearestObject$: initially holds object(s) with minimum distance for node n_i based on the edges popped till now. After probalistic calculation and all other necessary steps are taken to handle uncertain objects, it holds the value of most probable nearest object(s) for node n_i.
- $n'_{n_i}.Dist_{nearestObj}$: most probable distance to nearest object(s). It is calculated using eq. 4.
- $n'_{n_i}.ProbableNearestObjects$: a list of uncertain objects which have probaility of being nearest object of node n_i other than objects from $n'_{n_i}.NearestObject$. Objects are assigned using Lemma 4.5. It has two fields for each value : a pointer $Object$ to the particular object and corresponding minimum distance of the object from node n_i as $Distance$.
- $P_i.d_i$: diameter or largest distance covered between two points of an object supernode of an uncertain object P_i. $(d_i/2)$ can be termed as radius of an object supernode.
- $maxdist(n_i, P_i)$: maximum distance of an uncertain object P_i from a node n_i. It is calculated using eq. 2.
- $mindist(N_i, P_i)$: minimum distance of an object P_i from a node n_i. It can be calculated as the shortest distance of an extended object from a particular node.
- $n'_i.ProbableNearestObjects$.ADD($P_i, dist$): adds new $ProbableNearestObjects$ value to a node n_i's $ProbableNearestObjects$ list, with P_i is assigned to new value's $Object$ pointer and $dist$ is assigned to $Distance$ field.
- $n'_i.ProbableNearestObjects$.REMOVE($x_k$): removes the value specified by x_k from a node n_i's $ProbableNearestObjects$ list.
- $maxProbability$: holds the value of maximum probability found during probability calculation for a particular node.
- PROBAILITYCALCULATION($x_k, n'_i.ProbableNearestObjects, n_i$): a method thats calculate probability for an object specified by x_k to be the nearest object of the node n_i. This method will vary from application to application.
- DISTANCECALCULATION(P_i, n_i): calculates probable distance from a node n_i to one of its most probable nearest objects. It uses the theory specified in the last paragraph of Section 4.2.3.

The Algorithm 4 is very similar to its version for extended obejects with few modifications. At line 4.7, each element of N' is set to it's initial values. So, each element of N''s $ProbableNearestObjects$ field is also set to null. At lines $4.24-4.25$, nodes are made permanent using Lemma 4.5. Here, $maxdist(n_i, P_i)$ is calculated using $(n'_i.Dist_{nearestObj} + n'_i.NearestObject.d_i)$ equation. If this value is less than or equal to $LargestEdgePopped$, then the corresponding node is marked as permanent. All

Algorithm 4: FINDING-NEAREST-OBJECTS-OF-NODES-UNCERTAIN(E,N,P)

```
Input : A set of edges $E=\{e_1,e_2,\ldots,e_n\}$, a set of
        intersection nodes $N=\{n_1,n_2,\ldots,n_m\}$ and a set of
        uncertain objects $P=\{P_1,P_2,\ldots,P_l\}$.
Output: A list $N'=\{n'_1,n'_2,\ldots,n'_m\}$, where $n'_i$ represents
        $\{NearestObject, Dist_{nearestObj}, ProbableNearestObjects, isPermanent\}$
        attributes for any internal node $n_i$.
4.1  Initialize $N'$ to a list of $m$ items $\{n'_1,n'_2,\ldots,n'_m\}$;
4.2  Initialize a priority queue $Q_p$;
4.3  $NoOfPermanentNodes \leftarrow 0$;
4.4  for each $n_i$ in $N$ do
4.5      Initialize $n_i.adjacencyList$ to an empty list
4.6  for each $n'_i$ in $N'$ do
4.7      $n'_i.\{NearestObject, Dist_{nearestObj}, ProbableNearestObjects, isPermanent\} \leftarrow$
         $\{Null, \infty, Null, No\}$
4.8  $\{N,N',E\} \leftarrow$
     POSITION-APPROXIMATION-UNCERTAIN($P,N,N',E$);
4.9  for each $e_i$ in $E$ do
4.10     $Enqueue(Q_p, e_i, e_i.cost)$
4.11 $LargestEdgePopped \leftarrow 0$;
4.12 while $NoOfPermanentNodes < m$ do
4.13     $e_i \leftarrow Dequeue(Q_p)$;
4.14     $n_1 \leftarrow e_i.EndNode1$;
4.15     $n_2 \leftarrow e_i.EndNode2$;
4.16     $LargestEdgePopped \leftarrow e_i.cost$;
4.17     if $((n'_{n_1}.isPermanent = Yes$ AND $n'_{n_2}.isPermanent =$
         $Yes)$ OR $(n'_{n_1}.isPermanent = Yes$ AND $n_2.type =$
         $objectNode)$ OR $(n_1.type =$
         $ObjectNode$ AND $n'_{n_2}.isPermanent = Yes))$ then
4.18         $continue$;
4.19     if $((n'_{n_1}.isPermanent = No$ AND $n_1.type =$
         $intersectionNode)$ then
4.20         $N' \leftarrow$ UPDATE-CHECK-UNCERTAIN($n_1,e_i,N'$);
4.21     if $(n'_{n_2}.isPermanent = No$ AND $n_2.type =$
         $intersectionNode)$ then
4.22         $N' \leftarrow$ UPDATE-CHECK-UNCERTAIN($n_2,e_i,N'$);
4.23     for each $n'_i$ in $N'$ do
4.24         if $(n'_i.isPermanent = No$ AND $(n'_i.Dist_{nearestObj} +$
             $n'_i.NearestObject.d_i) \leq LargestEdgePopped)$ then
4.25             $n'_i.isPermanent \leftarrow Yes$;
4.26             $NoOfPermanentNodes{+}{+}$;
4.27
4.28 return PROBABILITY-EVALUATION-UNCERTAIN($N'$);
```

lines from $4.1-4.27$ are same as Algorithm 1. At line 4.8, Algorithm 5 is used. This algorithm shows steps of POSITION-APPROXIMATION-UNCETAIN process to create object nodes around each uncertain object and update N, E, N' accordingly.

Algorithm 5 is a slightly modified version of Algorithm 2. From lines $5.1-5.8$, object nodes are created for each uncertain object using the value of its diameter. $totalProbableLength$ is computed here if needed. At line 5.2, k object nodes are created using the diameter value from a particular point. As discussed in Section 4.2.3, for all nodes n_i falling within a supernode of an uncertain object P_i, $mindist(n_i,P_i)$ is equals to 0. Also, P_i is added to n_i's $NearestObject$ attribute. If any n_i falls within the supernode region of multiple objects, then it will have all of these objects as its nearest objects through this algorithm.

Algorithm 5: POSITION-APPROXIMATION-UNCERTAIN(P,N,N',E)

```
Input : A set of uncertain objects $P=\{P_1,P_2,\ldots,P_l\}$, a set
        of intersection nodes $N=\{n_1,n_2,\ldots,n_m\}$, A list
        $N'=\{n'_1,n'_2,\ldots,n'_m\}$, where $n'_i$ represents
        $\{NearestObject, Dist_{nearestObj}, ProbableNearestObjects, isPerma$
        attributes for any internal node $n_i$ and a set of
        edges $E=\{e_1,e_2,\ldots,e_n\}$ of the road network $G$.
Output: Updated $N$, $N'$, $E$.
5.1  for each $P_i$ in $P$ do
5.2      Move any $d_i/2$ distance in all directions from $P_i$;
5.3      Create k object nodes $\{P_{i_1},P_{i_2},\ldots,P_{i_k}\}$ in all possible k
         directions;
5.4      Update $N$ and $E$ for newly created object nodes;
5.5      for all intersection node $n_i$ within the area of super
         node $\{P_{i_1},P_{i_2},\ldots,P_{i_k}\}$ do
5.6          $n'_{n_i}.Dist_{nearestObj} \leftarrow 0$;
5.7          $n'_{n_i}.NearestObject.$ADD$(P_i)$;
5.8          $n'_{n_i}.isPermanent = Yes$;
5.9  return $N$, $N'$, $E$;
```

A node falling within the supernode region of any uncertain object is thus marked permanent immediately. These nodes are not needed for further processing during lines $4.1-4.27$ of algorithm 4. This is handled from lines $5.6-5.8$. The algorithm thus returns updated N,N',E.

Algorithm 4 also uses Algorithm 6 at various steps like algorithm 1. Algorithm 6 shows the steps of the UPDATE-CHECK-UNCERTAIN process which checks whether a node can have a smaller distance to nearest object using an edge given as parameter and updates the intersection node's attributes accordingly if needed. It also updates the list of probable nearest objects for a particular node using the information provided. This process is almost exactly like the process used for extended objects other than the handling of $ProbableNearestObjects$ field update for each node if possible. Lines $6.1-6.8$ are same as the extended version. From lines $6.9-6.21$, it is checked whether shorter $Dist_{nearestObj}$ for node n_i can be obtained using edge e_i. If sum of $Dist$ and edge cost $e_i.cost$ is less than n_i's current distance to nearest object, then n_i's attributes $Dist_{nearestObj}$ and $NearestObject$ is updated in N'. Here, list of probable nearest objects must also be updated for n_i as n_i's current distance to nearest object is changed. From lines $6.12-6.14$, this update of probable nearest objects list is performed. Any member of the current list exceeding the newly computed value of $(n'_i.Dist_{nearestObj} + n'_i.NearestObject.d_i)$ is removed from the list of probable nearest objects.

In second case (line 6.16), if sum of $Dist$ and edge cost $e_i.cost$ is equals to n_i's current distance to nearest object, then n_i is equidistant from two objects and value of $NearestObject$ varibale is appended to n_i's $NearestObject$ attribute. In this case list of probable nearest objects need to be updated as n_i's current distance to nearest object has not changed. At lines $6.26-6.27$, using Lemma 4.4, list of probable nearest objects is updated if possible. If $Dist + e_i.cost < (n'_i.Dist_{nearestObj} + n'_i.NearestObject.d_i)$ and the $NearestObject$ variable is not equals to current

Algorithm 6: UPDATE-CHECK-UNCERTAIN(n_i, e_i, N')

```
Input : A node $n_i$, an edge $e_i$ having $n_i$ as one of it's endnonodes and A list $N'=\{n'_1, n'_2, \dots, n'_m\}$, where $n'_i$ represents $\{NearestObject, Dist_{nearestObj}, ProbableNearestObjects, isPermanent\}$ attributes for any internal node $n_i$.
Output: Updated $N'$.
6.1  $n_o \leftarrow e_i.EndNodeOther$;
6.2  $n_i.adjacencyList$.ADD($n_o, e_i$);
6.3  if $n_o.type = ObjectNode$ then
6.4      $Dist \leftarrow 0$;
6.5      $NearestObject \leftarrow n_o.Object$;
6.6  else
6.7      $Dist \leftarrow n'_{n_i}.Dist_{nearestObj}$;
6.8      $NearestObject \leftarrow n'_{n_o}.NearestObject$;
6.9  if $Dist + e_i.cost < n'_{n_i}.Dist_{nearestObj}$ then
6.10     $n'_{n_i}.Dist_{nearestObj} \leftarrow Dist + e_i.cost$;
6.11     $n'_{n_i}.NearestObject \leftarrow NearestObject$;
6.12     for each $x_k$ in $n'_i.ProbableNearestObjects$ do
6.13         if $x_k.Distance \geq (n'_i.Dist_{nearestObj} + n'_i.NearestObject.d_i)$ then
6.14             $n'_i.ProbableNearestObjects$.REMOVE($x_k$);
6.15
6.16     for each $n_k$ in $n_i.adjacencyList$ do
6.17         if $n_k.type = intersectionNodes$ AND $N'_{n_k}.isPermanent = No$ then
6.18             $N' \leftarrow$ UPDATE-CHECK-UNCERTAIN($n_k, e_{ik}, N'$);
6.19
6.20 else if $Dist + e_i.cost = n'_{n_i}.Dist_{nearestObj}$ AND $NearestObject \neq n'_{n_i}.NearestObject$ then
6.21     $n'_{n_i}.NearestObject$.ADD($NearestObject$);
6.22     for each $n_k$ in $n_i.adjacencyList$ do
6.23         if $n_k.type = intersectionNodes$ AND $n'_{n_k}.isPermanent = No$ then
6.24             $N' \leftarrow$ UPDATE-CHECK-UNCERTAIN($n_k, e_{ik}, N'$);
6.25
6.26 else if $Dist + e_i.cost < (n'_i.Dist_{nearestObj} + n'_i.NearestObject.d_i)$ AND $NearestObject \neq n'_{n_i}.NearestObject$ then
6.27     $n'_{n_i}.ProbableNearestObjects$.ADD($NearestObject, Dist + e_i.cost$);
6.28 return $N'$;
```

Algorithm 7: PROBABILITY-EVALUATION-UNCERTAIN(N')

```
Input : A list $N'=\{n'_1, n'_2, \dots, n'_m\}$, where $n'_i$ represents $\{NearestObject, Dist_{nearestObj}, ProbableNearestObjects, isPerma$ attributes for any internal node $n_i$.
Output: Updated $N'$.
7.1  for each $n'_i$ in $N'$ do
7.2      if ($n'_i.ProbableNearestObjects = Null$) then
7.3          $continue$;
7.4      else if ($n'_i.Dist_{nearestObj} \neq 0$) then
7.5          $n'_i.ProbableNearestObjects$.ADD($n'_i.NearestObject, n'_i.Dist_{nearest$
7.6          $maxProbability \leftarrow 0$;
7.7          for each $x_k$ in $n'_i.ProbableNearestObjects$ do
7.8              $Probability \leftarrow$ PROBAILITYCALCULATION($x_k, n'_i.ProbableNearestObjects,$
7.9              if ($Probaility > maxProbability$) then
7.10                 $maxProbability \leftarrow Probability$;
7.11                 $n'_i.NearestObject \leftarrow x_k.Object$;
7.12             else if ($Probaility = maxProbability$) then
7.13                 $n'_i.NearestObject$.ADD($x_k.Object$);
7.14
7.15     $sumDist \leftarrow 0$;
7.16     for each $p_j$ in $n'_i.NearestObject$ do
7.17         $sumDist \leftarrow sumDist +$ DISTANCECALCULATION($p_j, n_i$);
7.18     $n'_i.Dist_{nearestObj} \leftarrow (sumDist/j)$;
7.19 return $N'$;
```

node's nearest object, then probable nearest object list is updated accordingly.

At line 4.28, N' is returned through method PROBABILITY-EVALUATION-UNCERTAIN. Algorithm 7 shows steps of PROBABILITY-EVALUATION-UNCERTAIN process to perform probability calculations and finalise *NearestObject* attribute of each element of N'. This is a completely new process needed only for uncertain objects. From line $7.1-7.18$, an iteration is done through each element of N'. If *ProbableNearestObjects* value is null for any element n'_i, there is no need for probabilitistic evaluation. At line $7.2-7.3$, this is checked and if this condition is satisfied, the algorithm moves on to next element of N'. Nodes falling within the region of supernode of any object has distance to nearest object value set to 0 in algorithm 5. Their nearest object attribute is handled in that algorithm properly. Their distance to nearest object needs to be calculated though. So, they are escaped in line 7.4 and from lines $7.5-7.14$, most probable nearest object is found using probabilistic calculations.

At line 7.5, current nearest object list is added to the list of probable nearest objects. Then a variable *maxProbability* is set to 0. During lines $7.7-7.13$, probaility value of all objects of the list of probable nearest object is calculated using the idea of Section 4.2.1 and value of maxprobability and nearest object is updated. At line 7.8, probability is calculated using application specific probability function PROBABILITYCALCULATION. If this probability value is greater than the current *maxProbability* value, then nearest objects list of the current node and *maxProbability* is re-assigned to current element of probable nearest objects and computed probability respectively in lines $7.9-7.11$. If the computed probability value is equals to *maxProbability* value, then the current element of probable nearest objects is added to the nearest objects list of the current node. This is done in lines $7.12-13$.

During lines $7.15-7.18$, probable distance to nearest object of the node is computed using the idea of Section 4.2.3. At first, a variable *sumdist* is set to 0. Then, for all objects in the nearest object list, a probable disatnce is calculated using application specific function DISTANCECALCULATION and is added to sumdist. Finally average of all these values are computed by dividing *sumdist* by the total number of objects in nearest object list j and this average value is the probable distance to nearest object. Thus updated N' is returned from Algorithm 7 considering all probabilistic calculations.

Algorithm 4 returns a complete list N', which is then can be used to construct Voronoi diagram using splitting point method and idea of Section 4.1.3. Thus *VDEUO* proposes an efficient algorithm to construct Voronoi Diagram efficiently for uncertain objects to correctly identify most probable nearest object(s) of a query point.

6 Conclusion and Future Directions

This dissertation is a modest approach by us to partition a road network graph based on previously adopted ideas. At first, we started with stating the fact that with the rapid increase of mobile based technology, Location Based Services are getting more and more public interest and in this regard we tried to address our issue. The fact that the uncertainty associated with queries in road networks is quiet obvuious due to security and privacy issues we tried to acclimatise our proposed approach with it which is somewhat different from all the prevailing techniques namely Range NN query, Probabilistic NN query, Linear NN query and so on. We began with stating the challenges we faced and the motivation behind this dissertation. Also we figured out the main contributions made by us. After that we described all the related works done by researchers throughout the world. Also we defined extended and uncertain objects with some new set of definitions e.g., Intersection node, Object node etc. Next using Lemmas and *VDEUO* approach we demonstrated how to construct Voronoi diagrams for both extended and uncertain objects. Instead of building Shortest Path Tree for each object in graph, we built up the graph starting from a null graph and adding edges gradually. This helped us in several ways: kept the graph size as minimum as possible, minimized the overhead associated with each edge cost calculation. Necessary pruning conditions also played a vital role in reducing computational efforts etc.. After successfully retrieving nearest object for all nodes in graph the Voronoi diagram was drawn using previously explained theories. For uncertain objects similar techniques were implied but they are now bounded by some uncertain region and they can be at any point in between it. We have proved that if the probability distribution of the location of uncertain object is uniform, we can eradicate the necessity of probability calculation in certain cases, which speeds up the proposed algorithm significantly. The algorithms proposed afterwards are quiet straightforward and they are classified for both extended and uncertain objects. They aim to find out nearest object for each network node. It is apparent from the experimental evaluations conducted that our proposed approach is much faster and outperforms all other prevailing algorithms in terms of time required to answer the queries.

References

[1] R. Cheng, D. V. Kalashnikov, and S. Prabhakar, "Querying imprecise data in moving object environments," *IEEE Trans. Knowl. Data Eng.*, vol. 16, no. 9, pp. 1112–1127, 2004.

[2] K. Mouratidis, M. L. Yiu, D. Papadias, and N. Mamoulis, "Continuous nearest neighbor monitoring in road networks," in *VLDB*, 2006, pp. 43–54.

[3] Z. Xu and H.-A. Jacobsen, "Processing proximity relations in road networks," in *SIGMOD Conference*, 2010, pp. 243–254.

[4] M. Sharifzadeh and C. Shahabi, "Vor-tree: R-trees with voronoi diagrams for efficient processing of spatial nearest neighbor queries," *PVLDB*, vol. 3, no. 1, pp. 1231–1242, 2010.

[5] S. Nutanong, E. Tanin, M. E. Ali, and L. Kulik, "Local network voronoi diagrams," in *GIS*, 2010, pp. 109–118.

[6] M. R. Kolahdouzan and C. Shahabi, "Voronoi-based k nearest neighbor search for spatial network databases," in *VLDB*, 2004, pp. 840–851.

[7] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao, "Query processing in spatial network databases," in *VLDB*, 2003, pp. 802–813.

[8] M. E. Ali, E. Tanin, R. Zhang, and K. Ramamohanarao, "Probabilistic voronoi diagrams for probabilistic moving nearest neighbor queries," *CoRR*, vol. abs/1106.5979, 2011.

[9] B. Kao, S. D. Lee, D. W. Cheung, W.-S. Ho, and K. F. Chan, "Clustering uncertain data using voronoi diagrams," in *ICDM*, 2008, pp. 333–342.

[10] G. Beskales, M. A. Soliman, and I. F. Ilyas, "Efficient search for the top-k probable nearest neighbors in uncertain databases," *PVLDB*, vol. 1, no. 1, pp. 326–339, 2008.

[11] R. Cheng, X. Xie, M. L. Yiu, J. Chen, and L. Sun, "Uv-diagram: A voronoi diagram for uncertain data," in *ICDE*, 2010, pp. 796–807.

[12] M. Jooyandeh, A. Mohades, and M. Mirzakhah, "Uncertain voronoi diagram," *Inf. Process. Lett.*, vol. 109, no. 13, pp. 709–712, 2009.

[13] R. Cheng, D. V. Kalashnikov, and S. Prabhakar, "Evaluating probabilistic queries over imprecise data," in *SIGMOD Conference*, 2003, pp. 551–562.

[14] X. Dai, M. L. Yiu, N. Mamoulis, Y. Tao, and M. Vaitis, "Probabilistic spatial queries on existentially uncertain data," in *SSTD*, 2005, pp. 400–417.

[15] H.-P. Kriegel, P. Kunath, and M. Renz, "Probabilistic nearest-neighbor query on uncertain objects," in *DASFAA*, 2007, pp. 337–348.

[16] X. Lian and L. Chen, "Probabilistic group nearest neighbor queries in uncertain databases," *IEEE Trans. Knowl. Data Eng.*, vol. 20, no. 6, pp. 809–824, 2008.

[17] B. Kao, S. D. Lee, F. K. F. Lee, D. W.-L. Cheung, and W.-S. Ho, "Clustering uncertain data using voronoi diagrams and r-tree index," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 9, pp. 1219–1233, 2010.

[18] J. Bao, C.-Y. Chow, M. F. Mokbel, and W.-S. Ku, "Efficient evaluation of k-range nearest neighbor queries in road networks," in *Mobile Data Management*, 2010, pp. 115–124.

[19] K. Zheng, G. Trajcevski, X. Zhou, and P. Scheuermann, "Probabilistic range queries for uncertain trajectories on road networks," in *EDBT*, 2011, pp. 283–294.

[20] D. Pfoser, C. S. Jensen, and Y. Theodoridis, "Novel approaches in query processing for moving object trajectories," in *VLDB*, 2000, pp. 395–406.

[21] D. Pfoser and C. S. Jensen, "Capturing the uncertainty of moving-object representations," in *SSD*, 1999, pp. 111–132.

[22] J. Chen and R. Cheng, "Efficient evaluation of imprecise location-dependent queries," in *ICDE*, 2007, pp. 586–595.

[23] H.-J. Cho and C.-W. Chung, "An efficient and scalable approach to cnn queries in a road network," in *VLDB*, 2005, pp. 865–876.

[24] V. T. de Almeida and R. H. Güting, "Supporting uncertainty in moving objects in network databases," in *GIS*, 2005, pp. 31–40.

[25] G. Trajcevski, R. Tamassia, H. Ding, P. Scheuermann, and I. F. Cruz, "Continuous probabilistic nearest-neighbor queries for uncertain trajectories," in *EDBT*, 2009, pp. 874–885.

[26] M. A. Soliman, I. F. Ilyas, and K. C.-C. Chang, "Top-k query processing in uncertain databases," in *ICDE*, 2007, pp. 896–905.

[27] C.-Y. Chow, M. F. Mokbel, J. Naps, and S. Nath, "Approximate evaluation of range nearest neighbor queries with quality guarantee," in *SSTD*, 2009, pp. 283–301.

[28] J. Sember and W. Evans, "Guaranteed voronoi diagrams of uncertain sites," in *CCCG*, 2008.