Continuous Maximal Reverse Nearest Neighbor Query on Spatial Networks

Parisa Ghaemi
Computer Science Department
University of Southern California
ghaemi@usc.edu

Kaveh Shahabi
Computer Science Department
University of Southern California
kshahabi@usc.edu

John P. Wilson
Spatial Sciences Institute
University of Southern California
jpwilson@usc.edu

Farnoush Banaei-Kashani
Computer Science Department
University of Southern California
banaeika@usc.edu

ABSTRACT

Given a set S of sites and a set O of weighted objects located on a spatial network, the optimal network location (ONL) query computes a location on the spatial network where introducing a new site would maximize the total weight of the objects that are closer to the new site than to any other site. The existing solutions for optimal network location query assume that sites and objects rarely change their location over time, whereas there are numerous new applications with which sites and/or objects frequently change location. Unfortunately, the existing solutions for optimal network location query are not applicable to answer these so-called *dynamic* optimal network location queries (DONL), since the result generated by such solutions is most probably invalid by the time computation is complete. In this paper for the first time we formalize the problem of DONL queries as Continuous Maximal Reverse Nearest Neighbor (CMaxRNN) queries on spatial networks, and introduce an approach that allows for efficient and incremental update of MaxRNN query results on spatial networks. With an extensive experimental study we evaluate and demonstrate the efficiency of our proposed approach with both synthetic and real-world datasets.

1. INTRODUCTION

In recent years, optimal location queries (and particularly, optimal *network* location queries) have been widely used in spatial decision support systems and marketing tools. Given a set S of sites and a set O of weighted objects located on a spatial network, the optimal network location query computes a location on the spatial network where introducing a new site would maximize the total weight of the objects that are closer to the new site than to any other site. For instance, a city planner can use optimal network location queries to answer questions such as "where is the optimal location to open a new public library, such that the number of patrons in proximity of nearest to the library is maximized?".

A common and important limitation of the existing solutions for optimal network location query is due to the assumption that sites and objects rarely (if ever) change their location over time. However, there are numerous real-world applications where sites and/or objects are moving entities that frequently change location. Examples of such applications are food truck location planning, diaster-response facility location planning, and mobile police unit assignment, to name a few. For instance, with the food truck location planning application, food trucks which frequently change their locations during the day, can use optimal location queries to determine the best location to stop next (i.e. where they can serve the most customers). With this application not only sites (i.e. food trucks) are moving, but also objects (i.e. customers) change location as they commute throughout the day. Similarly, with the disaster-response planning application aid supply, support units (sites) must be placed (on-the-fly) where they can serve most victims (objects). Likewise, the demand for aid in different areas is likely to frequently change over time as inspections identify new victims and the identified victims receive aid during the disaster response.

The existing solutions for optimal location query (e.g., see [10, 19]) consume hours (or tens of minutes at best) to compute the optimal location; nevertheless they are applicable to classic optimal location applications because the location of the sites and objects (most probably) remain unchanged during the query computation. However, with dynamic applications such as those described above, since sites and/or objects frequently move, the result generated by such solutions is most probably invalid by the time computation is complete; hence, these solutions are inapplicable. For example, according to the data collected from a real food truck application with only 32 trucks, one of the trucks may change location as often as every two minutes, whereas each run of the optimal location query on average takes about 39 minutes to complete (see Section 8.2 for more details).

To be able to support dynamic applications, one should avoid computation of the optimal location query from scratch, and instead, compute the query *incrementally* to leverage computations from past queries. We term such queries *Dynamic Optimal Network Location Queries* (or

DONL queries, for short). DONL queries continuously provide optimal network locations by incrementally updating the result of the query at each time t while as sites and objects change location over time. We should mention that while some similar problems have been studied under the topic of ”dynamic location modeling” by the operations research community, by design their solutions only scale to problems with very small site and object datasets, and can only approximate the exact result by applying heuristics (see Section 2).

In this paper, we formalize DONL queries as Continuous Maximal Reverse Nearest Neighbor (CMaxRNN) queries on spatial networks, and present a scalable and exact solution for CMaxRNN query computation (hereafter, we use the terms DONL and CMaxRNN interchangeable). We argue that answering any *basic* optimal network location query includes two main components. First, one has to compute a spatial neighborhood around each (and every) object o of the given object dataset such that if s is the nearest site to object o, any new site s' introduced within the locality of o will be closer to o as compared to the distance between s and o. The intersection areas where these neighborhoods overlap are the best candidate locations to introduce a new site. Therefore, at the second phase one must compute the overlap among object neighborhoods and identify the optimal network location, which is a network segment (or a set of segments) with maximum total weight. This approach is also applicable to dynamic optimal network location queries. However, repeating the execution of the two aforementioned steps for the entire dataset every time a site point moves or the weight of an object changes, results in a great amount of computational cost and resource consumption.

Instead, in our proposed approach to avoid redundant computation we present a framework for incrementally monitoring the MaxRNN queries in spatial networks. We first precompute a data structure and store the status of the spatial neighborhoods, the overlap among them, and their optimal network location in this data structure. This data structure is constructed in a way that it supports the CMaxRNN queries and is efficiently updateable. At any time instant t, upon receiving an update either in the location of the site or object points, we leverage the precomputed information in the initial phase and identify the part of the network that is impacted by these changes. Then, we update locally the spatial neighborhood of those objects within this locality. Thereafter, the status of the overlap among neighborhoods and the new optimal network location are efficiently updated and stored in the data structure to be used for future dynamic queries.

The remainder of this paper is organized as follows. Section 2 reviews the related work. We define our problem and terminology in Section 3 and Section 4, respectively. In Section 5 and 6, we present our precomputed data structure and introduce our proposed approach, and in Section 7 we present a complexity analysis. Section 8 evaluates our proposed approach via experiments. Finally, Section 9 concludes the paper and discusses directions for future research.

2. RELATED WORK

Since the pioneering work of Ballou [1], OR (operations research) researchers have shown continuing interest in dynamic location modeling. Such models typically result in a schedule or plan for opening and/or closing facilities (sites) at specific times and locations in response to changes in parameters (e.g., demand of the objects, location/relocation cost of facilities) over the time horizon. Common dynamic location/relocation models can deal with single facility [14] and multiple facilities [15, 8], as well as dynamic location/relocation and time-dependent facility location [6, 5], where the demand changes over time. However, given the computational complexity of most dynamic location modeling problems existing solutions mostly comprise of heuristics that can only “guestimate” the optimal location without any guaranteed error bounds. More importantly, due to their computational complexity these solutions/models fail to scale with real datasets that often consist of large numbers of sites and objects.

On the other hand, given similar scalability issues with existing solutions for the generic family of “location problems”, recently the database community has shown interest in developing scalable solutions for these problems. However, to the best of our knowledge, we are the first to introduce and address the CMaxRNN (or DONL) problem, providing a solution which is both exact and scalable. Below, we review the two closest types of related (but orthogonal) location problems that are previously studied by the database community; namely, the problem of maximal reverse nearest neighbor (MaxRNN), which assumes static site and object datasets, and the problem of continuous reverse nearest neighbor monitoring. The CMaxRNN problem can be thought of as a combination of these two problems.

Wong et al. [16] and Du et al. [7] both tackled the problem of MaxRNN queries. While efficient, both of the aforementioned approaches assume p-norm space ([16] assumes L_2 and [7] assumes L_1); hence, their solutions do not apply to spatial networks. The problem of MaxRNN on spatial networks is studied both by Ghaemi et al. [10] and Xiao et al. [19] . Ghaemi et al. [10] introduced two complementary approaches which enable efficient computation of optimal network location queries with datasets of uniform and skewed distributions, respectively. Xiao et al. also proposed a unified framework that addresses three variants of optimal location queries on spatial networks. They divide the edges of the networks into small intervals and find the optimal location on each interval. To avoid the exhaustive search on all edges, they first partition the spatial network graph to sub-graphes and process them in descending order of their likelihood of containing the optimal locations. In both aforementioned approaches, the assumption is that objects and sites are static and they do not change location over time. Therefore, these approaches are not applicable to CMaxRNN queries.

Continuous monitoring of RNN queries has received considerable attention. The first continuous RNN monitoring solution is presented by Benetis et al. [2]. However, they assume that velocities of the objects are known. First work that does not assume any knowledge of objects' motion patterns was presented by Xia et al. [18]. Their proposed solution is based on the six-regions approach. Kang et al. [11] used the concept of half space pruning for addressing continuous RNN queries. Wu et al. [17] proposed a solution for continuous monitoring of RkNN which is similar to the six-regions based RNN monitoring approach in [18]. Cheema et al. [3] focused on continuous bichromatic RkNN queries where only the data objects move. Sun et al. [13] studied the continuous monitoring of RNN queries in spatial networks, but their approach is only applicable to bichromatic RNN queries and also assumes that the query points do not move. Recently, Cheema et al. [4] presented a technique for continuously monitoring RkNN queries (monochromatic and bichromatic) on spatial networks where both the objects and queries continuously change their locations. None of the aforementioned approaches in this group considers continuous maximization of RNNs.

3. PROBLEM DEFINITION

In this section we formally define the problem of CMaxRNN queries. Consider a universal set S of sites (e.g., the set of food trucks, in our food truck location planning application), and a set O of objects with the weight w_o for each object $o \in O$ (e.g., each object can represent the group of people residing in a building, with number of the building occupants as the current weight of the group/object). We assume both sites and objects are located on a spatial network (a road network). At each time t, a site can be either *in-service* or *out-of-service*, where sites can switch between these states throughout the day. Moreover, an in-service site can relocate at any time. On the other hand, to model change of demand, we assume the weight of each object is time-dependent (e.g., in our running example, people can move from one building to another). Note that with this model we can also capture relocation of the objects.

Next, we first define the problem of CMaxRNN queries. Thereafter, we reduce this problem to a series of update operations, which if supported efficiently, one can continuously maintain a precomputation of the MaxRNN (i.e., the optimal location) as site and object datasets change.

3.1 Continuous Maximal Reverse Nearest Neighbor Query (CMaxRNN)

Intuitively, a CMaxRNN query is a continuous query that at time t returns a network segment (or a set of segments) where introducing a new site would maximize the total weight of the objects that are closer to the new site than to any other site. More formally, given a set S' of in-service sites and a set O of objects with weight $w_o(t)$ at time t, *CMaxRNN* returns a subset of the spatial network (i.e. a segment or collection of segments) where introducing a new site s would maximize the total weight of the objects in the bichromatic reverse nearest neighbor (BRNN) set of s. Here, we should remind the reader that the BRNN query on a given site s, returns all the objects $o \in O$ which their nearest neighbor site is s, i.e., there is no other site $s' \in S'$ such that $Dist(o, s') < Dist(o, s)$.

3.2 Update Operations

Our assumed changes in the site and object datasets (i.e., state change of the sites, relocation of the sites, and weight change of the objects) can be captured by one (or a combination) of the following three so-called "update operations":

- *Site Delete Operation* (termed *Delete*, for short), with which a site s is added to the set S' of in-service sites at some time t

- *Site Insert Operation* (termed *Insert*, for short), with which a site s is removed from the set S' of in-service sites at some time t at the optimal location

- *Object Weight Change Operation* (termed *Weight-Update*, for short), when the weight $w_o(t)$ of an object o changes at some time t

For example, site relocation can be implemented as a Delete followed by an Insert. Note that we assume sites are always inserted at the optimal location.

We argue that once MaxRNN is precomputed, for efficient execution of CMaxRNN (i.e., to maintain MaxRNN at every time t), one only needs to support efficient execution of the aforementioned update operations, which capture all changes in the site and operation datasets. With a naïve approach, one can recompute MaxRNN from scratch each time one of the update operations occurs. However, as we mentioned before, this approach fails to scale with large datasets. Accordingly, we propose an *incremental* solution that consists of two components: 1) to identify and precompute a set of state variables that are required for incremental computation of MaxRNN (i.e., the optimal location), and to organize and store the variables in a data structure that allows for efficient update of the variables, and 2) efficiently execute the update operations by updating the precomputed data structure that maintains the state variables. Next, after presenting our terminology in Section **??**, we discuss the two aforementioned components of our solution in Sections 5 and 6, respectively.

4. TERMINOLOGY

In this section we formally define our terminology to be used in the rest of the paper.

DEFINITION 1 (LOCAL NETWORK): *Given an object o, the local network $LN(o)$ of o, is a sub-network expanded at object o that contains all points on the road network with a network distance less than or equal to the network distance between o and its nearest site s; i.e: $LN(o)=\{q|q \in e, dN(o,q) \leq dN(o,s)\}$ where $e \in E$ and $s = argmin_{p \in S} dN(o,p)$.*$\Box$

In Figure 1, site s_1 is the nearest site to the object o_1 where $dN(o_1,s_1)=5$. $LN(o_1)$ is identified by expansion, i.e., starting from o_1 we traverse all possible paths up to the network distance equal to 5, and we delimit $LN(o_1)$ by marking the ending points, namely *markers* (shown as arrows in Figure 1). We term this delimitation process *edge marking*. The expanded network consists of a set of *local edges* connecting the associated object to all marked ending points. It is important to note that local edges can fully or partially cover an actual edge of the road network. For example, the local edges of $LN(o_1)$ are o_1n_2, o_1n_1, o_1n_4 and o_1n (shown as bold lines in Figure 1). Each local edge e is also assigned an influence value, denoted by *I(e)*, which is equal to the weight of the corresponding object. For instance, all local edges in $LN(o_1)$ have an influence value equal of 3 (i.e., the weight of object o_1).

DEFINITION 2 (OVERLAPPING LOCAL NETWORKS): *A local network $LN(o_1)$ overlaps a local network $LN(o_2)$ if $LN(o_1) \cap LN(o_2) \neq \varnothing$. In such cases, there exists at least one local edge e_1 in $LN(o_1)$ which intersects a local edge e_2 in $LN(o_2)$.*$\Box$

For instance, in Figure 1 $LN(o_1)$ overlaps with $LN(o_2)$ since the local edge o_1n_2 in $LN(o_1)$ overlaps with the local edge o_2n_3 in $LN(o_1)$.

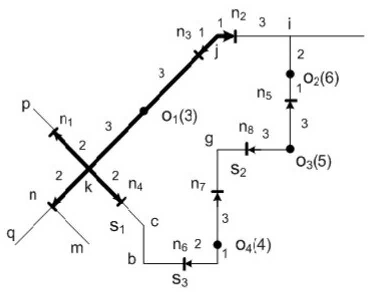

Figure 1: Local Networks

DEFINITION 3 (OVERLAP SEGMENT): *Given two overlapping local networks $LN(o_1)$ and $LN(o_2)$, an overlap segment s is a network segment where two overlapping local edges e_1 and e_2 from the two local networks intersect; i.e.: $s = \{q|q \in e_1, q \in e_2\}$ where $e_1 \in LN(o_1)$ and $e_2 \in LN(o_2)$ and $LN(o_1) \cap LN(o_2) \neq \varnothing$. Accordingly, the influence value of segment s, I_s, is defined as $I_s = I(e_1) + I(e_2)$.*□

For example, in Figure 1 the overlap segment jn_2 has an influence value of 9 and is identified by overlapping the local edges o_1n_2 and o_2n_3.

DEFINITION 4 (SNAPSHOT MAXRNN QUERY): *Given a set O of objects and a set S of sites, the SMaxRNN query returns the optimal network location p, the set of overlap segment(s) with maximum influence value (I_0):*

$p = \{s|s \in OS, s = argmax_{s \in OS} I_s\}$ where OS is the set of overlap segments.□

This is equivalent to the basic optimal network location queries that its aim is to maximize the BRNN set where both object and site points are considered static.

In the road network illustrated in Figure 1 the SMaxRNN query returns the set of overlap segments $\{o_3n_8,\ o_3n_5\}$, where each segment has an optimal influence value I_0 =11.

DEFINITION 5 (IMPACTED OBJECTS): *In "Delete" and "Insert" operations, the objects whose NN site is changed are called Impacted Objects(I-OBJ). When an existing site s is removed from the set of in-service sites, a number of object points get impacted by losing their NN site which are in the RNN set of site s. Also, adding a new site s to the set of in-service sites impacts a number of objects which get attracted to the new NN site s. Besides these two operations, in a "Weight-Update" operation, the objects whose weights change over time are called impacted objects as well.*□

DEFINITION 6 (IMPACTED EDGES): *The edges belonging to the local network of impacted objects are defined as Impacted Edges (I-EDG). Upon receiving a Delete or Insert operation, the impacted edges lose markers or receive new markers. However, in a "Weight-Update" operation, markers remains unchanged but the influence value of the impacted edges changes.*□

5. PRECOMPUTATION

In order to be able to continuously compute the optimal location with CMaxRNN, the main idea behind our proposed solution is to precompute the likelihood of containing the optimal location for each network edge, and maintain a ranking of the edges based on this likelihood from high to low. With this precomputation, one can efficiently identify the optimal location to insert a new site (i.e., when an Insert operation is executed) by starting from the edges with higher likelihood and avoiding the edges with lower likelihood during the search process (rather than exhaustively searching for the optimal location in the entire network). In particular, we compute the optimality likelihood for each edge by computing a "score" that reflects the total weight of the objects whose local networks overlap (at least partly) with the edge. Obviously, the higher the score of an edge, the more is the chance of finding an optimal segment on the edge (where the sum of the weights of the objects whose local networks all overlap on the entire segment is maximal among all network segments).

While precomputation of a ranked edge-optimality-likelihood list allows for efficient computation of the optimal location during Insert, we also need to *maintain* this ranked list as update operations are executed. In particular, with Insert and Delete operations, a new site is respectively added to or removed from the set of in-service sites, which may affect the local networks of some objects and in turn the optimality likelihood of some edges. Similarly, with Weight-Update operation, the weights of a number of objects are changed, and accordingly the score of the impacted edges may change. Accordingly, to enable incremental maintenance of the ranked edge-optimality-likelihood list during execution of the update operations, in addition to the precomputed list itself, we precompute and maintain the local network for each object. With the latter precomputation, we can quickly identify the impacted objects and corresponding edges as update operations are executed, and hence, we can localize execution of the update operations (rather than recomputing across the network). In the rest of this section, we present our precomputation procedure along with the data structure used to store the precomputed measures.

Figure 2 shows the schema of our precomputed data structure. The "Site" table, the "Object" table, and the "Edge" table (also called Marked Edge Table, or MET for short) are implemented as dynamic arrays, and maintain information about sites, objects and spatial network edges, respectively. As depicted in the figure, for each site we maintain the location of the site as well as the network edge it resides on. For each object, in addition to location and edge we maintain the nearest site of the object as well as (the edges that comprise) its local network. Finally, for each edge we main-

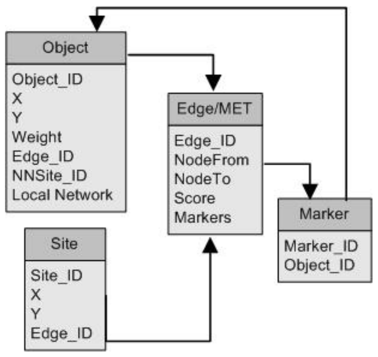

Figure 2: Schema of the precomputed data structure

tain the corresponding network nodes that delimit the edge, the computed score of the edge (as described above), and a list of pointers to the markers residing on the edge. The "Marker" table maintains information about all markers and is stored as a hash table to allow for efficient lookup and update during execution of the update operations.

Figure 3 represents the procedure we follow to populate the precomputed data structure. To emphasize, the precomputation procedure is executed only once and offline (as compared to update operations that are executed online). Below, we explain how we implement this procedure in three steps:

1. *Expanding local networks and marking the edges*: After populating the generic site, object and edge information onto the corresponding tables of our data structure, for each object o we first expand the local network of o, $LN(o)$, using the Dijkstra algorithm, and we stop when we reach the nearest site to o. Then, we mark the ending points/markers of the local networks on the edges and record the markers.
2. *Populating MET*: Once markers are generated, we populate MET entries with corresponding markers. In addition, for each edge entry e, we compute and store the edge score *Sc(e)* (as described above). MET represents our edge-optimality-likelihood list, to be used for computation of the optimal location/segment (described next as presented in Figure 4).
3. *Sorting MET*: Finally, we sort all entries in MET in descending order of *Sc(e)*, to identify the edges with higher likelihood of optimality.

The time complexity of the MET population step is $O(|E|)$ and that of the MET sorting step is $O(|E|log|E|)$. Thus, the overall running time of precomputation procedure is $O(|E|log|E|) + O(|O|(|N|log|N| + |E|))$.

Before we move on to discuss how update operations are implemented based on the precomputed data structure, here we will present a key computation which is frequently invoked during execution of the update operations, i.e., deriving the optimal location/segment based on MET. Figure 4 represents the procedure we follow to perform this computation. Below, we explain how we implement this computation in three steps:

1. *Initializing optimal result set*: Assume the set of optimal location(s)/segment(s) is denoted by S_o, with the optimal influence value I_o. At this step, we initialize S_o to empty set and I_o to zero.
2. *Identifying overlap segments on each edge*: From the set of marked edges in MET, we identify the optimal overlap segments by a process, so-called, edge collapsing. First, we split the edge e into a set of segments, $SG(e)$, where each segment is the part of the edge e which is located between two consecutive markers. Then, for each segment s of $SG(e)$, we identify the local networks overlapping with e which are fully covering s. Accordingly, we compute the influence value of the segment s by summing up the influence values of the corresponding local networks. For each edge entry in MET, the optimal overlap segment s_o, is the segment which has the highest influence value among all segments in $SG(e)$. Note that edge collapsing may produce more than one optimal overlap segment on each edge.
3. *Finding the maximum influence value*: After collapsing each edge, update S_o and I_o, if the influence value I_{s_o} of s_o is larger than the current I_o. Once the computation terminates, S_o includes the set of optimal overlap segment(s) with the optimal influence value I_o.

```
Precomputation Procedure
1: For each $o \in O$
2:     Expand the local network of object $o$
3:     Mark ending points/markers on edges
4: Construct Marked Edge Table (MET)
5: Sort MET table based on $Sc(e)$
```

Figure 3: The precomputation procedure

```
Optimal Location Computation
1: Initialize $S_o$ and $I_o$ to $\varnothing$
2: For each marked edge $e$ of MET table
3:     If $Sc(e) \geq I_o$
4:         Apply edge collapsing to edge $e$
5:         Retrieve $I_s$ and optimal overlap segment(s)
6:         Update $I_o = I_s$
7:         Update $S_o$ to the set of overlap segments
             with maximum influence value $I_s$
8: Return optimal solution set $S_o$ and $I_o$
```

Figure 4: Optimal location computation based on MET

The complexity of the edge collapsing computation is $O(|E||O|^2)$.

6. UPDATE OPERATION EXECUTION

In this section, we present the procedures we have developed to execute the update operations based on the data structure precomputed in Section 5.

6.1 Delete Operation

Once a site s is deleted from the set of *in-service* sites, the status of the corresponding impacted objects and edges must be updated in the data structure as follows:

1. *Retrieving the impacted objects*: All objects belonging to the RNN set of the site s are considered as impacted objects, because these objects are the ones losing their NN site s. With our proposed precomputed data structure, computing the RNN set for s is very efficient, because the NN site for each object is already stored in the Object table and can be used for quick RNN computation.
2. *Removing local networks of the impacted objects*: With our precomputed data structure, this is simply accomplished by removing the corresponding markers from the MET and Marker tables. In this way, we can avoid the costly recomputation of the local networks for all impacted objects.
3. *Expanding the new local networks for the impacted objects and identify their new NN site*: To identify the

new NN site for each impacted object, one needs to expand a new local network for the object. However, we observe that the distance between the new NN site and the impacted object is always larger or equal to that of the s and the object. Therefore, we can derive the new local network simply by extending the previous local network, avoiding reexpansion of the network from scratch.

4. *Populating the tables*: At this step, given new local networks we add new markers of the impacted objects to the MET and Marker tables.
5. *Updating the score of the impacted edges in MET and sorting MET*: Finally, the score of each impacted edge e that loses markers or receives new markers is updated as follows:

$$NewScore(e) = OldScore(e) + [Score(newMarkers) - Score(oldMarkers)]$$

Thereafter, we sort all entries in MET in descending order of their new score.

6.2 Insert Operation

Similar to the Delete operation, once a site s is added to the set of *in-service* sites, the status of the corresponding impacted objects and edges must be updated in the data structure as follows. Note that the computations required at each step is similar to that of the corresponding step in executing the Delete operation as discussed above; here, we avoid from repeating details:

1. *Insert the new site at the optimal location*: At the very first step, the optimal location computation process depicted in Figure 4 is invoked to identify the (segment of) an edge e, which is optimal place for the new insert. Accordingly, the precomputed data structure is updated with the new site information.
2. *Retrieving the impacted objects*: All objects whose local networks include the edge e are considered the impacted objects. These objects can be quickly identified from the Object table given the precomputed information about local networks of the objects.
3. *Removing local networks of the impacted objects*: See Step 2 of the Delete operation execution for details.
4. *Expanding the new local networks for the impacted objects and identify their new NN site*: Similar to Step 3 of the Delete operation execution, we can compute the new local networks based on the previous local network expansions, but here by contracting the expansion instead.
5. *Populating the tables*: See Step 4 of the Delete operation execution for details.
6. *Updating the score of the impacted edges in MET and sorting MET*: See Step 5 of the Delete operation execution for details.

6.3 Weight-Update Operation

Execution procedure for the Weight-Update operation can be summarized as follows (the steps are self-explanatory by now):

1. *Retrieving the impacted objects whose weight changes*
2. *Updating the score of the impacted edges belonging to the local networks of the impacted objects in MET and sorting MET*

7. COMPLEXITY ANALYSIS

In this section, we analyze the computational complexity of our three aforementioned update operations.

Delete Operation: Below, we discuss the computational complexity of various tasks with the Delete operation.

Retrieving the impacted objects: As mentioned earlier, the RNN set of site points are computed in precomputation phase. Accordingly, given the $EdgeID$ of edge e all impacted objects can simply be retrieved by accessing the Site, Edge and Marker tables in a sequence (see Figure 2). Therefore, this step takes about $O(|O|)$. However, this cost is very low compared to the approach of computing the RNN set by constructing the Voronoi cell of each in-service site ([12]). The NVD can be constructed using the parallel Dijkstra algorithm [9] with Voronoi generators as multiple sources (Recall the cost of running parallel Dijkstra is $O(|O|(|N|log|N|) + |E|))$.

Removing local networks of the impacted objects: The cost of removing the markers of corresponding impacted objects is $O(|E||O|)$ since in the worst case all $O(|O|)$ objects might get impacted and also all local networks might overlap each individual edge of the graph.

Expanding the new local networks for the impacted objects: Since the maximum number of overlapping local networks is theoretically equal $|O|$, the running time for expanding the local networks takes $O(|O|(|N|log|N|) + |E|))$.

Populating the tables: Marking all ending points on edges requires $O(|O||E|))$ time.

Updating the score of the impacted edges in MET and sorting MET: This step takes $O(|E|log|E|)$ time.

The overall running time of CMaxRNN query in response to a Delete operation is $O(|E|log|E|) + O(|O|(|N|log|N| + |E|))$.

Insert Operation: In response to an Insert operation, the first step requires $O(|E||O|^2)$ time which is the cost of edge collapsing. For the second step (i.e. retrieving the impacted objects), objects whose local network includes a specific edge, can be identified by accessing the Object, Edge and Marker tables in a sequence (see Figure 2). Therefore, this step takes about $O(|O|)$. The remainder of the steps have the same complexity as of those related to a Delete operation. Therefore, the overall running time of the CMaxRNN query in response to an Insert operation is $O(|E|log|E|) + O(|O|(|N|log|N| + |E|)) + O(|E||O|^2)$.

Weight-Update Operation: The overall running time of the CMaxRNN query in response to a Weight-Update operation is $O(|E|log|E|)$ since there is no cost of expansion involved.

8. EXPERIMENTAL EVALUATION

We next describe the setup we used for the experiments and then present and discuss the results.

8.1 Experimental Setup

All experiments are performed on an Intel Core 2.2GHz, 4 GB of RAM, running Windows 7 and the .NET platform 3.5. The algorithms are implemented in Microsoft $C\sharp$. We use a spatial network of $|N| = 375691$ nodes and $|E| = 871715$ bidirectional edges, representing the LA County road network. The spatial network covers 130 km * 130 km and is cleaned to form a connected graph. We use both real-

Table 1: Four synthetic datasets for objects and sites

Set	Object Size	Site Size	Spatial Distribution	Weight Distribution
S_1	2000, 5000, 10000, 20000	500	Uniform	Uniform
S_2	20000	500, 1000, 2000, 5000	Uniform	Uniform
S_3	20000	500, 1000, 2000, 5000	Uniform	Normal
S_4	20000	500, 1000, 2000, 5000	Normal	Uniform

world and synthetic datasets for objects and sites. All sites, objects, nodes and edges are stored in memory-resident data structures.

Real dataset: In our real-world dataset, objects are population data derived from the LANDSCAN population database compiled on a 30" x 30" latitude/longitude grid. The centroid of each grid cell is treated as the location of each object and the population within each grid cell as the weight of object. For the objects which are not located on road network edges, we snapped them to the closest edge of the road network. In total we have $|O|= 9662$ objects. The weights of objects are distributed nearly uniformly with an average of 1100. Sites are food trucks locations derived from a web mobile application operated by TruxMap (http://www.foodtrucksmap.com/la/). TruxMap is a live map of all the roaming food trucks in Los Angeles and elsewhere. Once a food truck has scheduled a start or stop through its Twitter account or an iPhone application using GPS, a marker is automatically generated on TruxMap food truck tracker.

Synthetic dataset: We synthesized four datasets (S_1, S_2, S_3, and S_4) with different size and spatial distributions: uniform and normal distribution (mean $\mu = 1$, standard deviation $\sigma = 3.2$). To select each object/site point, we randomly picked both X and Y dimensions of the point using the uniform or normal distribution. The cardinality and distribution of each dataset is shown in Table 1. For each dataset, both objects and sites are either uniformly distributed or skewed. With S_1, we considered a fixed site-dataset and various object-datasets whereas with S_2, S_3, and S_4 we used a fixed object-dataset and various site-datasets. Also, we considered datasets with two different weight object distributions. For instance, in S_1, S_2, and S_4 the weight of object datasets are all uniformly distributed with a weight equal to 1. However, in S_3 the weight of objects are normally distributed (mean $\mu = 1$, standard deviation $\sigma = 10.2$).

8.2 Experimental Results

Below we present the results of the three series of experiments that we ran on the aforementioned datasets.

8.2.1 *Feasibility Study*

We first verified that the SMaxRNN query is not applicable to CMaxRNN Queries. For this test, we selected our real-world dataset with 9662 object points and for site points, we retrieved the location of 32 food trucks from TruxMap tracked on a given day. We observed that for that given day, the location of sites points might change as frequently as every two minutes. We first applied the SMaxRNN query on the dataset and computed the execution time. Then, we performed the CMaxRNN algorithm and retrieved its corresponding execution time in response to the three Delete, Insert and Weight-Update operations. The execution time of each operation is retrieved by averaging the execution time of a hundred runs. We observed that the SMaxRNN query takes about 39 minutes to identify the optimal location (Table 2). However, the CMaxRNN query takes about 68, 37, and 19 seconds for Delete, Insert, Weight-Update, respectively. This experiment verifies the fact that using the SMaxRNN approach for continuously computing the MaxRNN set on spatial network databases is not feasible.

Table 2: Comparing the execution time of SMaxRNN and CMaxRNN queries

SMaxRNN	CMaxRNN		
	Delete	Insert	Weight-Update
39 minutes	68 seconds	37 seconds	19 seconds

8.2.2 *Empirical Analysis*

In order to evaluate the execution times of our proposed approach, we implemented a set of experiments with synthetic datasets. Below, we describe each experiment in more detail.

Effect of site and object cardinality on CMaxRNN and SMaxRNN: For this experiment, we selected both the S_1 and S_2 datasets and applied the SMaxRNN and CMaxRNN approaches to them and computed their execution times. In order to compute the execution time of each operation (Delete/Insert/Weight-Update), we sampled 100 iterations and picked the average of their execution time as the result. Figures 5 and 6 depict the results of our experiment. We observe that in both datasets the CMaxRNN is about three to ten times faster than the SMaxRNN depending on the number of objects. This is because all objects in the dataset and their corresponding local edges are engaged in the SMaxRNN computation. However, in CMaxRNN the computation is only limited to the impacted objects and impacted edges.

Effect of site and object cardinality on the three Delete, Insert and Weight-Update operations: For this experiment, we selected both the S_1 and S_2 datasets and applied the CMaxRNN approach to the three operations and computed their execution times. In order to compute the execution time of each operation (Delete/Insert/Weight-Update), we sampled 100 iterations and picked the average of their execution time as the result. Figures 7 and 8 depict the results of our experiment. We observe in Figure 7 the execution time of all operations increases when the size of the objects increases in the dataset S_1. Considering fixed site points in S_1, with an increase in the size of objects the cost of local network expansion becomes higher. However, in S_2 (Figure 8) with an increase in the size of site points, the cost of expansion (i.e. the cost of CMaxRNN in response to the three operations) decreases since the object points reach their NN site faster.

Effect of the spatial distribution of site and object datasets on the three Delete, Insert and Weight-Update operations: First, we studied the effect of datasets with different spatial distributions on the Insert operation. The results for the Delete operation is qualitatively similar. For this experiment, we selected the S_2 (uni-

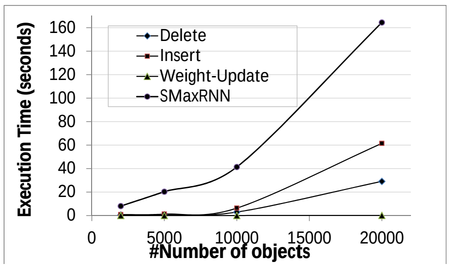

Figure 5: Execution times of CMaxRNN and SMaxRNN on S_1

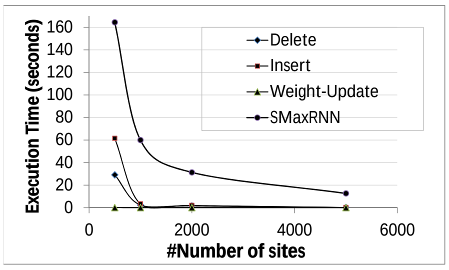

Figure 6: Execution times of CMaxRNN and SMaxRNN on S_2

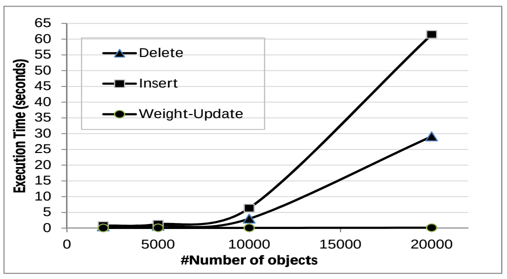

Figure 7: Execution times of the Delete/Insert/Weight-Update operations on S_1

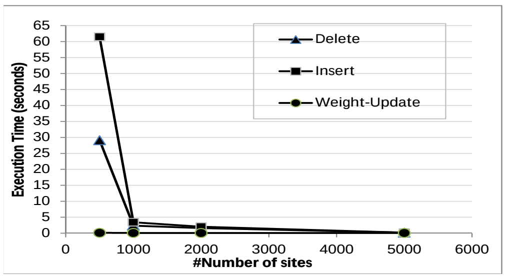

Figure 8: Execution times of the Delete/Insert/Weight-Update operations on S_2

form) and S_4 (normal) datasets. Thereafter, we applied the CMaxRNN approach to the aforementioned datasets and computed both its execution time and the number of impacted objects ($I - OBJ$) in response to Insert operations. In order to compute the execution time, we ran 100 Insert iterations and picked the average of their execution time as the result. It is important to note in each run the new site is placed on a location that is returned as the optimal location in the previous CMaxRNN query run. Figure 9 presents the result of our experiment. The left Y-axis represents the average of the execution time and the right Y-axis shows the number of $I - OBJ$. We observe that the number of $I - OBJ$ is higher with the normal distribution compared to the uniform distribution. This is because the distribution of object points is more skewed on a road network for object datasets with normal distribution. Therefore, the RNN query returns a higher number of $I - OBJ$ in more skewed areas. Accordingly, we observe that the average of execution time with a normal distribution is higher than the one with a uniform distribution which is proportional to the number of $I - OBJ$ and their network expansion, respectively. However, in Figure 9 the average of execution time for both distributions with site points in the range of 1000 to 5000 look similar since the values are low compared to the Y-axis scale and are not distinguishable.

Second, we focused on the effect of datasets with different spatial distributions on the Weight-Update operation. We applied the CMaxRNN approach to the S_2 and S_4 datasets and computed both the execution times and the number of the impacted edges ($I-EDJ$) in response to Weight-Update operations. In order to compute both aforementioned factors, we ran 100 Weight-Update iterations and picked their average as the result. For each Weight-Update operation run, we randomly selected two objects and moved the population data of the first one to the second one. Figure 10 depicts the results of our experiment. The left Y-axis represents the average of the execution time and the right Y-axis shows the number of $I - EDG$. We observe that in both datasets (uniform and normal) with an increase in the size of site points, the average of execution time improves. This is because considering fixed object points with larger site points the cost of network expansion decreases. We also observe that both the average of execution time and $I-EDG$ of datasets with skewed distribution are higher than those with uniform distribution. This effect is because of the fact that with skewed distribution it takes longer for object points to reach their NN site; hence, it takes longer to expand their local networks and compute the optimal location. The size of their local networks also become larger which causes higher number of $I - EDG$ while dealing with *Weight-Update* operations.

Effect of the weight distribution of the object dataset on the three Delete, Insert and Weight-

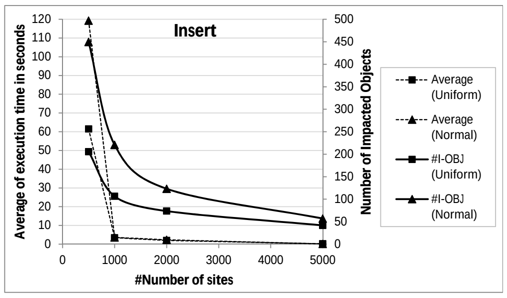

Figure 9: Comparing the execution times of the Insert operation on S_2(uniform) and S_4(normal)

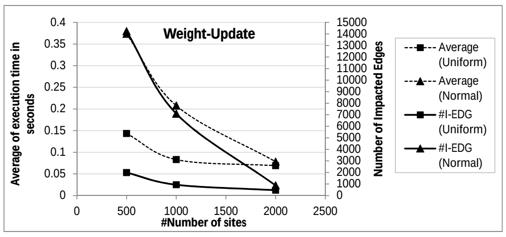

Figure 10: Comparing the execution times of the Weight-Update operation on S_2(uniform) and S_4(normal)

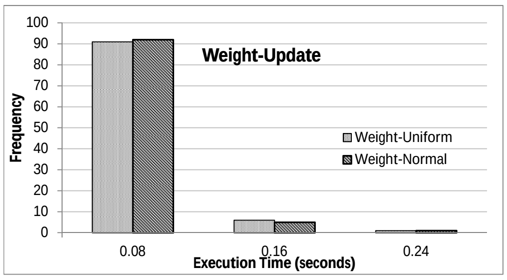

Figure 12: Distribution of the execution times of the Weight-Update operation on S_2(uniform weight) and S_3(normal weight)

Update operations With this experiment, we focus on the effect of the weight distribution of the object datasets on the Insert and Weight-Update operations. The effect on the Delete operation is qualitatively similar. We selected one series of data from the S_2(uniform weight) and S_3 (normal weight) with 1000 site points and 20000 object points. Thereafter, we applied the CMaxRNN approach and computed its execution time in response to the Insert and Weight-Update operations. The resulting execution time was computed by averaging the execution time over 100 runs of each operation. The method we used for performing 100 runs of each operation is as discussed in the previous experiments. Our results showed that in both operations the average of execution time for object datasets with uniform and normal weight distribution is similar. To verify this impact, we studied the distribution of execution times for each operation. Figure 11 presents the distribution of execution time in response to an Insert operation. As illustrated, the distribution of execution times with normal weight (the hashed bars) is more frequent in low values compared to those with uniform weight (the dotted bars). This is because, in object datasets with normal distributions, there exists a number of objects with high weight values which dominate the MaxRNN set. The more weight values in the MaxRNN set means less number of $I-OBJ$ which results in lower execution times in the CMaxRNN query.

Also, we observe in response to a Weight-Update operation (Figure 12), the distribution of execution times with different weight distribution is similar. As mentioned earlier , the execution time of a Weight-Update is proportional to the number of $I-EDG$. On the other hand, changes in the weight of objects only affects the weight of the $I-EDG$ not their quantities. Therefore, the performance of Weight-Update operations is not influenced by the changes in the weight of objects.

8.2.3 *Case Study*

With this experiment, we studied the behavior of the CMaxRNN algorithm in response to changes of the location of 24 food trucks roaming on LA county in a sample day. We retrieved their location data from TruxMap from 10:00 to 9:00p.m. and stored this information in a time table. As presented in Table 3, the minimum interval between the arrival of two trucks or the arrival of a truck and the departure of another truck is 2 minutes. The maximum interval is 2 hours and 24 minutes. As for the object dataset, we selected the real object dataset (9662 objects), aggregated their weights and created a new set with 483 objects. We assumed that we offer a decision making support system for food truck owners that they could ask our system for the optima location before they decide to change the location of their trucks. Thereafter, we performed the CMaxRNN queries and computed execution times. Figure 13 presents executions time for CMaxRNN for the interval from 9:57 to 11:11a.m. in response to the Insert operation. Figure 14 presents the execution time of CMaxRNN in response for the Delete operation and the same trucks studied in Figure 13 (but for the interval between 12:50 and 2:07p.m.). Table 3 provides a summary of the result of this experiment and shows how the CMaxRNN returned the optimal result in a couple of seconds whereas the minimum interval between two operations is about 2 minutes. Therefore, CMaxRNN supports continuous answers to MaxRNN queries for a real-world application. Also, comparing the execution time of SMaxRNN and the average in CMaxRNN verifies that the repeated execution

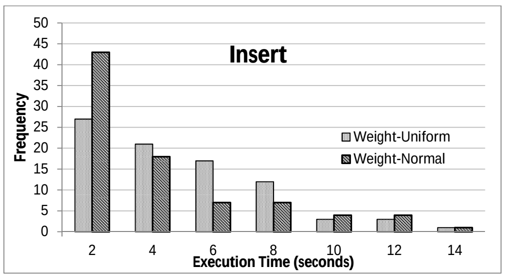

Figure 11: Distribution of the execution times of the Insert operation on S_2(uniform weight) and S_3(normal weight)

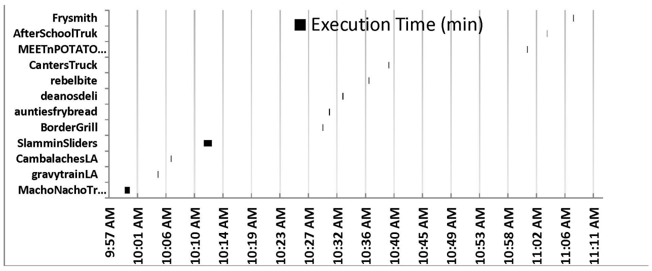

Figure 13: Execution times of CMaxRNN in response for Insert operations in the interval from 9:57 to 11:11a.m.

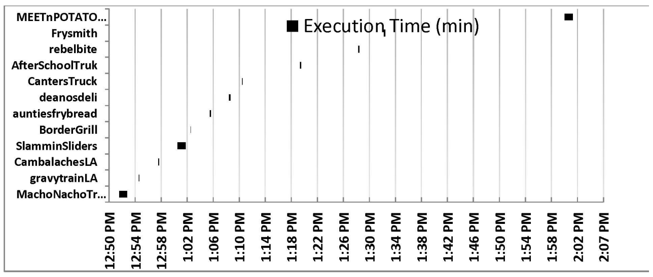

Figure 14: Execution times of CMaxRNN in response for Delete operations in the interval from 12:50 to 2:07p.m.

of SMaxRNN is not feasible to provide continuous answers for MaxRNN queries. We also observed that the execution time of CMaxRNN queries in response to both Delete and Insert operations is proportional to the number of impacted objects. With a larger number of impacted objects, expanding their corresponding local networks and identifying the optimal location, takes more time.

Table 3: Summary of the result of CMaxRNN queries on FoodTrucks

Number of food trucks	24
Number of Insert operations	24
Number of Delete operations	24
Minimum interval between two operations	2 minutes
Maximum interval between two operations	2 hour and 24 minutes
SMaxRNN execution time	5.14 minutes
Average of CMaxRNN execution time in response to an Insert operation	19 seconds
Average of CMaxRNN execution time in response to a Delete operation	41 seconds
Average of impacted objects in response to Insert operations	13
Average of impacted objects in response to Delete operations	23

9. CONCLUSION AND FUTURE WORK

This study is the first time to introduce the problem of continuously maximizing bichromatic reverse nearest neighbor for objects and sites located on spatial networks. Accordingly, we proposed an incremental approach for efficient computation of these queries. We evaluated and showed the efficiency of our proposed solution with rigorous complexity analysis as well as extensive experimental study, using real-world and synthetic datasets.

We intend to extend this study. With the proposed approach we assumed that multiple operations received sequentially are responded serially in a row. Thus, the execution time of computing CMaxRNN for multiple operations is equal to the sum of their individual execution times. However, in some real-world applications (e.g. disaster-response planning) requests for multiple operations might happen simultaneously. Accordingly, we plan to develop a batch solution for CMaxRNN queries and parallelize some steps toward providing faster response for multiple operations. For instance, in response to multiple operations, the steps of retrieving impacted objects, removing their local networks and expanding their new local networks can be accomplished simultaneously. Thereafter, all computed markers can be applied to MET at the same time and the optimal location can be identified as a one-step process.

10. REFERENCES

[1] R. H. Ballou. Dynamic warehouse location analysis. *Journal of Marketing Research*, 5:271–276, 1968.

[2] R. Benetis, S. Jensen, G. Karciauskas, and Saltenis. Nearest and reverse nearest neighbor queries for moving objects. *VLDB Journal*, 15(3):229–249, 2006.

[3] M. A. Cheema, X. Lin, W. Zhang, and Y. Zhang. Influence zone: efficiently processing reverse k nearest neighbour queries. In *ICDE*, 2011.

[4] M. A. Cheema, W. Zhang, X. Lin, Y. Zhang, and X. Li. Continuous reverse k nearest neighbors queries in euclidean space and in spatial networks. *VLDB Journal*, 21(1):69–95, Feb. 2012.

[5] R. Z. F. Z. Drezner and N. Asgari. Single facility location and relocation problem with time-dependent weights and discrete planning horizon. *Annals of Operations Research*, 167(1):353–368, 2009.

[6] Z. Drezner and G. O. Wesolowsky. Facility location when demand is time dependent. *Naval Research Logistics*, 38:763–777, 1991.

[7] Y. Du, D. Zhang, and T. Xia. The optimal location query. In *SSTD*, pages 163–180, 2005.

[8] D. Erlenkotter. A comparative study of approaches to dynamic location problems. *European Journal of Operational Research*, 6:133–143, 1975.

[9] M. Erwig and F. Hagen. The graph voronoi diagram with applications. *Networks*, 36:156–163, 2000.

[10] P. Ghaemi, K. Shahabi, J. P. Wilson, and F. Banaei-Kashani. Optimal network location queries. In *ACMGIS*, pages 478–481, 2010.

[11] J. M. Kang, M. F. Mokbel, S. Shekhar, T. Xia, and D. Zhang. Continuous evaluation of monochromatic and bichromatic reverse nearest neighbors. In *ICDE*, 2007.

[12] M. Safar, D. Ebrahimi, and D. Taniar. Voronoi-based reverse nearest neighbor query processing on spatial networks. *Multimedia Systems*, 15(5):295–308, 2009.

[13] H. Sun, C. Jiang, J. Liu, and L. Sun. Continuous reverse nearest neighbor queries on moving objects in road networks. In *WAIM*, pages 238–245, 2008.

[14] G. O. Wesolowsky. Dynamic facility location. *Management Science*, 7:1241–1248, 1973.

[15] G. O. Wesolowsky and W. G. Truscott. The multiperiod location-allocation problem with relocation of facilities. *Management Science*, 22:57–65, 1975.

[16] R. C. Wong, M. T. Ozsu, P. S. Yu, A. W. Fu, and L. Liu. Efficient method for maximizing bichromatic reverse nearest neighbor. In *VLDB*, pages 1126–1149, 2009.

[17] W. Wu, F. Yang, C. Chan, and K. Tan. Continuous reverse k-nearest-neighbor monitoring. In *MDM*, 2008.

[18] T. Xia and D. Zhang. Continuous reverse nearest neighbor monitoring. In *ICDE*, 2006.

[19] X. Xiao, B. Yao, and F. Li. Optimal location queries in road network databases. In *ICDE*, 2011.