Optimal Network Location Queries

Parisa Ghaemi, Kaveh Shahabi, John P. Wilson[*], Farnoush Banaei-Kashani

Computer Science Department
University of Southern California
Los Angeles, CA 90089

[ghaemi, kshahabi, jpwilson, banaeika]@usc.edu

ABSTRACT

Given a set S of sites and a set O of weighted objects, an optimal location query finds the location(s) where introducing a new site maximizes the total weight of the objects that are closer to the new site than to any other site. With such a query, for instance, a franchise corporation (e.g., McDonald's) can find a location to open a new store such that the number of potential store customers (i.e., people living close to the store) is maximized. Optimal location queries are computationally complex to compute and require efficient solutions that scale with large datasets. Previously, two specific approaches have been proposed for efficient computation of optimal location queries. However, they both assume p-norm distance (namely, L_1 and L_2/Euclidean); hence, they are not applicable where sites and objects are located on spatial networks. In this paper, we focus on *optimal network location* (*ONL*) queries, i.e., optimal location queries with which objects and sites reside on a spatial network. We introduce two complementary approaches, namely *EONL* (short for *Expansion-based ONL*) and *BONL* (short for *Bound-based ONL*), which enable efficient computation of ONL queries with datasets of uniform and skewed distributions, respectively. Moreover, with an extensive experimental study we verify and compare the efficiency of our proposed approaches with real datasets, and we demonstrate the importance of considering network distance (rather than p-norm distance) with ONL queries.

Keywords

Optimal Location Query, Query Processing in Spatial Network Databases

1. INTRODUCTION

Optimal location queries have been widely used in spatial decision support systems and marketing in recent years. For instance, a city planner might want to know: "What is the optimal location to open a new public library?" The optimal location is the site that would maximize the number of patrons for whom this is the closest library. Optimal location query is formally defined as follows: Given a set S of sites and a set O of weighted objects the optimal location query computes a location where introducing a new site would maximize the total weight of objects that are closer to the new site than to any other site.

Optimal location queries are computationally complex to answer. The existing work considers L_1 distance metrics or L_2/Euclidean as the measure of distance between objects and sites and proposes efficient solutions in these p-norm metric spaces [13, 5]. However, with many real world applications objects and sites are located on a spatial network (e.g., roads, railways, and rivers), and therefore, the approaches that assume p-norm distance fail to apply. We show this by an example as follows. Fig. 1.a (Fig. 1.b) compares the result of a simple optimal location query assuming L_2 (L_1) distance between objects and sites versus the result of the same query assuming the actual distance on the spatial network (i.e., the network distance). With this sample query, a set S of two sites S_1 and S_2, and a set O of three objects O_1, O_2, and O_3 with equal weights are located on a road network (shown by thick lines). Fig. 1.a depicts the approach proposed for optimal location query computation in L_2 space [13], where the intersection of multiple circles represents the identified optimal region R_1. As shown, the optimal region R_1 and the actual optimal network location, i.e., the network segment n_1n_2, are completely disjoint. Similarly, Fig. 1.b illustrates the optimal location query approach proposed for L_1 space [2]. The hatched area (comprising the rectangular areas R_2 and R_3) is the optimal region in L_1 space, which significantly overestimates the actual optimal location n_1n_2. We further verify the importance of assuming network distance with ONL queries in Section 6 via experiments, and we show that in 75% of the cases the results of optimal location queries in L_1 and L_2 spaces are totally disjoint from the actual optimal network location, with less than 20% overlapping in the rest of the cases.

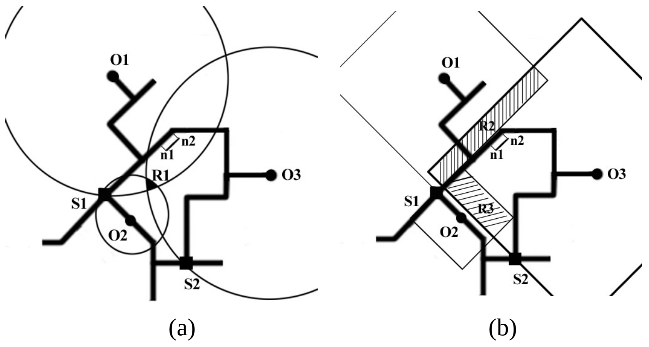


Fig. 1. Optimal Location Query (a) L_2 space result versus network space result, (b) L_1 space result versus network space result

In this paper, for the first time we introduce two complementary approaches for efficient computation of ONL

[*] John P. Wilson is a Professor of Computer Science, a Professor of Geography, and Director of GIS Research Laboratory at University of Southern California.

queries, namely *EONL* (short for *Expansion-based ONL*) and *BONL* (short for *Bound-based ONL*), which enable efficient computation of ONL queries with object-datasets of uniform and skewed distributions, respectively. We argue that the dominating computational complexity with ONL queries is twofold (this also applies to regular optimal location queries). To answer any ONL query, first one has to compute a spatial locality 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*. Second, one must compute the overlapping among object localities to identify the optimal network location, which is a network segment (or a set of segments) where the localities of a subset of objects with maximum total weight overlap.

Accordingly, with our two proposed algorithms, EONL and BONL, we focus on reducing the computational complexity of the latter and the former steps in ONL query answering, respectively. In particular, with Expansion-based ONL (EONL) we simply compute the locality of an object by expanding the network around the object until we reach the nearest site *s* to the object; hence, a costly computation at the first step of ONL query answering. However, during network expansion we identify and record potential overlapping between localities of the objects to avoid redundant computation at the second step; thus, efficient computation of overlapping among object localities at the second step. On the other hand, with Bound-based ONL (BONL), at the first step we avoid the costly network expansion and instead approximate object localities by an upper bound. In particular, we introduce two bound estimation techniques, which correspondingly result in two variations of BONL. Subsequently, at the second step we compute the overlapping among the actual object localities by network expansion, only if object bounds overlap.

Our experimental results with real datasets show that given uniformly distributed object-datasets (i.e., datasets with uniform *spatial* distribution), EONL is an order of magnitude faster than BONL, whereas with object-datasets of skewed distribution BONL outperforms EONL. We attribute the difference in efficiency of the two approaches with the two types of datasets to the fact that with skewed/clustered datasets, there is less overlapping between locality bounds of the objects; hence, less need for expansion at the second step. In real-world, skewed and uniform distribution of the object-datasets correspond to, for example, typical distribution of people/customers in urban and rural areas, respectively. Therefore, EONL and BONL have their own exclusive use-cases in real-world applications and are complementary.

The key contributions of this paper can be summarized as follows:

1. We define and formalize the optimal network location query problem.
2. We introduce two complementary approaches for efficient computation of optimal network location queries.
3. We experimentally compare our proposed approaches and discuss their use-cases with different real-world applications.

The remainder of this paper is organized as follows. Section 2 formally defines optimal network location queries in spatial network databases. Sections 3 and 4 introduce our proposed expansion-based and bound-based solutions for optimal network location queries, respectively. In Section 5, we present complexity analysis of our proposed approaches. In Section 6, we evaluate our proposed solutions via experiments with real data. Section 7 reviews the related work, and finally Section 8 concludes the paper and discusses directions for future research.

2. PROBLEM FORMALIZATION

In this section, we formalize the problem of optimal network location as a Maximum Overlap Segment (MaxOSN) problem. Assume we have a set *S* of sites (e.g. public schools, libraries, restaurants) in a 2D environment. Also we have a set *O* of objects with a weight *w(o)* for each object *o*. For instance, object *o* might be a residential building/property where *w(o)* represents the number of people living in this building. A MaxOSN query returns a subset of the spatial network (i.e., a segment or collection 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. We assume both sites and objects are located on a spatial network, e.g., a road network. We model the road network as a graph G (N, E), where *N* is the set of intersections/nodes and *E* is the set of edges of the road network. Each edge $e(n_1, n_2)$ has a travel cost. In this paper, we assume the cost of each edge *e* is proportional to the distance between the two end points n_1 and n_2 of *e*. Accordingly, the network distance $dN(n_1,n_2)$ between any two nodes n_1 and n_2, is the travel cost of the path with least cost from n_1 to n_2. Fig. 2 shows a road network with 14 nodes and weighted edges, four objects o_1, o_2, o_3, and o_4 with weights 3, 6, 5, and 4, respectively, and three sites s_1, s_2, and s_3.

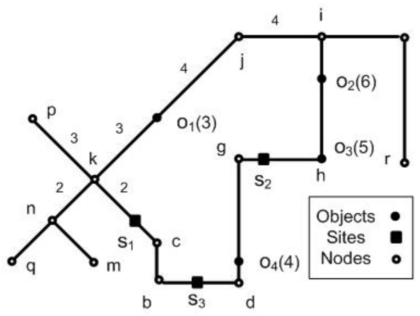


Fig. 2. Road network model

Below, we first define our terminology. Thereafter, we describe the MaxOSN query problem.

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.*

In Fig. 3, 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 (shown as arrows in Fig. 3). We term this delimitation process *edge marking*. The expanded network (shown as bold lines in Fig. 3) 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. 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 to 3 (i.e., the weight of object o_1).

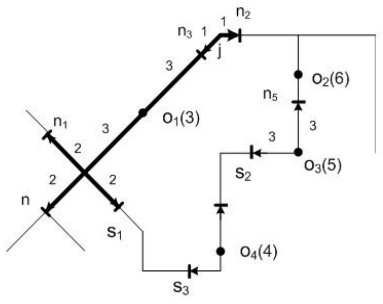


Fig. 3. Local networks

DEFINITION 2 (OVERLAPPING LOCAL NETWORKS). *A local network $LN(o_1)$ overlaps a local network $LN(o_2)$ if there exists a local edge e_1 in $LN(o_1)$ which intersects a local edge e_2 in $LN(o_2)$.*

For instance, in Fig. 3 $LN(o_1)$ 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 $LN(o_2)$.

DEFINITION 3 (OVERLAP SEGMENT). *Given two overlapping local networks, an overlap segment s is a network segment where the overlapping edges of the two local networks intersect. The influence value of segment s is equal to $I(e_1)+I(e_2)$, where e_1 and e_2 are the overlapping local edges from the two local networks.*

For example, the overlap segment jn_2 is identified by overlapping the local edges o_1n_2 and o_2n_3 in Fig. 3. The definition of the overlap segment can be generalized for more than two local edges: Given multiple local networks and multiple marked end points on each edge, the overlap segment on the edge can be identified by considering the direction and length of the overlapping local edges (in Section 3, we will discuss this process, called *edge collapsing*, in detail). For instance, Fig. 4 shows the overlap segment *jk* identified by overlapping local edges *ak*, *ai*, *bl* and *bj*.

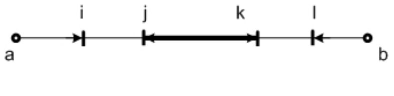


Fig. 4. Overlap segment of multiple local networks

DEFINITION 4 (MAXIMUM OVERLAP SEGMENT QUERY (MaxOSN)). *Given a set O of objects, a set S of sites, the MaxOSN query returns one or more overlap segment(s) which have the maximum influence value (I_0) among all overlap segments.*

For instance, in the road network illustrated in Fig. 3 the MaxOSN query returns the set of overlap segments {o_3S_2, o_3n_5}, where each segment has an optimal influence value I_0 =11.

LEMMA 1. *The MaxOSN query computes an optimal network location where introducing a new site would maximize the total weight of objects that are closer to the new site than to any other site. In other words, the MaxOSN query is equivalent to the definition of ONL query.*

PROOF. The proof is obvious. □

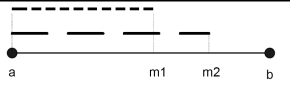
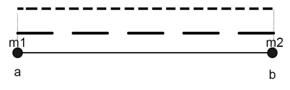
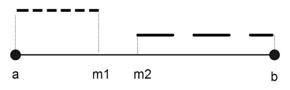
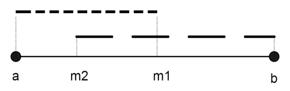
3. Expansion-Based Optimal Network Location (EONL)

As we mentioned in Section 1, answering an ONL query is a two-phase process. At the first phase, one needs to build the local networks of all objects, whereas at the second phase local networks of the objects are overlapped in order to identify the overlap segment(s) with maximum influence value (i.e., the optimal location/segment). With EONL, we focus on reducing the computational complexity of the second phase.

In particular, at the first phase EONL simply uses network expansion to build the local networks. At the second phase, assuming we have *n* objects (and therefore, *n* local networks), one should compute the overlap between 2^n combinations of local networks. In this case, if (for example) one of the network range-query processing techniques proposed by Papadias et al. [10] is used for overlap computation, the total computational complexity would be in the order of O ($2^{|O|}$ (|N| log |N| + |E|)). Obviously, this approach is not scalable. Instead, with EONL we identify the *potential* optimal segments while expanding local networks at the first phase, and leverage this information at the second phase to efficiently compute the segment(s) with maximum influence value. To be specific, while expanding the local networks at the first phase, for each edge we record all ending points (i.e., the points that mark the border of the local networks of the objects) that lie over the edge. Subsequently, at the second phase we use the information recorded at the first phase to compute a score for each edge, which is equal to the total weight of the objects whose local networks end on the edge. One should observe that a higher score for an edge indicates higher potential of containing an optimal segment. Next, through a refinement process we sort the edges based on their scores in descending order, and starting from the edge with higher score, we use a technique, termed *edge collapsing*, to compute the *actual* overlap segment(s) on each edge. It is important to note that through this refinement process we only have to compute the actual overlap segment(s) for an edge if the score of the edge is more than the influence value of the actual segments computed so far. With our experiments, we observe that EONL only computes the actual overlap segments for a limited subset of the network edges before it identifies the optimal location/segment; hence, effective pruning of the search space for better efficiency.

Below, we explain how we implement EONL in more detail; we begin by introducing our edge collapsing technique. Table 1 represents four possible cases by which two local edges e_1 and e_2 might overlap each other. The dashed lines represent local edges e_1 and e_2, the solid line represents the actual edge *ab* of the road network, and m_1 and m_2 are the end points of e_1 and e_2. The third column summarizes how the edge collapsing technique computes the overlap segment (S_0) with the maximum influence value (I_0) in each case. Next, we describe the implementation of EONL as a four-step algorithm (see Algorithm 1):

Table 1. Edge collapsing technique

Case	Overlapping Local Edges	Overlap Segment
1		$S_0 = m_1m_2$ $I_0 = I(e_1) + I(e_2)$
2		$S_0 = ab$ $I_0 = I(e_1) + I(e_2)$
3		If $(I(e_1) > I(e_2))$ $S_0 = a\,m_1$; $I_0 = I(e_1)$ Else $S_0 = b\,m_2$; $I_0 = I(e_2)$
4		$S_0 = m_2m_1$ $I_0 = I(e_1) + I(e_2)$

Step 1 (Expanding local networks) (lines 1-6): For each object point o, we first expand the local network of object *o*, *LN(o)*, using the Dijkstra algorithm. Then, we mark the ending points of the local networks on the edges. Table 2 shows sample subsets of the marked edges of Fig. 2. Each row of the Marked Edge Table (MET) is an entry in the form of (*e*, *M*, *w(e)*) where M is the set of ending points marked on edge *e* and *w(e)* is equal to the sum of influence values of the end points (i.e., the corresponding local edges).The MET table helps us to identify the overlapping segments with the maximum influence value.

Table 2. Marked Edge Table (MET)

E	M	w(e)
Kp	$\{n_1\}$	3
Ji	$\{n_2\}$	3
Ih	$\{n_5\}$	5
Hg	$\{S_2,S_2\}$	11

Step 2 (Sorting MET Table) (lines 7-9): We sort all entries in the MET table in descending order of *w(e)* because of our observation that the optimal solution is mostly derived from the entries with larger *w(e)* values. Then, we apply the edge collapsing technique to the first entry of MET and initialize the S_0 *and* I_0 values.

Step 3 (Identifying overlap segments) (lines 10-19): From the set of marked edges in MET table, we identify the overlap segments using the edge collapsing technique. We could apply the edge collapsing technique to all marked edges; however, we do not need to apply this approach for some marked edges if there is another marked edge whose influence value has a greater value than I_0. Thus, we can prune some marked edges using an *edge pruning* technique based on the following lemma.

LEMMA 2 (EDGE PRUNING). *Let I_o be the optimal influence value. An optimal network location is a collection of overlap segments that does not involve any marked edge e where $w(e) < I_o$.*

PROOF. We prove by contradiction. Assume the optimal solution involves a marked edge *e* where $w(e) < I_o$. This fact contradicts the definition of the optimal solution which ensures that I_0 has always the maximum influence value. □

Based on Lemma 2, we can prune any marked edge with $w(e) < I_o$.

Step 4 (Finding the maximum influence value): When the algorithm terminates, S_0 returns the set of optimal overlap segment(s) and I_0, the optimal influence value.

Algorithm 1 EONL Algorithm

```
1:  For each o ∈ O
2:      Expand the local network of object o
3:      Mark ending points on edges
4:  For each marked edge e ∈ E
5:      M=  Set of ending points marked on  e
6:        w(e)= Sum (w(m)| m ∈ M)
7:  Sort MET table based on w(e)
8:  Apply edge collapsing to the first entry of MET ($e_1$)
8:  $S_o$ ={ $g_1$ | maximum overlap segment on $e_1$ }
9:  $I_o$ = w($g_1$)
10: For each marked edge e ∈ E
11:     If w(e) ≥ $I_o$
12:       Apply edge collapsing to edge e
13:       g = maximum overlap segment on e
14:       Update w(e) : w(e)= w(g)
15:        If w(e) > $I_o$
16:            $I_o$ = w(e)
17:            $S_o$ = {g}
18:       If w(e) = $I_o$
19:            $S_o$ = $S_o$ ⋃ {g}
```

Here, we illustrate application of the EONL algorithm using the example depicted in Fig. 2. Assume we have performed the local network expansion for four objects o_1, …, o_4 and all ending points are marked on edges as shown using arrows in

Fig. 3. We construct the MET table and sort its entries based on their *w(e)* values. The first edge in MET table is *hg*. By applying the edge collapsing technique to *hg* we initialize the optimal solution set S_0 to {o_3s_2} and the optimal influence value I_0 to 11. Then, we perform the iterative steps on each marked edge. Among 14 marked edges from the road network shown in Fig. 2, only the marked edge *ih* satisfies the condition $w(e) \geq I_0$. Thus, we only apply the edge collapsing technique to it. By applying edge collapsing on *ih*, the overlapping segment o_3n_5 is derived which leaves I_0 unchanged and $S_0 =$ {o_3n_5} $\cup$ {o_3s_2}. At this point the algorithm terminates since all marked edges eligible for edge collapsing have been processed. Therefore, the optimal network queries on the dataset shown in Fig. 3 returns overlapped segments {o_3n_5, o_3s_2} with an optimal influence value of 11.

4. Bound-Based Optimal Network Location (BONL)

Similar to EONL, our bound-based optimal network location (BONL), is implemented as a two-phase process. However, with BONL we avoid the computational complexity of network expansion at the first phase by approximating the local networks with their corresponding spatial bounds. In particular, we define a (circular) spatial bound around each object *o* such that it is guaranteed to contain the local network of the object. For example, given an object point *o* and its nearest site *s* in the spatial network, one can use the Euclidean Restriction property [10] to define such a circular bound with radius equal to or greater than *dN(o,s)*, which guarantees containment of the local network of *o*. Fig. 5 shows the local bounds of four objects o_1, o_2, o_3, and o_4 as well as their corresponding local networks. The weight of local bound *lb* for an object, denoted by *w(lb)*, is defined to be equal to the weight of the corresponding object.

In order to form the local bound for an object using the Euclidean Restriction property, BONL must compute the (exact or approximate) distance between the object and its corresponding nearest site in the spatial network. Toward that end, we propose two variations of BONL. With BONL-U (i.e., BONL with upper bound), we approximate the local bound of an object by an *upper* bound which is derived using two different landmark selection techniques. On the other hand, with BONL-M (i.e., BONL with minimum bound), we introduce an efficient approach to compute the exact distance between an object and its nearest site. While BONL-M always provides a more accurate approximation of the local networks, with our study we also considered BONL-U as an option with potentially more efficient bound computation. We explain our bound computation approaches with BONL-U and BONL-M in Sections 4.1 and 4.2, respectively.

Here, assuming that local bounds (either upper bound with BONL-U or exact/minimum bound with BONL-M) are computed at the first phase of BONL, we explain the second phase of ONL query answering with BONL. At the second phase, we need to overlap the computed spatial bounds and prioritize the investigation of those overlapping areas that have a higher potential of covering the optimal segments (similar to the concept of MET table and edge collapsing technique with EONL). It is important to mention that overlapping spatial bounds helps us predict those areas that might cover the optimal segments. However, to identify the exact optimal overlap segments we need to expand the local networks of spatial bounds and retrieve the optimal overlap segments using edge collapsing technique. Below we explain our implementation of BONL in more detail.

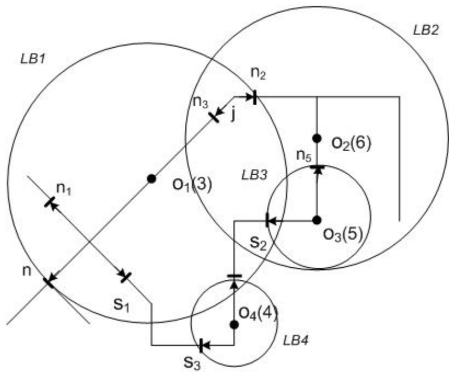


Fig. 5. Local bounds

With BONL, once local bounds of the objects are identified, for each local bound *lb* we find a list of other local bounds that overlap with *lb* and we call this list the overlapping list *OL(lb)* of *lb*. Lemma 3 defines the condition to identify overlapping bounds:

LEMMA 3. *Local bound lb_1 with radius r_1 overlaps local bound lb_2 with radius r_2 if and only if* $|r_1| + |r_2| \geq |o_1o_2|$.

PROOF. It is easy to verify that the above condition covers all the possible overlapping relationships between local bounds.□

Once the overlapping list for each local bound is generated, we construct a Pair-wise Overlapping Table (POT), where each row is an entry in the form (*lb*, *OL(lb)*). We call *OL (lb)* simply *OL*. The entries of POT are sorted in descending order of *w(OL)*, where $w(OL) = \sum_{lb \in OL} w(lb)$. Table 3 shows the POT constructed for the example depicted in Fig. 5.

Table 3. Pair-wise Overlapping Table (POT)

Lb	OL(lb)
lb_1	Lb_2, lb_3, lb_4
lb_3	lb_1, lb_2
lb_2	lb_1, lb_3
lb_4	lb_1

Finally, starting from the first entry, BONL processes each entry of POT to find the optimal segments as follows:

Step 1 (Expanding local networks): For each entry (*lb*,*OL*) in the POT table, we pick the *OL* list and expand the corresponding local networks as well as the local network of *lb* using the Dijkstra algorithm, while marking all ending points on each edge of the network.

Step 2 (Identifying overlap segments): For each entry, we identify the overlap segments for the overlapping local networks derived from the previous step using the edge collapsing technique described in Section 3.

Step 3 (Finding the maximum influence value): Among all identified overlap segments, we pick the one with the maximum I_0 value as the optimal solution.

It is important to note that we do not need to expand the local networks of some entries if there is another entry in the POT table whose actual influence value has a greater value. This means we can prune some entries from the POT table where $w(OL) + w(lb) > I_o$ and I_o is the current optimal influence in the current iteration.

4.1 Bound Based Optimal Location with Upper Bound (BONL-U)

With BONL-U, the upper bound value of the network distances between each object point and its nearest site is computed based on landmarks, and the Dijkstra algorithm [4]. This approach is inspired by the ALT algorithm of Goldberg et al. [6]. However, the lower bound of the shortest path distances in ALT is computed based on an A* search, landmarks and triangle inequality. Our upper bound value computation approach entails carefully choosing a small (constant) number of landmarks, then computing shortest path distances between all nodes of the spatial network and from/to each of these landmarks using the Dijkstra algorithm. Then, upper bounds are computed in constant time using these distances.

Calculating "From" and "To" distances: Since each edge of our experimental road network (LA County road network) is directional, we calculate the shortest path distances between all nodes of the road graph both "From" and "To" all Landmarks points using the Dijkstra algorithm (Fig. 6).

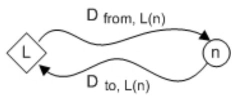


Fig. 6. Distance from and to landmarks

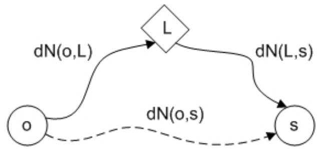


Fig. 7. Upper bound value calculation

Calculating Upper Bound Value: Fig. 7 illustrates how we compute the upper bound value of the network distance *dN(o,s), UdN(o,s).* The *UdN(o,s)* is calculated from the network distance *o* and landmark *L, dN(o,L),* and the network distance *dN(L,s)* as $UdN(o,s) = dN(o,L) + dN(L,s) \geq dN(o,s)$.

For calculating the upper bound value of object point *o* and its nearest site *s*, first we calculate the shortest path distance value between object *o* and all site points traversing all possible landmark points. Then, from all computed upper bound values we pick the one with the minimum value as the *UdN(o,s).*

Landmark Selection: Finding good landmarks is critical for the overall performance of upper bound value computation. The optimal approach is the one which contributes to the computation of an upper bound value very close to the actual value of the network distance. In the following we discuss two alternate techniques used for landmark selection: ***uniform and variable grid-based landmark selection.*** In both techniques we guarantee two landmarks are spatially located farther than a specific range from each other.

In the uniform landmark selection approach, we randomly select a constant number of landmarks in a series of grid cells spanning the Los Angeles (LA) County road network. With the variable approach, we select more landmarks in regions with more site points. For this purpose, we count the number of sites which falls within each cell, denoted by T_c. Then, we assign $k * T_c / |S|$ landmarks to this particular grid cell where k is the total number of landmarks, and |S| is the total number of site points in the entire dataset. The grid cells measured 10 km on a side, given that larger cells gave a result similar to the uniform selection strategy and grid sizes smaller than 10 km generate numerous grid cells with no assigned landmarks. Our experimental results (see section 6) show that the uniform landmark selection outperforms the grid-based landmark selection in terms of computation cost.

Our experimental results showed that the BONL-U algorithm has low performance due to the cost of the upper bound value computation for network distances using landmark selection. The drawback of using landmark selection is that the radius of local bounds (U*dN(o,s))* is always larger than the actual one. This fact produces large numbers of overlapping local bounds and local networks which leads to relatively high computation cost when using the BONL algorithm.

In the next section, we introduce the Bound Based Optimal Location Queries with Minimum Bound (BONL-M) in which we improve the local bounds by computing the actual network distance between objects and their nearest sites.

4.2 Bound Based Optimal Location with Minimum Bound (BONL-M)

With BONL-M, we compute the actual network distance value between each object point and its nearest site, *dN(o,s),* using the following three-step approach:

Step 1 (Reversing the road network graph): We first reverse the road network graph.

Step 2 (Calculating the network distance of each node to its nearest site): We then calculate the shortest distance between each node and its nearest site using the Dijkstra Algorithm. Toward this end, we run the Dijkstra algorithm from each site point *s*, and traverse all nodes of the graphs. By traversing each node *n*, we store a value called g_n, which represents the shortest path distance value between site *s* and node *n*. Each time we pick a new site *s′*, we check the g_n value while traversing the nodes and if the current g_n value is greater than the shortest distance value between node *n* and site *s′*, we update the g_n value and set it to the shortest distance value between n and s′. After processing all site points, the g_n values stored with the nodes represent the shortest path distances between the nodes and the nearest sites.

Step 3 (Computing the network distance of each object to its nearest site): We calculate the network distance of each object to its nearest site using the g_n values computed with the previous step.

This three-step approach calculate the actual network distance value, *dN(o,s)* and these values are used in place of the *upper bound values* used in BONL-U. Our experiments demonstrate that the BONL-M approach reduced the radius of the local bounds and improved the performance of the BONL-M algorithm compared to the BONL-U approach.

5. COMPLEXITY ANALYSIS

In this section, we analyze the computational complexity of our proposed approaches.

BONL-U: Below, we discuss the computational complexity of various tasks with BONL-U:

Landmark Selection: The running time of landmark selection step takes O (k^2 |N|) (Recall *k* is the number of selected landmark points).

Calculating "From" and "To" Distances: Given *k* landmark points, computing "From" and "To" distances takes O (*k* (|N| log |N| +|E|)) and O (|N| (|N| log |N| +|E|)), respectively. In total, the running time of upper bound value computation would be O (|N| (|N| log |N| + |E|)) which is extremely high in very large road networks. To improve the running time of this step we reverse the road graph (O (|E|)) and calculate the distances "From" landmarks to nodes. This technique improves the running time to O (*k* (|N| log |N| + |E|)).

Calculating Upper Bound Value: This step takes O (k |O| |S|) and the total running time for computing *Ud(N,S)* is O (k^2 |N|) + O (k (|N| log |N| + |E|)) + O (k |O| |S|).

Forming Local Bounds: This step takes O (1) time.

Constructing the POT Table: This step takes O (|O|) time since there are |O| local bounds.

Sorting the POT Table: Sorting takes O (|O| log |O|) running time.

Expanding the Local Networks: Since the maximum number of overlapping local bounds can theoretically be equal to |O|, the running time for expanding the local networks takes O (|O| (|N| log |N|) + |E|)). Then, we mark all ending points on edges which requires O (|E|)) time. Note that the edge marking step cannot be performed at the same time as expanding the Dijkstra algorithm in step 1. Because Dijkstra's algorithm assumes that the objects and sites fall on network nodes while in our scenario ending points may fall on edges.

Identifying Overlap Segments with the Maximum Influence Values: The edge collapsing technique takes O (|E| $O|^2$) time. Thus, considering |O| entries in POT table, edge collapsing step has a complexity equal to O (|E| $|O|^3$).

Finding the Maximum Influence Value: This task takes O (1) time.

The overall running time of BONL-U is O (k^2 |N|) + O (k(|N| log |N| + |E|)) + O(k |O| |S|) + O(|O|) + O(|O| log |O|) + O($|O|^2$ (|N| log |N| + |E| + |O| |E|)).

BONL-M: The complexity of BONL-M is similar to BONL-U but for the computation of the actual network distance values which requires the following steps:

Reversing the Road Network Graph: This step can be done in O (|E|) time.

Calculating the Network Distance of Each Node to Its Nearest Site: The running time of this step is O (|S| (|N| log |N| + |E|)).

Computing the Network Distance of Each Object to Its Nearest Site: This step can be done in O (|O|).

The overall running time of BONL-M is O (|E|) + O (|S| (|N| log |N| + |E|)) + O(|O|) + O(|O| log |O|) + O($|O|^2$ (|N| log |N| + |E| + |O| |E|)).

EONL: In this case, we reduce the running time of optimal network location query by eliminating the cost of upper bound/minimum upper bound value computation. The cost of constructing the MET table is O (|E|) and sorting the table O (|E| log |E|). The overall running time is O (|E| log |E|) + O (|O| (|N| log |N| +|E|)) +O (|E| $|O|^2$) because edge collapsing is performed only once for each edge. Thus, the complexity of this technique improves to O (|E| $|O|^2$) in EONL. However, in BONL-U and BONL-M we needed to apply edge collapsing for each entry of POT table.

6. EXPERIMENTAL EVALUATION

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

6.1 Experimental Setup

All experiments are performed on an Intel Core Duo 3GHz, 4 GB of Ram, running Windows 7 and the .NET platform 3.5. The algorithms are implemented in Microsoft C#. 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 real dataset for objects and sites. Objects are population data derived from LANDSCAN population data 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. In total we have |O|= 9662 objects. The weights of objects are distributed nearly uniformly with an average of 1100. For each experiment, we use a subset of object points selected from this base dataset that we will describe them in each part of experiments. We also deployed five datasets consisting of Johnny Rockets restaurants, McDonald's restaurants, hospitals, schools, and all fast food restaurants in LA County (including McDonald's and Johnny Rockets) for the sites. The cardinality of each site dataset is shown in Table 4. All sites, objects, nodes and edges are stored in memory-resident data structures.

Table 4. Five real datasets for sites

Datasets	Cardinality
Johnny Rockets	28
McDonald's	328
Hospitals	308
Schools	2621
Fast Foods	19160

6.2 Experimental Results

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

Accuracy: We first verified that the optimal location query in L_1 and L_2/Euclidean space is not applicable to spatial networks. For this test, we selected four datasets with 20, 40, 60, and 85 object points that were randomly selected from the population data (DS_1-DS_4). All four sets of object points were located on LA county road network. For site points, we selected a subset of McDonald's including seven sites. We applied the L_2 [13] and L_1 [5] distance approaches and identified the optimal location in each case. Then, we performed the EONL algorithms on each dataset and retrieved their corresponding optimal network location. The result of this experiment showed that in 75% of cases (we call it set A) the optimal locations derived by the L_1/L_2 approach did not overlap the optimal network location derived by EONL and when they did overlap, there was < 20% common coverage. Fig. 8 shows one of the non-overlapping cases of set A (the circles represent objects and triangles represents sites). From cases included in set A, the average distance between the optimal network location and the optimal location derived from the L_1 and L_2 approaches (<N, L_1>, <N, L_2 >) are similar to the size of the entire area covered by these datasets (see Table 5) and verifies that using the existing L_1 and L_2 approaches for optimal location queries on spatial network databases is not accurate and likely to return irrelevant results.

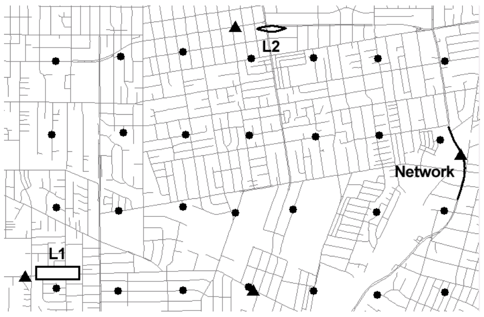


Figure 8. Non-overlapping case

We also observed that the maximum influence value returned by the optimal network location query is 13% and 12% higher than those returned by the optimal location queries in the L_1 and L_2 approaches, respectively and would therefore identify larger numbers of customers for those interested in running these kinds of queries.

Table 5. Average distance of optimal network location and optimal location derived by L_1 and L_2 approaches

(size of the entire area is 6.2 km x 9 km)

Dataset	<N, L_1> (meters)	<N, L_2 > (meters)
DS_1	Overlaps (< 20% coverage)	
DS_2	4998	5305
DS_3	4995	2743
DS_4	6663	6396
Average	5552	4814

Execution Time: In order to evaluate the execution times of our proposed approaches, we implemented two experiments. With the first one, we considered a fixed site-dataset and used various object-datasets with different sizes and spatial distributions. With the second experiment, we fixed the object-dataset and used various site-dataset. Below, we describe each experiment in more detail.

Effect of Object-Dataset: For this experiment, we sub-sampled four subsets of objects from the base dataset with sizes 366 (C1), 567 (C2), 1049 (C3) and 1533 (C4). We sub-sampled the objects with two different spatial distributions: uniform and skewed. To select each object, we randomly picked both X and Y dimensions of the grid cell corresponding to the object using a uniform or skewed distribution. As for the fixed site dataset, we picked the set of Johnny Rockets restaurants which has a small number of site data points comparing to other site datasets in Table 4. Thereafter, we applied BONL-M and EONL approach to the aforementioned datasets and computed the execution times (as we show later, BONL-M outperforms BONL-U, hence excluded from this experiment). Fig. 9 depicts the results of our experiment. We observe that when the size of the object-dataset is small (C1) and its distribution is skewed, the execution time of BONL-M is higher than EONL. This is because the cost of computing the radius of the local bounds (O (|E|) + O (|S| (|N| log |N| + |E|)) is comparable to the cost of expansion of local networks (O (|O| (|N| log |N| +|E|))) when the number of object points (|O|) is low. However, with the larger object-datasets (C2 to C4), the performance of BONL-M increasingly improves as compared to EONL, because with a skewed object distribution, the number of overlapping local bounds is significantly reduced. Therefore, the cost of overlap computation with BONL-M becomes less than the cost of local network expansion with EONL.

On the other hand, with uniformly distributed object points, the EONL outperforms BONL-M with all object-datasets (Fig. 10). This is because with uniform distribution of object points, the number of overlapping local bounds is always high which results in higher cost of identifying overlap segments as compared to the cost of local network expansion.

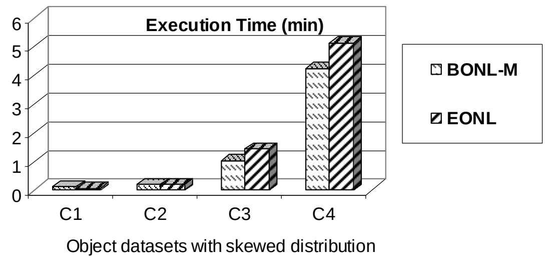


Fig 9. Execution times of the algorithms with fixed site dataset and skewed distributed objects

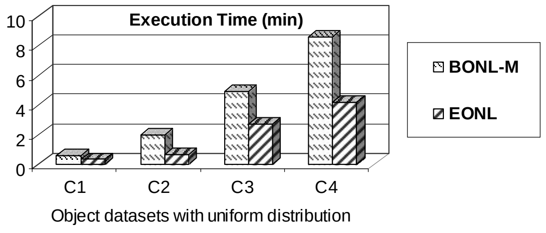


Fig 10. Execution times of the algorithms with fixed site dataset and uniformly distributed objects

Effect of Site-Dataset: For this experiment, we applied all three algorithms to the four sites datasets of Table 4 and we selected the uniformly distributed population data as the fixed object dataset (including all 9662 points). Thereafter, we computed the execution times to compare their performance. Fig. 11 shows that EONL has the highest performance, beating BONL-M and BONL-U by factors of 6 and 12 on average, respectively.

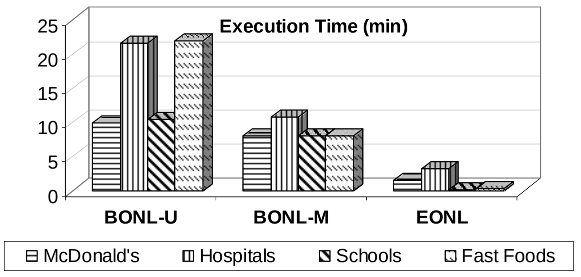


Fig. 11. Execution times of the algorithms with uniformly distributed objects

The three algorithms behaved similarly with the small McDonald's, Hospitals, and Schools datasets. The Hospitals sites were more skewed and this variability meant that the expansion and edge marking took longer in those parts of the graph with few hospitals (see Fig. 11). The most important result was the significant improvement in the execution time of EONL for the Fast Foods restaurants (despite their uniform distribution) because the complexity of EONL is O (|E| log |E|) + O (|O| (|N| log |N| +|E|)) +O (|E| $|O|^2$) which is not proportional to the number of site points, |S|.

Tightness of Local Bounds: The radius of the local bounds was improved an average of 53% by using BONL-M in place of the BONL-U algorithm. Fig. 12 shows how the radius of local bounds was reduced by using the BONL-M algorithm in place of BONL-U for each of the aforementioned datasets. Furthermore, we observed that the Hospital dataset has the highest average of local bound radius in both algorithms. With a non-uniform and more skewed site distribution, expansion of the local network traverses a longer path until it hits the nearest site. This fact results in producing larger local bounds comparing to other datasets.

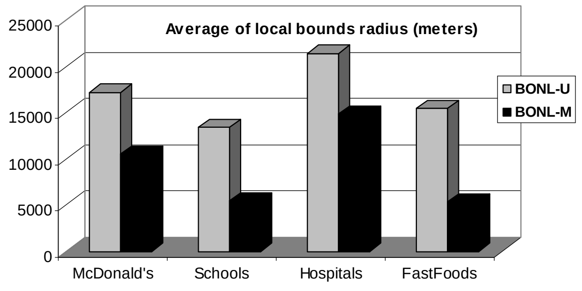


Fig. 12. Average size of local bounds

Landmark Selection: For this experiment, we selected 100 landmarks and applied the BONL-U algorithm to four datasets. The results in Table 6 show that the grid-based approach took more time for three of the four datasets and especially the McDonald's and Hospitals datasets which are sparser. This is because the grid-based approach assigns more landmark points to the areas with more site points; hence, it takes more time for Dijkstra expansion in the areas with lower site density.

Table 6. Comparing the execution time of BONL-U with the two landmark selection techniques

	McDonald's	**Hospitals**	**Schools**	**Fast Foods**
BONL-U Uniform	10 min	21.5 min	10.5 min	22 min
BONL-U Grid-based	> 1hr	> 1hr	10.5 min	31 min

7. RELATED WORK

Optimal location queries have been studied by researchers in operations research (OR) and database systems. In OR, most optimal location problems (also called facility location problems) are formulated as *covering problems*. These involve locating *n* sites to cover all or most of the (so-called) demand objects assuming a fixed service distance for sites. Covering problems are generally classified into two main classes. The first is the Location Set Covering Problems (LCSPs) [12] that seek to position a *minimum number of sites* in such a way that each and every demand object has at least one site placed within some threshold distance. The second class is the Maximal Covering Location Problems (MCLPs) [3] which seek to establish a set of *m* sites to maximize the total weight of the "covered" objects, where an object is considered covered if it is located within a specified distance from the closest facility. Many other problems in this class extend the original MCLP by imposing various placement restrictions for sites [8, 2], assuming various types of objects (points, lines and polygons) [9], and considering various definitions for coverage [1].

While OR-based solutions are effective and address various types of optimal location problems, many of these solutions fail to scale with real datasets that consist of large numbers of sites and objects due to their computational complexity. Accordingly, a number of complementary solutions are proposed by the database community for scalable optimal location query answering.

One is the Bichromatic Reverse Nearest Neighbor (BRNN) query [7, 15, 11] studied by the database community. With the BRNN query, all objects $o \in O$ whose nearest neighbor site is *s* are returned. The optimal location query can be formulated as a BRNN maximization problem, with which we try to locate a new site *s* such that the size of the BRNN set of *s* is maximized; hence, BRNN and optimal location are orthogonal problems. Another relevant problem involves finding the top-k most influential sites [14]. Here, the influence of a site *s* is defined as the total weight of the objects in a BRNN set of *s*. With this problem, a set of existing sites are assumed among which we want to find the most influential sites, whereas with the optimal location problem, the goal is to locate a new site with maximum influence.

Wong et al. [13] and Du et al [5] tackle the optimal location problem. Both approaches form a spatial bound around each object *o* such that it includes a location *l* if and only if *o* is closer to *l* than to any other site. The intersection areas where these bounds overlap are the best candidate locations to introduce a new site. Therefore, to compute the optimal location query one can start with the areas with the maximum number of overlapping bounds and avoid other areas to reduce the search space and improve the query efficiency. While efficient, both of the aforementioned approaches assume p-norm space (namely, [13] assumes L_2 and [5] assumes L_1), which as shown in Fig. 1 cannot support optimal location queries on spatial networks. Our proposed solutions utilized network distance to address optimal network location queries.

8. CONCLUSION AND FUTURE DIRECTIONS

In this study, for the first time we introduced the problem of optimal location for objects and sites located on spatial networks. Accordingly, we proposed EONL and BONL as two complementary approaches for efficient computation of optimal network location queries with datasets of different spatial distributions. In particular, we showed that avoiding network expansion with BONL is more effective when the given object-dataset has a skewed spatial distribution, whereas EONL outperforms BONL with uniformly distributed objects. We verified and compared the performance of our proposed solutions with rigorous complexity analysis as well as extensive experimental evaluation with real data.

We intend to extend this study in two ways. First, with the optimal network location problem, alike all previous work we assumed a site covers an object if and only if the site is the closest site to the object. We plan to study optimal network location queries under a more generalized definition of coverage where a site covers an object based on a combination of mutual relationships (not only proximity), such as accordance of site properties (e.g., hotel amenities) with the requirements of an object (e.g., the interests of potential travelers). Second, we want to study a more complex optimal location problem setting where the sets of sites and/or objects might be located both on and off spatial networks. With this problem, we will investigate and develop hybrid solutions.

9. REFERENCES

[1] Berman, O., and Krass, D. 2002. The Generalized Maximal Covering Location Problems. *Computers and Operations Research*, 29, 6, 2002, 563-581.

[2] Church, R. L. 1984. The Planar Maximal Covering Location Problem. *Journal of Regional Science*, 24, 1984, 185-201.

[3] Church, R. L., and Revelle, C. 1974. The Maximal Covering Location Problem. *Papers of the Regional Science Association*, 32, 1974, 101-118.

[4] Dijkstra, E. W. A Note on Two Problems in Connection with Graphs. *Numeriche Mathematik*, 1, 1, 269-271.

[5] Du, Y., Zhang, D., and Xia, T.2005. The Optimal-Location Query. SSTD 2005, 163-180.

[6] Goldberg, A. V., and Harrelson, C. 2005. Computing the Shortest Path: A* Search Meets Graph Theory. ACM-SIAM 2005, 156-165.

[7] Korn, F., and Muthukrishnan, S. 2000. Influence Sets Based on Reverse Nearest Neighbor Queries. SIGMOD 2000, 29, 2, 201-212.

[8] Mehrez, A., and Stulman, A. 1982. The Maximal Covering Location Problem with Facility Placement on the Entire Plane. *Journal of Regional Science*, 22, 1982, 361-365.

[9] Murray, A.T., and Tong, D. 2007. Coverage Optimization in Continuous Space Facility Siting. *International Journal of Geographical Information Science*, 21, 7, 2007, 757-776.

[10] Papadias D., Zhang, J., Mamoulis N., and Tao, Y. 2003. Query Processing in Spatial Network Databases. VLDB 2003, 802-813.

[11] Stanoi, I., Riedwald, M., and El Abbadi, A. 2001. Discovery of Influence Sets in Frequently Updated Databases. VLDB 2001, 99-108.

[12] Toregas, C., Swain, R., Revelle, C., and Bergman, L. 1971. The Location of Emergency Service Facilities. *Operations Research*, 19, 6, 1971, 1363-1373.

[13] Wong, R. C., Ozsu, M. T., Yu, P. S., Fu, A. W., and Liu, L 2009. Efficient Method for Maximizing Bichromatic Reverse Nearest Neighbor. VLDB 2009, 1126-1149.

[14] Xia, T., Zhang, D., Kanoulas, E., and Du, Y. 2005. On Computing Top-t Most Influential Spatial Sites. VLDB 2005, 946-957.

[15] Yang, C., and Lin, K. I. 2001. An Index Structure for Efficient Reverse Nearest Neighbor Queries. ICDE 2001, 51-60.