Geoinformatica (2014) 18:229–251
DOI 10.1007/s10707-013-0179-x

A comparative study of two approaches for supporting optimal network location queries

Parisa Ghaemi · Kaveh Shahabi · John P. Wilson · Farnoush Banaei-Kashani

Received: 14 February 2011 / Revised: 5 March 2012 /
Accepted: 11 April 2013 / Published online: 28 April 2013

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 article, we focus on *optimal network location* (*ONL*) queries, i.e., optimal location queries in 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 world datasets, and we demonstrate the importance of considering network distance (rather than p-norm distance) with ONL queries.

Keywords Optimal location queries · Spatial network databases

P. Ghaemi (✉) · K. Shahabi · F. Banaei-Kashani
Computer Science Department, University of Southern California, Los Angeles, CA 90089, USA
e-mail: ghaemi@usc.edu

K. Shahabi
e-mail: kshahabi@usc.edu

F. Banaei-Kashani
e-mail: banaeika@usc.edu

J. P. Wilson
Spatial Sciences Institute, University of Southern California, Los Angeles, CA 90089, USA
e-mail: jpwilson@usc.edu

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. An 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 [5, 14]. 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 do not apply. We show this by an example as follows. Figure 1a (Fig. 1b) compares the result of a simple optimal location query assuming L_2 (L_1) distance between objects and sites vs. 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). Figure 1.a depicts the approach proposed for optimal location query computation in L_2 space [14], 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 7 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 remainder of the cases.

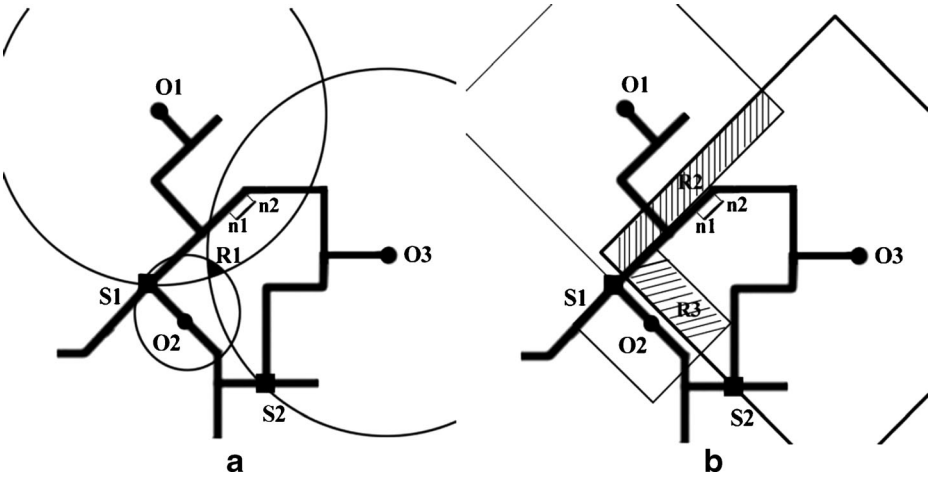


Fig. 1 Optimal location query **a** L_2 space result vs. network space result, **b** L_1 space result vs. network space result

In this article, we introduce two complementary approaches for efficient computation of ONL queries, namely *EONL* (short for *Expansion-based ONL*) and *BONL* (short for *Bound-based ONL*), which enable efficient computation of ONL queries for object-datasets with 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 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 neighborhood of o will be closer to o as compared to the distance between s and o. Second, one must compute the overlap among object neighborhoods to identify the optimal network location, which is a network segment (or a set of segments) where the neighborhoods of a subset of objects with the 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 queries, respectively. In particular, with Expansion-based ONL (EONL) we simply compute the neighborhood of an object by expanding the network around the object until we reach the nearest site s to the object. This is a costly computation at the first step of ONL query answering. However, we identify and record the potential overlaps between the neighborhoods of the objects during network expansion to avoid redundant computation at the second step; thus, ensuring efficient computation of overlaps among object neighborhoods 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 neighborhoods 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 overlap among the actual object neighborhoods 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* distributions), EONL is an order of magnitude faster than BONL, whereas with object-datasets with skewed distributions 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 overlap between neighborhood bounds of the objects; hence, less need for expansion at the second step. In the real-world, skewed and uniform distributions of the object-datasets correspond to, for example, the typical distributions 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 article 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 reviews the related work and Section 3 formally defines optimal network location queries for spatial network databases. Sections 4 and 5 introduce our proposed expansion- and bound-based solutions for optimal network location queries, respectively. In Section 6, we present the complexity analysis of our proposed approaches. In Section 7, we evaluate our proposed solutions via experiments with real world data. Section 8 concludes the paper and discusses directions for future research.

2 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) [13] 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 [2, 9], assuming various types of objects (points, lines and polygons) [10], 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 world 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 to support scalable optimal location queries.

One is the Bichromatic Reverse Nearest Neighbor (BRNN) query [8, 12, 17] by which 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 [15]. 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. [14] and Du et al. [5] tackled the optimal location problem by forming 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, [14] 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.

More recently, Xiao et al. [16] proposed a unified framework that addresses three variants of optimal location queries in road networks efficiently. One of these variants called *Competitive Location Queries* is the same problem we have defined as ONL queries in this article. To address this problem, 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, in their optimized method called FGP-OTF, they first partition the road network graph to sub-graphs and process them in descending order of their likelihood of containing the optimal locations. Their extensive experiments show the high performance of the FGP-OTF approach in terms of running time and memory consumption. However, our experiment in Section 7 shows that the FGP-OTF approach does not perform efficiently with a large road

network dataset and a set of object points with a nearly uniform weight distribution. Our experimental results illustrate that a single approach for ONL queries might not perform efficiently for different distributions of object and site points (e.g., uniform vs. skewed distributions).

Finally, the EONL approach was originally introduced in our prior work [6] which is performing efficiently with uniform distributions of object datasets. This article extends this prior work by introducing a complementary approach for efficient computation of optimal network location queries in datasets with skewed distributions.

3 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(a,b)* 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 *a* and *b* of *e*. Accordingly, the network distance *dN(a,b)* between any two nodes *a* and *b*, is the travel cost of the path with least cost from *a* to *b*. Figure 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.

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

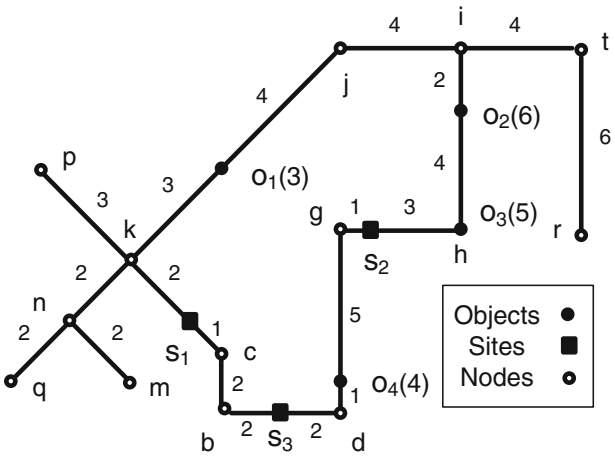


Fig. 2 Road network model

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* $s = argmin_{p \in S} dN(o,p)$. 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, namely *markers* (shown as arrows in Fig. 3). 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 Fig. 3). 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).

Definition 2 (Overlapping local networks) A local network $LN(o_1)$ overlaps a local network $LN(o_2)$ if $LN(o_1) \cap LN(o_2) \neq \emptyset$. In such case, there exists at least one 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 $LN(o_1)$ 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.:

$$s = \{q | q \in e_1, q \in e_2\}$$

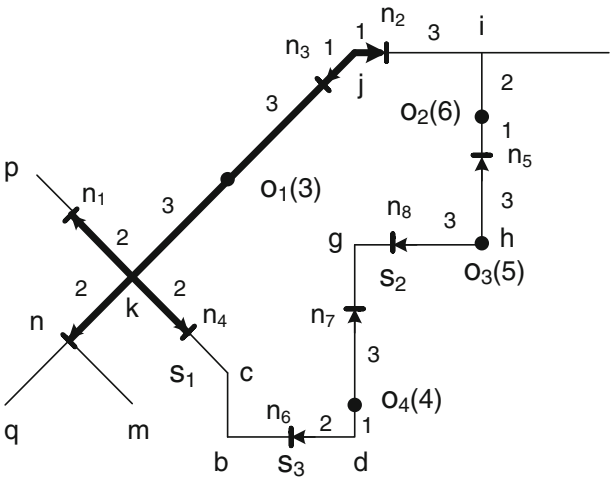


Fig. 3 Local networks

where $e_1 \in LN(o_1)$ and $e_2 \in LN(o_2)$ and $LN(o_1) \cap LN(o_2) \neq \emptyset$. Accordingly, the influence value of segment s, I_s,is defined as $I_s = I(e_1) + I(e_2)$.

For example, in Fig. 3 the overlap segment jn_2 is identified by overlapping the local edges o_1n_2 and o_2n_3. Also, its influence value is equal to 9. The definition of the overlap segment can be generalized for more than two local edges: Given multiple local networks and multiple markers 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 4, 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* and *bj*.

Definition 4 (Maximum Overlap Segment Query (MaxOSN)) Given a set O of objects and a set S of sites, the MaxOSN query returns the optimal network location p, the set of overlap segment(s) with maximum influence value (I_0):

$$p = \{s | s \in OS, s = argmax_{s \in 0S} I_s\}$$

where OS is the set of overlap segments.

For instance, in the road network illustrated in Fig. 3 the MaxOSN query returns the set of overlap segments $\{o_3n_8, o_3n_5\}$, where each segment has an optimal influence value $I_0 = 11$.

4 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 overlaid in order to identify the overlap segment(s) with the 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. [11] 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. 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 cover fully or partially that edge. The higher score for an edge the more likely it is to contain the optimal

Fig. 4 Overlap segment of multiple local networks

segment. Next, through a refinement process we sort the edges based on their scores in descending order, and starting from the edge with the highest 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, it provides effective pruning of the search space for better efficiency.

Below, we explain how we implement EONL in six steps (Algorithm 1):

Step 1 (Expanding Local Networks and Marking the Edges) (Lines 1–3): For each object point o, we first compute the local network of the object o, termed *LN (o).* We compute *LN (o)* by expanding the network (using the Dijkstra algorithm [4]) starting from o until we reach the site nearest to object o. Next, we mark the ending points/markers of the local networks on the edges and store the location of the markers.

Step 2 (Constructing the Marked Edge Table) (Line 4): Once markers are generated, we construct a *Marked Edge Table (MET).* Table 1 shows a sample subset of the marked edges in Fig. 3. Each row of the Marked Edge Table (MET) is an entry in the form of (*e*, *M, Sc(e)*), where M is the set of markers marked on the edge e (including the starting and ending node of the edge e), and *Sc(e)* is the score of e which is equal to the total weight of the objects whose local networks cover e (fully or partially). The MET helps us to identify the overlapping segments with the maximum influence value using the edge collapsing technique described in step 5.

Step 3 (Sorting the MET) (Line 5): Next, we sort all entries in the MET in descending order of *Sc(e)*.

Step 4 (Initializing the Optimal Result Set) (Line 6): The EONL algorithm will eventually return the set of optimal overlap segments (S_0) with the optimal influence value I_0. At this step, we initialize these two sets to empty sets.

Step 5 (Identifying the Overlap Segments using the Edge Collapsing Technique) (Lines 7–11): At this step, we use the set of marked edges in the MET to identify the optimal overlap segments using our *edge collapsing* technique as follows. First, we split the edge e to a set of segments, *SG(e)*, where each segment is a part of the edge e between two consecutive markers. Then, for each segment s of *SG(e)*, we determine the local networks that cover s. Accordingly, we compute the influence value for s as the sum of weights for the corresponding local networks. For each edge of the MET, the optimal overlap segment (*os*), is the segment which has the highest influence value among all segments in *SG(e)*. It is important to note that edge collapsing may produce more than one optimal overlap segment on each edge. As an example, consider the second marked edge of MET shown in Table 1. Since there are three markers on this edge e, we have $SG(e)=\{kn_3, n_3j\}$. The first segment, kn_3, is only covered by local network $LN(o_1)$. However, the second segment, n_3j, is covered by two local networks $LN(o_1)$ and $LN(o_2)$ which results in a higher influence value compared to

Table 1 Marked Edge Table (MET)

e	M	Sc(e)
kp	{k, n_1, p}	3
kj	{k, n_3, j}	9
hg	{h, n_8, g}	11

segment kn_3. Therefore, the actual overlap segment of the edge kp is the segment n_3j with an influence value equal to 9.

Table 2 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 two markers. The third column summarizes how the edge collapsing technique computes the overlap segment (os) with the maximum influence value (I_s) in each case.

It is important to note that 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 score is a smaller value than I_0. Thus, we can prune any marked edge with $Sc\ (e)<I_o$. For other edges, we update I_0 to the influence value of the actual overlap segment (I_s).

Step 6 (Finding the Maximum Influence Value) (Lines 12–13): When the algorithm terminates, S_0 returns the set of optimal overlap segment(s) with the optimal influence value I_0.

Algorithm 1 represents our implementation of the EONL algorithm:

Algorithm 1 EONL Algorithm

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 $Sc(e)$

6: Initialize S_o and I_o to $\varnothing$

7: For each marked edge e of MET table

8: If $Sc(e) \geq I_o$

9: Apply edge collapsing to edge e

10: Retrieve I_s and optimal overlap segment(s)

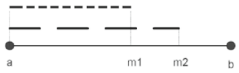
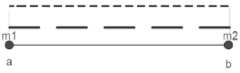
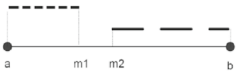
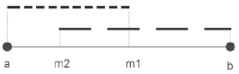
11: Update $I_o=I_s$

12: Update S_o to the set of overlap segments with maximum influence value I_s

13: Return optimal solution set S_o and I_o

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 (Recall that the starting and ending nodes of the edges are considered as markers which are not illustrated using arrows in Fig. 3). We construct the MET and sort its entries based on their $Sc(e)$ values. The first edge in MET is hg. By applying the edge collapsing technique to hg we retrieve the optimal overlap segment $\{o_3n_8\}$ with an influence value I_s equal to 11. Then, we update I_0 to 11. Thereafter, we perform the iterative steps on the remainder of the marked edges. Among 14 marked edges from the road network shown in Fig. 2, only the marked edge ih satisfies the condition $Sc(e) \geq I_0$. Therefore, the remainder of the marked edges can be pruned. By applying edge collapsing on ih, the overlap segment o_3n_5 is derived which leaves I_0 unchanged. At this point the algorithm terminates since all marked edges have been

Table 2 Edge collapsing technique

Case	Overlapping Local Edges	Overlap Segment
1		$os = \mathrm{am}_1$ $I_s = \mathrm{I}(e_1) + \mathrm{I}(e_2)$
2		$os = \mathrm{ab}$ $I_s = \mathrm{I}(e_1) + \mathrm{I}(e_2)$
3		If $(\mathrm{I}(e_1) > \mathrm{I}(e_2))$ $os = \mathrm{a\ m}_1 ; I_s = \mathrm{I}(e_1)$ Else $os = \mathrm{b\ m}_2 ; I_s = \mathrm{I}(e_2)$
4		$os = \mathrm{m}_2\mathrm{m}_1$ $I_s = \mathrm{I}(e_1) + \mathrm{I}(e_2)$

processed. Therefore, the optimal network queries on the dataset shown in Fig. 3 returns overlap segments $\{o_3n_5, o_3n_8\}$ with an optimal influence value of 11.

5 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 [11] 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*. Figure 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 5.1 and 5.2, respectively.

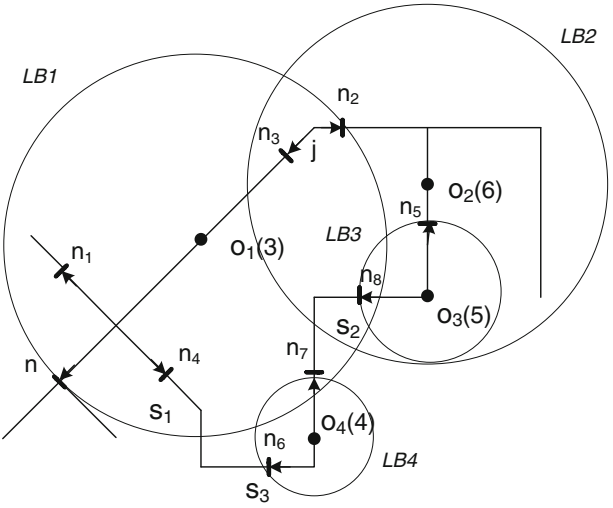


Fig. 5 Local bounds

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 performing ONL queries 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 the MET and edge collapsing technique with EONL). It is important to mention that overlapping spatial bounds help 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 our edge collapsing technique. Below we explain our implementation of BONL in more detail.

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 the overlapping list *OL(lb)* of *lb*. Lemma 1 defines the condition to identify overlapping bounds:

LEMMA 1 Local bound lb_1 with radius r_1 overlaps local bound lb_2 with radius r_2 if and only if $|r_1|+|r_2|\geq|o_1o_2|$ where o_1 and o_2 are centers of the circular bounds lb_1 and lb_2, respectively.

PROOF**.** Proof is obvious.

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*

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

(lb) simply *OL*. The entries of the POT are sorted in descending order of *w(OL),* where $w(OL)= \sum_{lb \in OL} w(lb)$ (Recall *w(lb)* is the weight of local bound *lb*). Table 3 shows the POT constructed for the example depicted in Fig. 5.

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

Step 1 (Expanding Local Networks and Marking the Edges): For each entry (*lb*,*OL*) in the POT, 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 end points on each edge of the network.

Step 2 (Identifying the 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 whose actual influence value has a greater value. This means we can prune some entries from the POT where $w(OL)+w(lb)<I_o$ and I_o is the current optimal influence in the current iteration.

5.1 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 a set of landmarks (a landmark can be any point on the road network). This approach is inspired by the ALT algorithm of Goldberg and Harrelson [7]. 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 [4]. Then, upper bounds are computed in constant time using these distances.

Calculating "From" and "To" Distances: Since each edge of our experimental road network Los Angeles (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).

Calculating Upper

Bound Value: Figure 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*

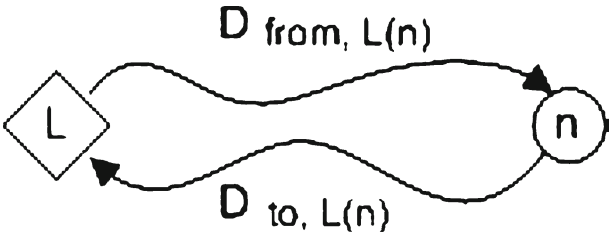


Fig. 6 Distance from and to landmarks

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*, we first 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 weighted 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 LA County road network. With the weighted 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 square 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 7) show that the uniform landmark selection outperforms the weighted 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 ($UdN(o,s)$) 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 with Minimum Bound (BONL-M) method in which we improve the local bounds by computing the actual network distance between objects and their nearest sites.

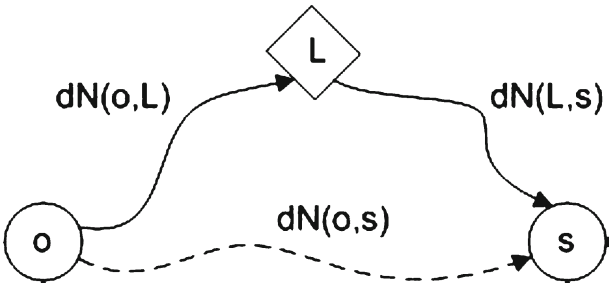


Fig. 7 Upper bound value calculation

5.2 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. Let G (N, E) represent a road network, where *N* is the set of nodes and *E* is the set of edges. We define G′ (N, E′) as the reverse graph of the road network G if for each edge e(a,b)∈ E, there exists a reverse edge e′(b,a) ∈ E′.

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 is greater than the shortest distance 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 calculates 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.

6 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 the 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 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: This step takes $O(|O|^2)$ time since there are |O| local bounds.

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

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

The dominant factors in the overall running time of BONL-U are O (k |N| log |N|+k |E|)+O ($|O|^2$ |N| log |N|+$|O|^3$ |E|)).

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

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

The dominant factors in the overall running time of BONL-M are O (|S| (|N| log |N|+|E|))+O ($|O|^2$ |N| log |N|+$|O|^3$ |E|).

EONL: In this case, we reduce the running time of the optimal network location query by eliminating the cost of the upper bound/minimum upper bound value computation. The costs of constructing the MET and sorting the table are O (|E|) and O (|E| log |E|) respectively. 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. The dominant factors in the overall running time of EONL are O (|O| |N| log |N|+$|O|^2$ |E|).

7 Experimental evaluation

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

7.1 Experimental setup

All experiments were performed on an Intel Core Duo 3GHz, 4 GB of RAM, Dual-Boot Windows 7/Fedora 16 Linux system. The algorithms are implemented in Microsoft C# in .NET platform 3.5. The reason we chose a dual-boot system is the fact that we will later compare our implemented approach with that of Xiao et al. [16]. Their approach was also programmed in C++ on a Linux machine. 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 datasets for objects and sites. Objects

Table 4 Five real datasets for sites

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

are population data derived from LANDSCAN and compiled on a 30″×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. For each experiment, we use a subset of object points selected from this base dataset that we will describe in each of the experiments. We also deployed five datasets consisting of Johnny Rockets restaurants, McDonald's restaurants, hospitals, schools, and all fast food restaurants (i.e. outlets) 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.

7.2 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 queries in L_1 and L_2/Euclidean space are 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 the LA county road network. For site

Fig. 8 Non-overlapping case

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

points, we selected a subset of McDonald's including seven sites (Fig. 8 shows only four of the site points). We applied the L_2 [14] 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. Figure 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.

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.

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

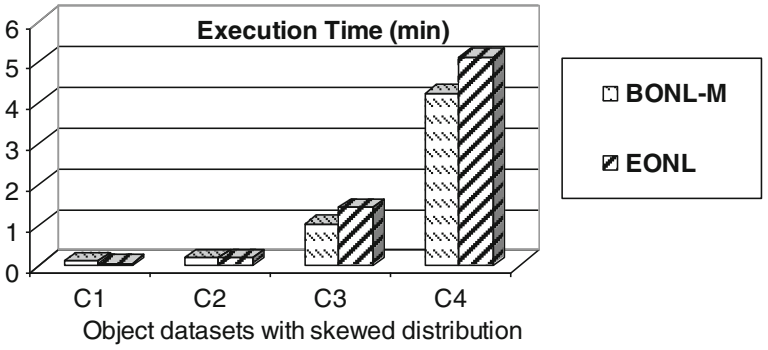


Fig. 9 Execution times of the algorithms with a single site dataset and four skewed object datasets

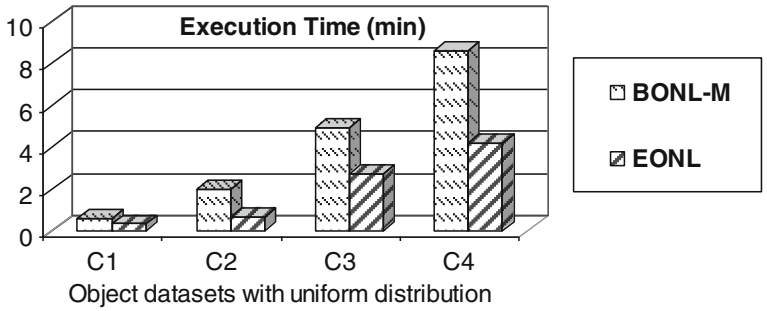


Fig. 10 Execution times of the algorithms with a single site dataset and four uniformly distributed object datasets

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. For the fixed site dataset, we picked the set of Johnny Rockets restaurants which has a small number of site data points compared to the other site datasets in Table 4. Thereafter, we applied the BONL-M and EONL approaches to the aforementioned datasets and computed the execution times (as we show later, BONL-M outperforms BONL-U, hence the latter was excluded from this experiment). Figure 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 relative 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 always outperformed BONL-M (Fig. 10). This is because with a uniform distribution of object points, the number of overlapping local bounds is always high which results in a higher cost of identifying overlap segments relative to the cost of local network expansion.

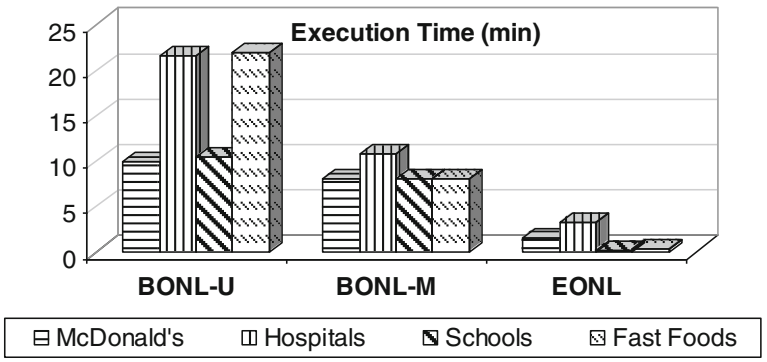


Fig. 11 Execution times of the algorithms with uniformly distributed objects

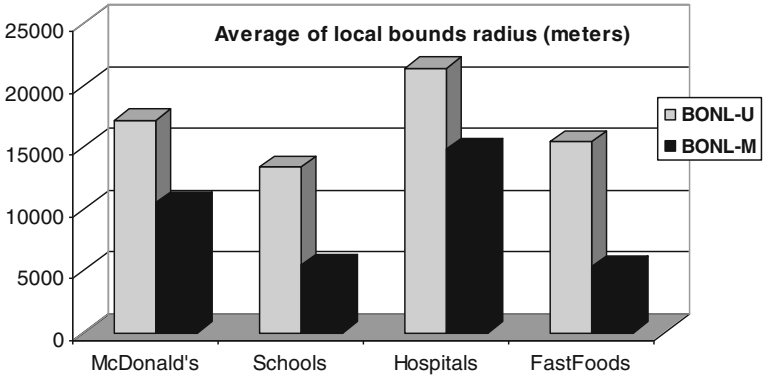


Fig. 12 Average size of local bounds

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

The three algorithms behaved similarly with the small McDonald's, Hospitals, and Schools datasets. The Hospital 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). Also, although the size of fast food restaurants is large, the execution time of EONL is low. This is because the complexity of EONL (O (|E| log |E|)+O (|O| (|N| log |N|+|E|))+O (|E| $|O|^2$)) is independent of the number of site points, |S|.

Magnitude of Local Bounds: The radius of the local bounds was improved 53 % on average by using BONL-M in place of the BONL-U algorithm. Figure 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 Hospitals dataset has the highest average of local bound radii in both algorithms because the skewed site distribution meant that the expansion of the local network traverses a longer path until it hits the nearest site.

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 weighted 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 weighted approach assigned 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.

Comparison with FGP-OTF method ([16]: In this experiment, we compare the performance of optimal location queries introduced in [16] with our proposed approach. From the several techniques presented in [16], we focused on the FGP-OTF method since it was reported as the most efficient approach in terms of execution time. We applied the FGP-OTF[1] algorithm and EONL algorithm to the four sites datasets of Table 4 and we selected the uniformly distributed population data with 9662 points as the

[1] There is a user defined parameter called $\theta \in (0,1]$ used in the FGP-OTF algorithm. For this experiment, we ran the FGP-OTF algorithm with different θ values in the range of [0.0001, 1]. The θ value equal to 0.001 resulted in less computation cost and the corresponding execution time is reported in Fig. 13.

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

	McDonald's	Hospitals	Schools	Fast foods
BONL-U Uniform	10 min	21.5 min	10.5 min	22 min
BONL-U Weighted	> 1 h	> 1 h	10.5 min	31 min

fixed object dataset. Thereafter, we identified the optimal location derived by each approach to compare their accuracy. Both approaches reported the same set of segments as the optimal location. Then, we computed their execution times to compare their performance. Figure 13 shows that the EONL approach outperforms the FGP-OTF approach. Despite the fact that FGP-OTF avoids the exhaustive search on all edges of the network by partitioning the network into sub-graphs and pruning the edges of some sub-graphs, it still shows a significant overhead in computation for these datasets. In this experimental setup, as we mentioned before, the weights of objects are distributed nearly uniformly. We observed while running FGP-OTF the upper-bound values of the weight of the sub-graphs returns relatively similar values in graph partitioning process. This feature produces fewer numbers of pruned sub-graphs and edges, respectively. As a result, in a large road network like the one used in this experiment (recall |E|=871715 and |N|=375691), the FGP-OTG algorithm resulted in a high computation cost (it took hours for some datasets) (Fig. 13).

It is important to note that according to [16], FGP-OTF can find the optimal location in minutes for road networks with sizes of similar order (e.g. |E|=223000 and |N|=174955). However, in that case unlike our experimental data set the weight distribution of the objects is skewed (not uniform). As a result the pruning algorithm of FGP-OTF can effectively filter out a large number of sub-graphs and their corresponding edges and efficiently identify the optimal location.

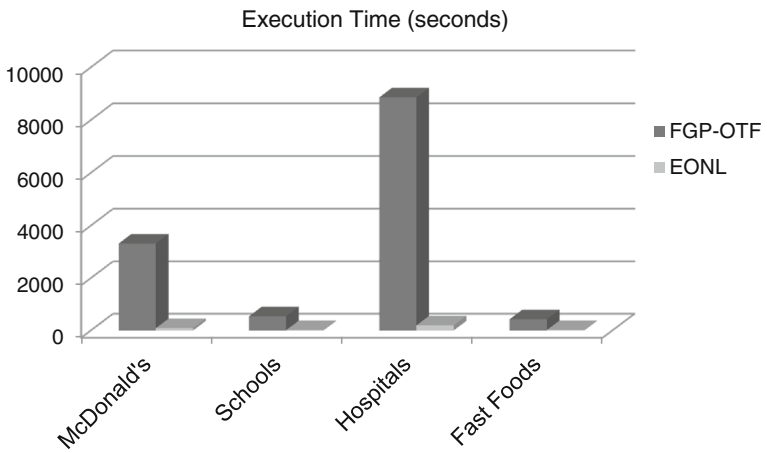


Fig. 13 Comparing the execution time of the EONL algorithm with the FGP-OTF algorithm [16]

8 Conclusions and future directions

In this study, we proposed a set of scalable solutions for the problem of optimal location for objects and sites located on spatial networks. Accordingly, we proposed EONL and BONL as two complementary approaches for the 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 using real-world data.

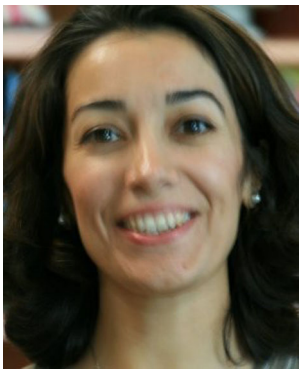
We intend to extend this study in two ways. First, with the optimal network location problem, like all previous work we assumed a site covers an object if and only if the site is the closest site to the object. We plan to study optimal network location queries under a more generalized definition of coverage where a site covers an object based on a combination of mutual relationships (not only proximity), such as when site properties (e.g., hotel amenities) match 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.

Acknowledgments The authors would like to thank Professor FeiFei Li for making the source code and the corresponding datasets used in [16] accessible.

References

1. Berman O, Krass D (2002) The generalized maximal covering location problems. Comput Oper Res 29(6):563–581
2. Church RL (1984) The planar maximal covering location problem. J Reg Sci 24(1984):185–201
3. Church RL, Revelle C (1974) The maximal covering location problem. Pap Reg Sci Assoc 32(1974):101–118
4. Dijkstra EW (1959) A note on two problems in connection with graphs. Numeriche Math 1(1):269–271
5. Du Y, Zhang D, Xia T (2005) The optimal-location query. SSTD 2005:163–180
6. Ghaemi P, Shahabi K, Wilson JP, Banaei-Kashani F (2010) Optimal network location queries. Proceedings of the 18th SIGSPATIAL international conference on advances in geographic. Inf Syst 2010:478–481
7. Goldberg AV, Harrelson C (2005) Computing the shortest path: a* search meets graph theory. ACM-SIAM 2005:156–165
8. Korn F, Muthukrishnan S (2000) Influence sets based on reverse nearest neighbor queries. SIGMOD 29(2):201–212
9. Mehrez A, Stulman A (1982) The maximal covering location problem with facility placement on the entire plane. J Reg Sci 22(1982):361–365
10. Murray AT, Tong D (2007) Coverage optimization in continuous space facility siting. Int J Geogr Inf Sci 21(7):757–776
11. Papadias D, Zhang J, Mamoulis N, Tao Y (2003) Query processing in spatial network databases. VLDB 2003:802–813
12. Stanoi I, Riedwald M, El Abbadi A (2001) Discovery of influence sets in frequently updated databases. VLDB 2001:99–108
13. Toregas C, Swain R, Revelle C (1971) Bergman L (1971) The location of emergency service facilities. Oper Res 19(6):1363–1373
14. Wong RC, Ozsu MT, Yu PS, Fu AW, Liu L (2009) Efficient method for maximizing bichromatic reverse nearest neighbor. VLDB 2009:1126–1149

15. Xia T, Zhang D, Kanoulas E, Du Y (2005) On computing top-t most influential spatial sites. VLDB 2005:946–957
16. Xiao X, Yao B, Li F (2011) Optimal location queries in road network databases. Proceedings 27th ICDE Conference, 2011
17. Yang C, Lin KI (2001) An index structure for efficient reverse nearest neighbor queries. ICDE 2001:51–60



Parisa Ghaemi received her B.S. in Computer Engineering from Amirkabir University of Technology in 1996, her M.S. in Computer Engineering from Sharif University of Technology in 1998, and her PhD in Computer Science from University of Southern California in 2012. She currently focuses her research on Spatial Databases and Location Planning. She can be contacted at ghaemi@usc.edu.

Kaveh Shahabi is a doctoral student in the Department of Computer Science at USC and graduate research assistant in the USC Spatial Science Institute. He received his B.S. in Physics from Iran University of Science and Technology. His research focus is on Scalable Dynamic Evacuation Routing. He developed many GIS-based software in the Spatial Science Institute and is planning to extend his knowledge to mobile and web development as well. He can be reached at kshahabi@usc.edu.

Dr. John P. Wilson is Professor of Spatial Sciences and Sociology at the University of Southern California (USC) where he directs the Spatial Sciences Institute as well as the Geographic Information Science & Technology Graduate Programs and GIS Research Laboratory, and also holds adjunct appointments as Professor in the School of Architecture and in the Viterbi School of Engineering's Departments of Computer Science and Civil & Environmental Engineering. He founded the journal Transactions in GIS in 1996 and has served as Editorin-Chief since its inception. His research is focused on the modeling of environmental systems and makes extensive use of GIS software tools, fieldwork, spatial analysis techniques, and computer models. Much of this work is collaborative and multi-disciplinary in character with the general goal of improving our knowledge andunderstanding of human well-being in both the natural and built environments.

Farnoush Banaei-Kashani is a Research Associate and Associate Director of Research at NSF's Integrated Media Systems Center (IMSC) at the University of Southern California. He received his B.S. in Computer Engineering from Sharif University of Technology in 1996, and his M.S. and Ph.D. Degrees in Computer Networks and Computer Science from the University of Southern California in 2002 and 2006, respectively. He has authored more than forty research articles in the areas of Sensor and Peer-to-Peer Databases, Distributed Databases, and Spatial Databases, Data Streams, Cloud Computing, and Social Network Analysis. Dr. Banaei-Kashani regularly serves on the program committee of various database conferences.