Optimal Location Queries in Road Network Databases

Xiaokui Xiao[1], Bin Yao,[2] Feifei Li[2]
[1]*School of Computer Engineering, Nanyang Technological University, Singapore*
[2]*Department of Computer Science, Florida State University, Tallahassee, FL, USA*
[1]xkxiao@ntu.edu.sg, [2]{yao,lifeifei}@cs.fsu.edu

***Abstract*— Optimal location (OL) queries are a type of spatial queries particularly useful for the strategic planning of resources. Given a set of existing facilities and a set of clients, an OL query asks for a location to build a new facility that optimizes a certain cost metric (defined based on the distances between the clients and the facilities). Several techniques have been proposed to address OL queries, assuming that *all clients and facilities reside in an L_p space*. In practice, however, movements between spatial locations are usually confined by the underlying road network, and hence, the actual distance between two locations can differ significantly from their L_p distance.**

Motivated by the deficiency of the existing techniques, this paper presents the first study on OL queries in road networks. We propose a unified framework that addresses three variants of OL queries that find important applications in practice, and we instantiate the framework with several novel query processing algorithms. We demonstrate the efficiency of our solutions through extensive experiments with real data.

I. Introduction

An *optimal location (OL) query* concerns three spatial point sets: a set F of *facilities*, a set C of *clients*, and a set P of *candidate locations*. The objective of this query is to identify a candidate location $p \in P$, such that a new facility built at p can optimize a certain cost metric that is defined based on the distances between the facilities and the clients. OL queries find important applications in the strategic planning of resources (e.g., hospitals, post offices, banks, retail facilities) in both public and private sectors [1]–[3]. As an example, we illustrate three OL queries based on different cost metrics.

Example 1: Julie would like to open a new supermarket in Gotham city that can attract as many customers as possible. Given the set F (C) of all existing supermarkets (residential locations) in the city, Julie may look for a candidate location p, such that a new supermarket on p would be the closest supermarket for the largest number of residential locations. ■

Example 2: John owns a set F of pizza shops that deliver to a set C of places in Gotham city. When John wants to add another pizza shop, a natural choice for him is a candidate location that minimizes the average distance from the points in C to their respective nearest pizza shops. ■

Example 3: Gotham city government plans to establish a new fire station. Given the set F (C) of existing fire stations (buildings), the government may seek a candidate location that minimizes the maximum distance from any building to its nearest fire station. ■

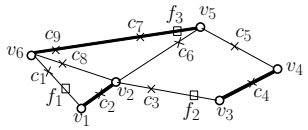


Fig. 1. Example of G^o

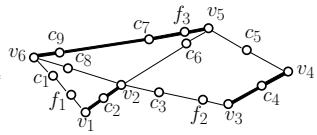


Fig. 2. Example of G

Several techniques [1]–[4] have been proposed for processing OL queries under various cost metrics. All those techniques, however, assume that F and C are point sets in an L_p space. This assumption is rather restrictive because, in practice, movements between spatial locations are usually confined by the underlying road network, and hence, the commute distance between two locations can differ significantly from their L_p distance. Consequently, the existing solutions for OL queries cannot provide useful results for practical applications in road networks.

Problem Formulation. This paper presents a novel and comprehensive study on OL queries in road network databases. We consider a problem setting as follows. First, any facility in F or any client in C should locate on an edge in an undirected connected graph $G^\circ = (V^\circ, E^\circ)$, where V° (E°) denotes the set of vertices (edges) in G°. Second, every client $c \in C$ is associated with a positive weight $w(c)$ that captures the importance of the client. For example, if each client point c represents a residential location, then $w(c)$ may be specified as the size of the population residing at c. Third, there should exist a user-specified set E_c° of edges in E°, such that a new facility f can be built on any point on any edge in E_c°, as long as f does not overlap with an existing facility in F. E_c° can be arbitrary, e.g., we can have $E_c^\circ = E^\circ$. We define P as the set of points on the edges in E_c° that are not in F, and we refer to any point in P as a *candidate location*. For example, Figure 1 illustrates a road network that consists of 6 vertices and 8 edges. The squares (crosses) in the Figure denote the facilities (clients) in the road network. The highlighted edges are the user-specified set E_c° of edges where a new facility may be built.

We investigate three variants of OL queries as follows:
1) The *competitive location query* asks for a candidate location $p \in P$ that maximizes the total weight of the clients *attracted* by a new facility built on p. Specifically, we say that a client c is *attracted* by a facility f, and that f is an *attractor* for c, if the network distance $d(c, f)$ between c and f is at most the distance between c and any facility in F. In other words, the

competitive location query ensures that

$$p = \operatorname*{argmax}_{p \in P} \sum_{c \in C_p} w(c), \quad (1)$$

where $C_p = \{c \mid c \in C \wedge \forall f \in F, d(c,p) \leq d(c,f)\}$, i.e., C_p is the set of clients attracted by p. Example 1 demonstrates an instance of this query.

2) The *MinSum location query* asks for a candidate location $p \in P$ on which a new facility can be built to minimize the total *weighted attractor distance* (WAD) of the clients. In particular, the WAD of a client c is defined as $\widehat{a}(c) = w(c) \cdot a(c)$, where $a(c)$ denotes the distance from c to its attractor (referred as the *attractor distance* of c). That is, the MinSum location query requires that

$$\begin{aligned} p &= \operatorname*{argmin}_{p \in P} \sum_{c \in C} w(c) \cdot \min\Big\{d(c,f) \mid f \in F \cup \{p\}\Big\} \\ &= \operatorname*{argmin}_{p \in P} \sum_{c \in C} \widehat{a}(c). \end{aligned} \quad (2)$$

Example 2 shows a special case of the MinSum location query where all clients have the same weight.

3) The *MinMax location query* asks for a candidate location $p \in P$ to construct a new facility that minimizes the maximum WAD of the clients, i.e.,

$$p = \operatorname*{argmin}_{p \in P} \big(\max_{c \in C} \Big\{\widehat{a}(c) \mid F = F \cup \{p\}\Big\}\big). \quad (3)$$

Example 3 illustrates a MinMax location query.

One fundamental challenge in answering an OL query is that there exists an *infinite* number of candidate locations in P where the new facility may be built. (Recall that P contains *all* points on the edges in the user-specified set E_c°, except the points where existing facilities are located.) This necessitates query processing techniques that can identify query results without enumerating all candidate locations. Another complicating issue is that the answer to an OL may not be unique, i.e., there may exist multiple candidate locations in P that satisfy Equation 1, 2, or 3. We propose to identify *all* answers for any given optimal location query, and return them to the user for final selection. This renders the problem even more challenging, since it requires additional efforts to ensure the completeness of the query results.

Contributions. In this paper, we propose a unified solution that addresses all aforementioned variants of optimal location queries in road network databases. Our first contribution is a solution framework based on the divide-and-conquer paradigm. In this framework, we process a query by first (i) dividing the edges in G° into smaller intervals, then (ii) computing the best query answers on each interval, and finally (iii) combining the answers from individual intervals to derive the global optimal locations. A distinct feature of this framework is that most of its algorithmic components are *generic*, i.e., they are not specific to any of the three types of OL queries. This significantly simplifies the design of query processing algorithms, and enables us to develop general optimization techniques that work for all three query types.

Second, we instantiate the proposed framework with a set of novel algorithms that optimize query efficiency by exploiting the characteristics of OL queries. We provide theoretical analysis on the performance of each algorithm in terms of time complexity and space consumption.

Third, we demonstrate the efficiency of our algorithms with extensive experiments on large-scale real datasets. In particular, in a road network with $174,955$ vertices and $500,000$ clients, our algorithms can answer an OL query in less than 200 seconds on a commodity machine.

II. Related Work

The problem of locating "preferred" facilities with respect to a given set of client points, referred to as the *facility location problem*, has been extensively studied in past years (see [5], [6] for surveys). In its most common form, the problem (i) involves a finite set C of clients and a finite set P of candidate facilities, and (ii) asks for a subset of k ($k > 0$) facilities in P that optimizes a predefined metric. The problem is polynomial-time solvable when k is a constant, but is NP-hard for general k [5], [6]. Existing work on the problem mainly focuses on developing approximate solutions.

OL queries can be regarded as variations of the facility location problem with three modified assumptions: (i) P is an infinite set, (ii) $k = 1$, i.e., only *one* location in P is to be selected (but all locations that tie with each other need to be returned), and (iii) a finite set F of facilities has been constructed in advance. These modified assumptions distinguish OL queries from the facility location problem.

Previous work [1]–[4] on OL queries considers the case when the transportation cost between a facility and a client is decided by their L_p distance. Specifically, Cabello et al. [3] and Wong et al. [4] investigate competitive location queries in the L_2 space. Du et al. [2] and Zhang et al. [1] focus on the L_1 space, and propose solutions for competitive and MinSum location queries, respectively. None of the solutions developed therein is applicable when the facilities and clients reside in a road network.

There also exist two other variations of the facility location problem, namely, the *single facility location* problem [5], [6] and the *online facility location* problem [7], [8], that are related to (but different from) OL queries. The single facility location problem asks for *one* location in P that optimizes a predefined metric with respect to a given set C of clients. It requires that no facility has been built previously, whereas OL queries consider the existence of a set F of facilities.

The online facility location problem assumes a dynamic setting where (i) the set C of clients is initially empty, and (ii) new clients may be inserted into C as time evolves. It asks for a solution that constructs facilities incrementally (i.e., one at a time), such that the quality of the solution (with respect to some predefined metric) is competitive against any solutions that are given all client points in advance. This problem is similar to OL queries, in the sense that they all aim to optimize the locations of new facilities based on the existing facilities and clients. However, the techniques [7], [8] for the online

facility location problem cannot address OL queries, since those techniques assume that the set P of candidate facility locations is finite; in contrast, OL queries assume that P contains an infinite number of points, e.g., P may consist of all points (i) in an L_p space (as in [1]–[4]) or (ii) on a set of edges in a road network (as in our setting).

Lastly, there is a large body of literature on query processing techniques for road network databases [9]–[18]. Most of those techniques are designed for the *nearest neighbor* (NN) query [9], [10], [16] or its variants, e.g., *approximate NN queries* [12], [13], *aggregate NN queries* [14], *continuous NN queries* [15], *path NN queries* [17], etc. None of those techniques can address the problem we consider, due to the fundamental differences between NN queries and OL queries. Such differences are also demonstrated by the fact that, despite the plethora of solutions for L_p-space NN queries, considerable research effort [1]–[4] is still devoted to OL queries in L_p spaces.

III. Solution Overview

We propose one unified framework for the three variants of OL queries. In a nutshell, our solution adopts a divide-and-conquer paradigm as follows. First, we divide the edges in E° into smaller intervals, such that all facilities and clients fall on only the endpoints (but not the interior) of the intervals. As a second step, we collect the intervals that are segments of some edges in E_c°, i.e, all points in such an interval are candidate locations in P. Then, we traverse those intervals in a certain order. For each interval I examined, we compute the *local* optimal locations on I, i.e., the points on I that provide a better solution to the OL query than any other points on I. The *global* optimal locations are pinpointed and returned, once we confirm that none of the unvisited intervals can provide a better solution than the best local optima found so far.

In the following, we will introduce the basic idea of each step in our framework; the details of our algorithms will be presented in Sections IV-VI. For convenience, we define n as the maximum number of elements in V°, E°, C, and F, i.e., $n = \max\{|V^\circ|, |E^\circ|, |C|, |F|\}$. Table I summarizes the notations frequently used in the paper.

Construction of Road Intervals. We divide the edges in E° into intervals, by inserting all facilities and clients into the road network $G^\circ = (V^\circ, E^\circ)$. Specifically, for each point $\rho \in C \cup F$, we first identify the edge $e \in E^\circ$ on which ρ locates. Let v_l and v_r be the two vertices connected by e. We then break e into two road segments, one from v_l to ρ and the other from ρ to v_r. As such, ρ becomes a vertex in the network. Once all facilities and clients have been inserted into G°, we obtain a new road network $G = (V, E)$ where $V = V^\circ \cup C \cup F$. For example, Figure 2 illustrates a road network transformed from the one in Figure 1. Transforming G° to G requires only $O(n)$ space and $O(n)$ time, since $|C| = O(n)$, $|F| = O(n)$, and it takes only $O(1)$ time to add a vertex in G°. In the sequel, we simply refer to G as our road network.

Traversal of Road Intervals. After G is constructed, we collect the set E_c of edges in E that are partial segments of some edges in E_c°. For example, the highlighted edges in Figure 2 illustrate the set E_c that correspond to the set E_c° of highlighted edges in Figure 1. As a next step, we traverse E_c to look for the optimal locations. A straightforward approach is to process the edges in E_c in a random order, which, however, incurs significant overhead, since the optimal locations cannot be identified until all edges in E_c are inspected. Section VI addresses this issue with novel techniques that avoid the exhaustive search on E_c. The idea is to first divide E_c into subsets, and then process the subsets in descending order of their likelihood of containing the optimal locations.

TABLE I
Frequently Used Notations

Symbol	Description
$G^\circ = (V^\circ, E^\circ)$	the road network with vertex (edge) set V° (E°)
C	the set of clients
F	the set of existing facilities
E_c°	the user-specified set of edges on which the new facility can be built
P	the set of candidate locations
$d(p_1, p_2)$	the network distance between points p_1 and p_2
$w(c)$	the weight of a client c
$a(c)$	the attractor distance of a client c
$\widehat{a}(c)$	the weighted attractor distance of a client c
C_p	the set of clients attracted by a point p
n	$n = \max\{\|V^\circ\|, \|E^\circ\|, \|C\|, \|F\|\}$
$G = (V, E)$	the road network transformed from G° (see Section III)
E_c	the set of edges in E that are segments of the edges in E_c° (see Section III)
$\mathcal{A}(v)$	the *attraction set* of a vertex v in G (see Section III)
$m(p)$	the *merit* of a point p (see Section IV-B)

Identification of Local Optimal Locations. In Section IV, we will present algorithms for computing the local optimal locations on any edge $e \in E_c$, based on (i) the attractor distance of each client, and (ii) the *attraction set* $\mathcal{A}(v)$ of each endpoint v of e. Specifically, the attraction set $\mathcal{A}(v)$ contains entries of the form $\langle c, d(c, v)\rangle$, for any client c such that $d(c, v) \leq a(c)$. That is, $\mathcal{A}(v)$ records the clients that are closer to v than to their respective attractors (i.e., the respective nearest facilities). The attraction sets of e's endpoints are crucial to our algorithm, since they capture all clients that might be affected by a new facility built on e (see Section IV for a detailed discussion). We will present our algorithms for computing attraction sets and attractor distances in Section V.

IV. Local Optimal Locations

This section presents our algorithms for computing local optimal locations on any edge $e \in E_c$, given the attraction sets of e's endpoints, and the attractor distances of the clients. For ease of exposition, we will elaborate our algorithms under the assumption that none of e's endpoints is an existing facility in F, i.e., both endpoints of e are candidate locations in P. We will discuss how our algorithms can be extended (for the general case) in the end of the discussion for each query type.

Algorithm *CompLoc* (e)

1. construct an empty one-dimensional plane R
2. let ℓ be the length of e, and v_l (v_r) be the left (right) endpoint of e
3. for each client c that appears in $\mathcal{A}(v_l)$ but not $\mathcal{A}(v_r)$
4. create in R a line segment $[0, a(c) - d(c, v_l)]$
5. assign a weight $w(c)$ to the segment
6. for each client c that appears in $\mathcal{A}(v_r)$ but not $\mathcal{A}(v_l)$
7. create in R a segment $[\ell - a(c) + d(c, v_r), \ell]$ with a weight $w(c)$
8. for each client c that appears in both $\mathcal{A}(v_l)$ and $\mathcal{A}(v_r)$
9. if $\ell \leq 2 \cdot a(c) - d(c, v_l) - d(c, v_r)$
10. create in R a line segment $[0, \ell]$ with a weight $w(c)$
11. else
12. create in R two line segments $[0, a(c) - d(c, v_l)]$ and $[\ell - a(c) + d(c, v_r), \ell]$, each with a weight $w(c)$
13. compute the intervals $I \subseteq [0, \ell]$, such that I maximizes the total weights of the line segments in R that fully cover I
14. return the intervals identified at Line 13

Fig. 3. The *CompLoc* Algorithm

A. Competitive Location Queries

Recall that a competitive location query asks for a new facility that maximizes the total weight of the clients attracted by it. Intuitively, to decide the optimal locations for such a new facility on a given edge $e \in E_c$, it suffices to identify the set of clients that can be attracted by each point p on e. As shown in the following lemma, the clients attracted by any p can be easily computed from the attraction sets of e's endpoints.

Lemma 1: A client c is attracted by a point p on an edge $e \in E_c$, iff there exists an entry $\langle c, d(c, v)\rangle$ in the attraction set of an endpoint v of e, such that $d(c, v) + d(v, p) \leq a(c)$.

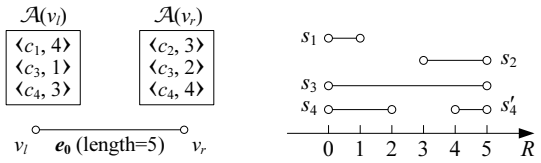
Proof: Observe that $d(c, p) \leq d(c, v) + d(v, p)$. Hence, when $d(c, v) + d(v, p) \leq a(c)$, we have $d(c, p) \leq a(c)$, i.e., c is attracted by p. Thus, the "if" direction of the lemma holds.

Now consider the "only if" direction. Since p is a point on e, the shortest path from p to c must go through an endpoint v of e. Observe that $d(p, c) \geq d(v, c)$. Therefore, if c is attracted by p, we have $a(c) \geq d(p, c) \geq d(v, c)$, which indicates that $\langle c, d(c, v)\rangle$ must be an entry in $\mathcal{A}(v)$. ■

Based on Lemma 1, we propose the *CompLoc* algorithm (in Figure 3) for finding local competitive locations on an edge $e \in E_c$. We illustrate the algorithm with an example.

Example 4: Suppose that we apply *CompLoc* on an edge e_0 with a length $\ell = 5$. Figure 4(a) illustrates $\mathcal{A}(v_l)$ and $\mathcal{A}(v_r)$, where v_l (v_r) is the left (right) endpoint of e_0. Assume that each client c has a weight $w(c) = 1$ and an attractor distance $a(c) = 5$.

CompLoc starts by creating a one-dimensional plane R. After that, it identifies those clients that appear in $\mathcal{A}(v_l)$ but not $\mathcal{A}(v_r)$. By Lemma 1, for any c of those clients, if c is attracted a point p on e_0, then $d(p, v_l) \in [0, a(c) - d(c, v_l)]$, and vice versa. To capture this fact, *CompLoc* creates in R a line segment $[0, a(c) - d(c, v_l)]$, and assigns a weight $w(c) = 1$ to the segment. In our example, c_1 is the only client that appears in $\mathcal{A}(v_l)$ but not $\mathcal{A}(v_r)$, and $a(c_1) - d(c_1, v_l) = 1$. Hence,



(a) Edge e_0 (b) Plane R

Fig. 4. Demonstration of *CompLoc*

CompLoc adds in R a segment $s_1 = [0, 1]$ with a weight $w(c_1) = 1$, as illustrated in Figure 4(b).

Next, *CompLoc* examines the only client c_2 that is contained in $\mathcal{A}(v_r)$ but not $\mathcal{A}(v_l)$. By Lemma 1, a point $p \in e_0$ is an attractor for c, if and only if $d(p, v_l) \in [\ell - a(c_2) + d(c_2, v_r), \ell]$. Accordingly, *CompLoc* inserts in R a segment $s_2 = [\ell - a(c_2) + d(c_2, v_r), \ell]$ with a weight $w(c_2) = 1$.

After that, *CompLoc* identifies the clients c_3 and c_4 that appear in both $\mathcal{A}(v_l)$ and $\mathcal{A}(v_r)$. For c_3, we have $\ell \leq 2 \cdot a(c_3) - d(c_3, v_l) - d(c_3, v_r)$, which (by Lemma 1) indicates that any point on e_0 can attract c_3. Hence, *CompLoc* creates in R a segment $[0, 5]$ with a weight $w(c_3) = 1$. On the other hand, since $\ell > 2 \cdot a(c_4) - d(c_4, v_l) - d(c_4, v_r)$, a point p on e_0 can attract c_4, if and only if $d(p, v_l) \in [0, a(c_4) - d(c_4, v_l)]$ or $d(p, v_l) \in [\ell - a(c_4) + d(c_4, v_r), \ell]$. Therefore, *CompLoc* inserts in R two segments $s_4 = [0, 2]$ and $s'_4 = [4, 5]$, each with a weight 1 (see Figure 4(b)).

As a next step, *CompLoc* scans through the line segments in R to compute the local competitive locations on e_0. Let p be any point on e_0, and o be the point in R whose coordinate equals the distance from p to v_l. Observe that, a client $c \in C$ is attracted by p, if and only if there exists a segment s in R, such that (i) s is constructed from c and (ii) s covers o. Therefore, to identify the local competitive locations on e_0, it suffices to derive the intervals I in R, such that (i) $I \subseteq [0, \ell]$, and (ii) I maximizes the total weight of the line segments that fully cover I. Such intervals can be computed by applying a standard *plane sweep* algorithm [19] on the line segments in R. In our example, the local competitive locations on e_0 correspond to two intervals in R, namely, $[0, 1]$ and $[4, 5]$, each of which is covered by three segments with a total weight 3. Finally, *CompLoc* terminates by returning the two intervals $[0, 1]$ and $[4, 5]$, as well as the weight 3. ■

Our discussion so far assumes that no facility in F locates on an endpoint of the given edge e. Nevertheless, *CompLoc* can be easily extended for the case when either of e's endpoints is a facility. The only modification required is that, we need to exclude the facility endpoint(s) of e, when we construct the line segment(s) on R that corresponds to each client. For example, if we have a line segment $[0, 5]$ and the left endpoint of e is a facility, then we should modify segment as $(0, 5]$ before we compute the local competitive locations on e. The case when the right endpoint of e is a facility can be handled in a similar manner.

CompLoc runs in $O(n \log n)$ time and $O(n)$ space. First, constructing line segments in R takes $O(n)$ time and $O(n)$ space, since (i) there exist $O(n)$ clients in the attraction sets of the endpoints of e, (ii) at most two segments are created from

each client. Second, since there are only $O(n)$ line segments in R, the plane sweep algorithm on the segments runs in $O(n \log n)$ time and $O(n)$ space.

B. MinSum Location Queries

For any candidate location p, we define the *merit* of p (denoted as $m(p)$) as

$$m(p) = \sum_{c \in C} w(c) \cdot \max\{0, a(c) - d(c,p)\}.$$

That is, $m(p)$ captures how much the total WAD of all clients may reduce, if a new facility is built on p. A point is a local MinSum location on an edge $e \in E_c$, if and only if it has the maximum merit among all points on e. Interestingly, the merit of the points on any edge e is always maximized at one endpoint of e, as shown in the following lemma.

Lemma 2: For any point p in the interior of an edge $e \in E$, if $m(p)$ is larger than the merit of one endpoint of e, then $m(p)$ must be smaller than the merit of the other endpoint.

Proof: Let v_l (v_r) be the left (right) endpoint of e. Recall that C_p is the set of clients attracted by p. First of all,

$$\begin{aligned} m(v_l) &= \sum_{c \in C} w(c) \cdot \max\{0, a(c) - d(c, v_l)\} \\ &\geq \sum_{c \in C_p} w(c) \cdot \big(a(c) - d(c, v_l)\big), \text{ and similarly,} \\ m(v_r) &\geq \sum_{c \in C_p} w(c) \cdot \big(a(c) - d(c, v_r)\big). \end{aligned} \tag{4}$$

Assume w.l.o.g. that $m(p) > m(v_l)$. We have

$$\begin{aligned} m(p) &= \sum_{c \in C_p} w(c) \cdot \big(a(c) - d(c,p)\big) \\ &\geq m(v_l) \geq \sum_{c \in C_p} w(c) \cdot \big(a(c) - d(c, v_l)\big), \end{aligned}$$

which leads to

$$\sum_{c \in C_p} w(c)(d(c, v_l) - d(c,p)) > 0. \tag{5}$$

Let C_p^l (C_p^r) be the subset of clients c in C_p, such that the shortest path from c to p passes through v_l (v_r). Clearly, $C_p^r = C_p - C_p^l$, and $d(c,p) = d(c, v_l) + d(v_l, p)$ for any $c \in C_p^l$. By Equation 5,

$$\begin{aligned} &\sum_{c \in C_p^r} w(c) \cdot \big(d(c, v_l) - d(c,p)\big) \\ &> \sum_{c \in C_p^l} w(c) \cdot \big(d(c,p) - d(c, v_l)\big) = d(v_l, p) \cdot \sum_{c \in C_p^l} w(c). \end{aligned} \tag{6}$$

Since $d(c, v_l) \leq d(c,p) + d(v_l, p)$ for any $c \in C_p^r$, we have

$$d(v_l, p) \cdot \sum_{c \in C_p^r} w(c) \geq \text{LHS of (6)} \geq d(v_l, p) \cdot \sum_{c \in C_p^l} w(c),$$

which means that

$$\sum_{c \in C_p^r} w(c) \quad > \quad \sum_{c \in C_p^l} w(c). \tag{7}$$

Note that $d(c,p) = d(c, v_r) + d(v_r, p)$ for any $c \in C_p^r$, and $d(c, v_r) \leq d(c,p) + d(v_r, p)$ for any $c \in C_p^l$. By Eqn. 4 & 5,

$$\begin{aligned} m(v_r) - m(p) &\geq -m(p) + \sum_{c \in C_p} w(c) \cdot (a(c) - d(c, v_r)) \\ &= \sum_{c \in C_p^l} w(c) \cdot \big(d(c,p) - d(c, v_r)\big) + \sum_{c \in C_p^r} w(c) \cdot \big(d(c,p) - d(c, v_r)\big) \\ &\geq d(v_r, p) \cdot \Big(- \textstyle\sum_{c \in C_p^l} w(c) + \sum_{c \in C_p^r} w(c) \Big) \end{aligned} \tag{8}$$

By Equations 7 and 8, $m(v_r) - m(p) \geq 0$. Hence, the lemma is proved. ■

By Lemma 2, if the endpoints of an edge $e \in E_c$ have different merits, then the endpoint with the larger merit should be the only local MinSum location on e. But what if the merits of the endpoints are identical? The following lemma provides the answer.

Lemma 3: Let e be an edge in E with endpoints v_l, v_r, such that $m(v_l) = m(v_r)$. Then, either all points on e have the same merit, or v_l and v_r have larger merit than any other points on e.

Proof: First of all, by Lemma 2, for any point ρ on e, it must satisfy $m(\rho) \leq m(v_l) = m(v_r)$, given that $m(v_l) = m(v_r)$. Now, assume on the contrary that there exist two points p and q on e, such that $m(v_l) = m(v_r) = m(p) \neq m(q)$. This indicates that $m(q) < m(v_l) = m(v_r) = m(p)$. Assume without loss of generality that $d(v_l, p) < d(v_l, q)$. We will prove the lemma by showing that $m(p) = m(v_l)$ cannot hold given $m(p) > m(q)$.

Let C_p be the set of clients attracted by p. We divide C_p into three subsets C_1, C_2, and C_3, such that

$$\begin{aligned} C_1 &= \{c \in C_p \mid d(c,p) = d(c,q) - d(p,q)\}, \\ C_2 &= \{c \in C_p \mid d(c,p) = d(c,q) + d(p,q)\}, \\ C_3 &= C_p - C_1 - C_2. \end{aligned}$$

It can be verified that, for any client $c \in C_3$, the shortest path from c to p must go through v_l. This indicates that,

$$d(c, v_l) = d(c,p) - d(v_l, p), \forall c \in C_3. \tag{9}$$

Given $m(p) > m(q)$ and $C_p \subseteq C$, we have

$$\begin{aligned} &\sum_{c \in C_1} w(c) \cdot \big(d(c,q) - d(c,p)\big) - \sum_{c \in C_2} w(c) \big(d(c,p) - d(c,q)\big) \\ &\quad + \sum_{c \in C_3} w(c) \cdot |d(c,q) - d(c,p)| \quad > \quad 0. \end{aligned}$$

This leads to

$$\sum_{c \in C_1 \cup C_3} w(c) - \sum_{c \in C_2} w(c) > 0 \tag{10}$$

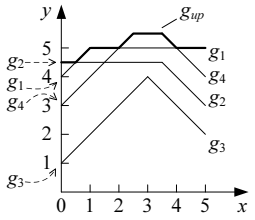
On the other hand, we have

$$\begin{aligned} m(v_l) - m(p) &\geq \sum_{c \in C_1 \cup C_3} w(c) \cdot \big(d(c,p) - d(c, v_l)\big) \\ &\quad - \sum_{c \in C_2} w(c) \cdot \big(d(c, v_l - d(c,p)\big) \\ &= d(v_l, p) \cdot \Big(\sum_{c \in C_1 \cup C_3} w(c) - \sum_{c \in C_2} w(c) \Big) \\ &> 0. \qquad \text{(By Equation 10)} \end{aligned} \tag{11}$$

Thus, the lemma is proved. ■

$a(c_1) = 5$
$a(c_2) = 4.5$
$a(c_3) = 6$
$a(c_4) = 5.5$

(a) Attractor Distances Without the New Facility



(b) Piecewise Linear Functions

Fig. 5. Representing Attractor Distances as Functions of the Location of the New Facility

By Lemmas 2 and 3, we can identify the local MinSum locations on any given edge e as follows. First, we compute the merits of e's endpoints based on their attraction sets. If the merits of the endpoints differ, then we return the endpoint with the larger merit as the answer. Otherwise (i.e., both endpoints of e have the same merit γ), we inspect any point p in the interior of e, and derive $m(p)$ using the attraction sets of the endpoints. If $m(p) < \gamma$, both endpoints of e are returned as the result; otherwise, we must have $m(p) = \gamma$, in which case we return the whole edge e as the answer. In summary, the local MinSum locations on e can be found by computing the merits of at most three points on e, which takes $O(n)$ time and $O(n)$ space given the attraction sets of e's endpoints.

Note that the above algorithm assumes that both endpoints of e are candidate locations. To accommodate the case when either endpoint of e is a facility, we post-process the output of our algorithm as follows. If the set S of local MinSum locations returned by our algorithm contains a facility endpoint, we set $S = \emptyset$; otherwise, we keep S intact. To understand this post-processing step, observe that the merit of any facility point is zero, since building a new facility on any point in F does not change the attractor distance of any client. Hence, if S contains a facility point, then the maximum merit of all points on e should be zero. In that case, the global MinSum location must not be on e, and hence, we can ignore the local MinSum locations found on e.

C. MinMax Location Queries

Next, we present our solution for finding the local MinMax locations on any edge $e \in E_c$, i.e., the points on e where a new facility can be built to minimize the maximum WAD of all clients. Our solution is based on the following observation: For any client c, the relationship between the WAD of c and the new facility's location can be precisely captured using a piecewise linear function.

For example, consider the edge e_0 in Figure 4(a). Assume that there exist only 4 clients c_1, c_2, c_3, and c_4, as illustrated in the attraction sets in Figure 4(a). Further assume that (i) the clients' attractor distances are as shown in Figure 5(a), and (ii) all clients have a weight 1. Then, if we add a new facility on e_0 that is x ($x \in [0, 5]$) distance away from the left endpoint v_l of e_0, the WAD of c_3 can be expressed as a piecewise linear function:

$$g_3(x) = \begin{cases} x + 1, & \text{if } x \in [0, 3] \\ 7 - x, & \text{if } x \in (3, 5] \end{cases}$$

Algorithm *MinMaxLoc* (e)

1. let ℓ be the length of e, and v_l (v_r) be the left (right) endpoint of e
2. construct an empty two-dimensional plane R
3. let C_- be the set of clients that appear in neither $\mathcal{A}(v_l)$ nor $\mathcal{A}(v_r)$
4. find the client $c_0 \in C_-$ with the largest WAD
5. construct the WAD function of c_0, i.e., draw in R a line segment from point with coordinate $\big(0, \widehat{a}(c_0)\big)$ to point with coordinate $\big(\ell, \widehat{a}(c_0)\big)$
6. let C_Δ be the set of clients that appear in both $\mathcal{A}(v_l)$ and $\mathcal{A}(v_r)$
7. for each client $c \in C - C_\Delta - C_-$
8. if c appears in $\mathcal{A}(v_l)$
9. $x_1 = 0, \quad y_1 = w(c) \cdot d(c, v_l)$
10. $x_3 = \ell, \quad x_2 = \min\{\ell, a(c) - d(c, v_l)\}$
11. $y_2 = y_3 = w(c) \cdot \big(x_2 + d(c, v_l)\big)$
12. else /*if c does not appear in $\mathcal{A}(v_l)$, but appears in $\mathcal{A}(v_r)$*/
13. $x_1 = \ell, \quad y_1 = w(c) \cdot d(c, v_r)$
14. $x_3 = 0, \quad x_2 = \max\{0, \ell - a(c) + d(c, v_r)\}$
15. $y_2 = y_3 = w(c) \cdot \big(\ell - x_2 + d(c, v_r)\big)$
16. construct the WAD function of c, i.e., draw in R two line segments, from (x_1, y_1) to (x_2, y_2), then to (x_3, y_3)
17. for each client $c \in C_\Delta$ /*c appears in both $\mathcal{A}(v_l)$ and $\mathcal{A}(v_r)$*/
18. $x_1 = 0, \quad y_1 = w(c) \cdot d(c, v_l)$
19. $\beta = \frac{1}{2}\ell - \frac{1}{2}d(c, v_l) + \frac{1}{2}d(c, v_r)$
20. $x_2 = \min\{\beta, a(c) - d(c, v_l)\}, \quad y_2 = w(c) \cdot \big(x_2 + d(c, v_l)\big)$
21. $x_3 = \max\{\beta, \ell - a(c) + d(c, v_r)\}, \quad y_3 = y_2$
22. $x_4 = \ell, \quad y_4 = w(c) \cdot d(c, v_r)$
23. construct the WAD function of c, i.e., draw in R three line segments, from (x_1, y_1) to (x_2, y_2), then to (x_3, y_3), then to (x_4, y_4)
24. compute the upper envelope g_{up} of the WAD functions in R
25. identify and return the points on which g_{up} is minimized

Fig. 6. The *MinMaxLoc* Algorithm

We define g_3 as the WAD *function* of c_3. Similarly, we can also derive a WAD function g_i for each of the other client c_i ($i = 1, 2, 4$). Figure 5(b) illustrates g_i ($i \in [1, 4]$).

Let g_{up} be the *upper envelope* [19] of $\{g_i\}$, i.e., $g_{up}(x) = \max_i\{g_i(x)\}$ for any $x \in [0, 5]$ (see Figure 5(b)). Then, $g_{up}(x)$ captures the maximum WAD of the clients when a new facility is built on x. Thus, if the point (on e_0) that is x distance away from v_l is a local MinMax location, then g_{up} must be minimized at x, and vice versa. As shown in Figure 5(b), g_{up} is minimized when $x \in [0, 0.5]$. Hence, the local MinMax locations on e_0 are the points p on e_0 with $d(p, v_l) \in [0, 0.5]$.

In general, to compute the local MinMax locations on an edge e, it suffices to first construct the upper envelope of all clients' WAD functions, and then identify the points at which the upper envelope is minimized. This motivates our *MinMaxLoc* algorithm (in Figure 6) for computing local MinMax locations.

Given an edge $e \in E_c$, *MinMaxLoc* first retrieves two attraction sets $\mathcal{A}(v_l)$ and $\mathcal{A}(v_r)$, where v_l (v_r) is the left (right) endpoint of e. After that, it creates a two-dimensional plane R, in which it will construct the WAD functions of some clients. Specifically, *MinMaxLoc* first identifies the set C_- of clients that appear in neither $\mathcal{A}(v_l)$ nor $\mathcal{A}(v_l)$. By Lemma 1, for any client $c \in C_-$, the attractor distance of c is not affected by a new facility built on e. Hence, the WAD function of c can be

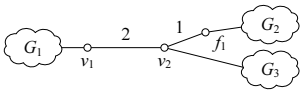


Fig. 7. Example of Lemma 4

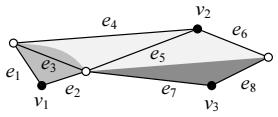


Fig. 8. Example of *GPart*

represented by a horizontal line segment in R. Observe that, only one of those segments may affect the upper envelope g_{up}, i.e., the segment corresponding to the client c^* with the largest WAD in C_-. Therefore, given C_-, *MinMaxLoc* only constructs the WAD function of c^*, ignoring all the other clients in C_-.

Next, *MinMaxLoc* examines each client $c \in C - C_-$, and derive the WAD function of c based on $\mathcal{A}(v_l)$ and $\mathcal{A}(v_r)$. In particular, each WAD function is represented using at most three line segments in R. Finally, *MinMaxLoc* computes the upper envelope g_{up} of the WAD functions in R, and then identifies and returns the points at which g_{up} is minimized.

MinMaxLoc can be implemented in $O(n \log n)$ time and $O(n)$ space. Specifically, given the attractor distances of the clients in C, we can identify the client c^* with $O(n)$ time and space. After that, it takes only $O(n)$ time and space to construct the WAD functions of clients, since each function is represented with $O(1)$ line segments. As there exist $O(n)$ segments in R, the upper envelope g_{up} should contain $O(n)$ linear pieces, and can be computed in $O(n \log n)$ time and $O(n)$ space [20]. Finally, by scanning the $O(n)$ linear pieces of g_{up}, we can compute the local MinMax locations on e in $O(n)$ time and space.

In addition, *MinMaxLoc* can also be extended to handle the case when either endpoint of e is a facility in F. In particular, if the left endpoint v_l of e is a facility, then *MinMaxLoc* excludes v_l when it computes the upper envelope g_{up} of the WAD functions. That is, the domain of g_{up} is defined as $(0, \ell]$ instead of $[0, \ell]$. The case when the right endpoint of e is a facility can be addressed similarly.

V. Computing Attraction Sets and Attractor Distances

Our algorithms in Section IV require as input (i) the attractor distances of all clients in C, and (ii) the attraction sets of the endpoints of the given edge $e \in E_c$. The attractor distances can be easily computed using the algorithm by Erwig and Hagen [21]. Specifically, Erwig and Hagen's algorithm takes as input a road network G and a set F of facilities. With $O(n \log n)$ time and $O(n)$ space, the algorithm can identify the distance from *each vertex v in G to its nearest facility in F*. In the following, we will investigate how to compute the attraction sets of the vertices in G, given the attractor distances derived from Erwig and Hagen's algorithm.

A. The Blossom Algorithm

By definition, a client c appears in the attraction set of a vertex v, if and only if $d(c, v)$ is no more than the attractor distance $a(c)$ of c. Therefore, given the attractor distances of all clients, we can compute the attraction sets of all vertices in G in a batch as follows. First, we set the attraction set of every vertex in G to $\emptyset$. After that, for each client $c \in C$,

Algorithm *Blossom* (G)

1. initialize the attraction set of each vertex v in G as $\emptyset$
2. for each client c
3. employ Dijkstra's algorithm to traverse the vertices in G in ascending order of their distances to c
4. for each vertex v traversed
5. if $d(c, v) \leq a(c)$
6. add an entry $\langle c, d(c, v) \rangle$ to the attraction set of v
7. else goto Line 2
8. return

Fig. 9. The *Blossom* Algorithm

we apply Dijkstra's algorithm [22] to traverse the vertices in G in ascending order of their distances to c. For each vertex v encountered, we check whether $d(c, v) \leq a(c)$. If $d(c, v) \leq a(c)$, c is inserted into the attraction set of v. Otherwise, $d(c, v') > a(c)$ must hold for any unvisited vertex v', i.e., none of the unvisited vertices can attract c. In that case, we terminate the traversal and proceed to the next client. Once all clients are processed, we obtain the attraction sets of all vertices in G. We refer to the above algorithm as *Blossom*, as illustrated in Figure 9.

Blossom has an $O(n^2 \log n)$ time complexity, since it invokes Dijkstra's algorithm once for each client, and each execution of Dijkstra's algorithm takes $O(n \log n)$ time in the worst case [22]. *Blossom* requires $O(n^2)$ space, as it materializes the attraction set of each vertex in G, and each attraction set contains the information of $O(n)$ clients. Such space consumption is prohibitive when n is large. To remedy this deficiency, Section V-B proposes an alternative solution that requires only $O(n)$ space.

B. The OTF Algorithm

The enormous space requirement of *Blossom* is caused by the massive materialization of attraction sets. A natural idea to reduce the space overhead is to avoid storing the attraction sets, and derive them only when needed. That is, whenever we need to compute the local optimal locations on an edge $e \in E_c$, we compute the attraction sets of e's endpoints *on the fly*, and then discard the attraction sets once the local optimal locations are found. But the question is, given a vertex v in G, how do we construct the attraction set $\mathcal{A}(v)$ of v? A straightforward solution is to apply Dijkstra's algorithm to scan through all vertices in G in ascending order of their distances to v. In particular, for each client c encountered during the scan, we examine the distance $d(v, c)$ from c to v. If $d(v, c) < a(c)$, we add c into $\mathcal{A}(v)$; otherwise, c is ignored. Apparently, this solution incurs significant computation overhead, as it requires traversing a large number of vertices. Is it possible to compute $\mathcal{A}(v)$ without an exhaustive search of the vertices in G? The following lemma provides us some hints.

Lemma 4: Given two vertices v and v' in G, such that $d(v, v')$ is larger than the distance from v' to its nearest facility f'. Then, $\forall c \in \mathcal{A}(v)$, the shortest path from v to c must not go through v'.

Algorithm *OTF* (v)

1. initialize the attraction set $\mathcal{A}(v)$ of v as $\emptyset$
2. employ Dijkstra's algorithm to traverse the vertices in G in ascending order of their distances to v
3. for each vertex v' examined
4. let λ be distance from v' to its closest facility
5. if $d(v, v') \leq \lambda$ and v' is a client point
6. insert an entry $\langle v',\ dist(v', v)\rangle$ into $\mathcal{A}(v)$
7. if $d(v, v') > \lambda$
8. ignore all edges adjacent to v', i.e., regard them as deleted
9. if none of the unvisited vertices can be reached from v
10. return $\mathcal{A}(v)$
11. return $\mathcal{A}(v)$

Fig. 10. The *OTF* Algorithm

Proof: Assume on the contrary that the shortest path from v to c goes through v'. Then, we have

$$d(v, c) = d(v, v') + d(v', c) > d(v', f') + d(v', c) \geq d(f', c).$$

This contradicts our assumption that c is attracted by v. ■

Consider for example the road network in Figure 7, which contains three subgraph G_1, G_2, and G_3 that are connected by a facility f_1 and two vertices v_1 and v_2. In addition, $d(v_1, v_2) = 2$, $d(v_2, f_1) = 1$, and f_1 is the facility closest to v_2. Since $d(v_1, v_2) > d(v_2, f_1)$, by Lemma 4, v_2 must not be on the shortest path from v_1 to any client attracted by v_1. This means that no client in G_2 or G_3 can be attracted by v_1, because all paths from v_1 to G_2 or G_3 go through v_2. Therefore, if we are to compute $\mathcal{A}(v_1)$, it suffices to examine only the clients in G_1.

Based on Lemma 4, we propose the *OTF (On-The-Fly)* algorithm (in Figure 10) for computing the attraction set of a vertex v in G. Given v, *OTF* first sets $\mathcal{A}(v) = \emptyset$, and then applies Dijkstra's algorithm to visit the vertices in G in ascending order of their distances to v. For each vertex v' visited, *OTF* retrieves the distance λ from v' to its closest facility (recall that λ is computed using Erwig and Hagen's algorithm [21]). If $d(v, v') \leq \lambda$ and v' is a client, then *OTF* adds v' into $\mathcal{A}(v)$. On the other hand, if $d(v, v') > \lambda$, then *OTF* ignores all edges adjacent to v' when it traverses the remaining vertices in G. This does not affect the correctness of *OTF*, since, by Lemma 4, deleting v' from G does not change the shortest path from v to any client attracted by v. After v is processed, *OTF* checks whether any of the unvisited vertices in G is still connected to v. If none of those vertices is connected to v, *OTF* terminates by returning $\mathcal{A}(v)$; otherwise, *OTF* proceeds to the unvisited vertex that is closest to v.

It is not hard to verify that *OTF* runs in $O(n \log n)$ time and $O(n)$ space. Therefore, if we employ *OTF* to compute the local optimal locations on every edge in G, then the total time required for deriving the attraction sets would be $O(n^2 \log n)$, and the total space needed is $O(n)$ (as *OTF* does not materialize any attraction sets). In contrast, computing the attraction sets with *Blossom* incurs $O(n^2 \log n)$ time and $O(n^2)$ space overhead. Hence, *OTF* is more favorable than *Blossom* in terms of asymptotic performance.

VI. Pruning of Road Segments

Given the algorithms in Sections IV and V, we may answer any OL query by first enumerating the local optima on each edge in E_c, and then deriving the global optimal solutions based on the local optima. This approach, however, incurs a significant overhead when E_c contains a large number of edges. To address this issue, in this section we propose a *fine-grained partitioning (FGP)* technique to avoid the exhaustive search on the edges in E_c.

A. Algorithm Overview

At a high level, FGP works in four steps as follows. First, we divide G into m edge-disjoint subgraphs $G_1, G_2, \ldots, G_m$, where m is an algorithm-specific parameter. Second, for each subgraph G_i ($i \in [1, m]$), we derive a *potential client set* C_i of G_i, i.e., a superset of all clients that can be attracted by a new facility built on any edge in G_i.

As a third step, we inspect each potential client set C_i ($i \in [1, m]$), based on which we derive an upper-bound of the *benefit* of any candidate location p in G_i. Specifically, for competitive location queries, the benefit of p is defined as the total weight of the clients attracted by p; for MinSum (MinMax) location queries, the benefit of p is quantified as the reduction in the total (maximum) WAD of all clients, when a new facility is built on p.

Finally, we examine the subgraphs $G_1, G_2, \ldots, G_m$ in descending order of their benefit upper-bounds. For each subgraph G_i, we apply the algorithms in Sections IV and V to identify the local optimal locations in G_i. After processing G_i, we inspect the set S of best local optima we have found so far. If the benefits of those locations are larger than the benefit upper-bounds of all unvisited subgraphs, we terminate the search and return S as the final results; otherwise, we move on to the next subgraph.

The efficacy of the above framework rely on three issues, namely, (i) how the subgraphs of G are generated, (ii) how the potential client set C_i of each subgraph G_i is derived, and (iii) how the benefit upper bound of G_i is computed. In the following, we will first clarify how FGP derives benefit upper-bounds, deferring the solutions to the other two issues to Section VI-B. We begin with the following lemma.

Lemma 5: Let G_i be a subgraph of G, C_i be the potential client set of G_i, and p be a candidate location in G_i. Then, for any competitive location query, the benefit of p is at most $\sum_{c \in C_i} w(c)$. For any MinSum location query, the benefit of p is at most $\sum_{c \in C_i} \widehat{a}(c)$. For any MinMax location query, the benefit of p is at most $\max_{c \in C} \widehat{a}(c) - \max_{c \in C - C_i} \widehat{a}(c)$.

Proof: The lemma follows from the facts that (i) C_i contains all clients that can be attracted by p, and (ii) for any client in C_i, its WAD is at least zero after a new facility is built on p. ■

The benefit upper-bounds in Lemma 5 require knowledge of all clients' attractor distances, which, as mentioned in

Algorithm *GPart* (G, θ)

1. construct a set V' that contains all the endpoints of the edges that appear in both G and E_c
2. randomly sample a set V_Δ of vertices from V' with a sampling rate θ
3. create $|V_\Delta|$ empty subgraphs, and assign each vertex in V_Δ as the "center" of a distinct subgraph
4. feed G and V_Δ as input to Erwig and Hagen's algorithm [21] to compute, for each vertex v in G, (i) the vertex $v' \in V_\Delta$ that is closest to v, as well as (ii) the distance $d(v, v')$ from v to v'
5. insert each edge in G to the subgraph whose center is the closest to either endpoint of the edge
6. return all subgraphs

Fig. 11. The *GPart* Algorithm

Section V, can be computed in $O(n \log n)$ time and $O(n)$ space using Erwig and Hagen's algorithm [21]. We can further sort the attractor distances in descending order in $O(n \log n)$ time and $O(n)$ space. Observe that, given the sorted attractor distances and the potential client set C_i of a subgraph G_i, the benefit upper-bound of G_i (for any OL query) can be computed efficiently in $O(|C_i|)$ time and $O(n)$ space.

B. Graph Partitioning

We are now ready to discuss how FGP generates the subgraphs from G and computes the potential client set of each subgraph. In particular, FGP generates subgraphs from G by applying an algorithm called *GPart* (as illustrated Figure 11), which takes as input G and a user defined parameter $\theta \in (0, 1]$. *GPart* first identifies the set V of vertices in G that are adjacent to some edges in E_c. As a second step, *GPart* computes a random sample set V_Δ of the vertices in V with a sampling rate θ, after which it splits G into subgraphs based on V_Δ.

Specifically, *GPart* first constructs $|V_\Delta|$ empty subgraphs, and assigns each vertex in V_Δ as the "center" of a distinct subgraphs. After that, for each vertex v in G, *GPart* identifies the vertex $v' \in V_\Delta$ that is the closest to v, and computes $d(v, v')$. This step can be done by applying Erwig and Hagen's algorithm [21], with G and V_Δ as the input. Next, for each edge e in G, *GPart* checks the two endpoints v_l and v_r of e, and inserts e into the subgraph whose center v' minimizes $\min\{d(v', v_l), d(v', v_r)\}$. After all edges in G are processed, *GPart* terminates by returning all subgraphs constructed. For example, if G equals the graph in Figure 8 and $V_\Delta = \{v_1, v_2, v_3\}$, then *GPart* would construct three subgraphs $\{e_1, e_2, e_3\}$, $\{e_4, e_5, e_6\}$, and $\{e_7, e_8\}$, whose centers are v_1, v_2, and v_3, respectively. In summary, *GPart* ensures that the edges in the same subgraph form a cluster around the subgraph center. As such, the edges belonging to the same subgraph tend to be close to each other. This helps tighten the benefit upper-bounds of the subgraph because, intuitively, points in proximity to each other have similar benefits. Regarding asymptotic performance, it can be verified that *GPart* runs in $O(n \log n)$ time and $O(n)$ space.

Given a subgraph G_i obtained from *GPart*, our next step is to derive the potential client set for each G_i. Let E_{G_i} be the set of edges in G_i that also appear in E_c, and V_{G_i} be the set of endpoints of the edges in E_{G_i}. By Lemma 1, for any candidate location p on an edge e in G_i, the set of clients attracted by p is always a subset of the clients in the attraction sets of e's endpoints. Therefore, the potential client set of G_i can be formulated as the set of all clients that appear in the attraction sets of the vertices in V_{G_i}. In turn, the attraction sets of the vertices in V_{G_i} can be derived by applying either the *Blossom* algorithm or the *OTF* algorithm in Section V. In particular, if *Blossom* is adopted, then we feed the graph G as the input to *Blossom*[1]. In return, we obtain the attraction sets of all vertices in G, based on which we can compute the potential clients set of *all* subgraphs in G. In addition, the attraction sets can be reused when we need to compute the local optimal locations on any edge in G.

Algorithm *P-OTF* (G, G_i)

1. let V_i' be the set of endpoints of the edges in $G_i \cap E_c$
2. insert a new vertex v_0 in G
3. connect v_0 to each vertex in V_i' via an edge with a length 0
4. compute the attraction set of v_0 in G using the *OTF* algorithm
5. return the set of clients in the attraction set of v_0

Fig. 12. The *P-OTF* Algorithm

On the other hand, if *OTF* is adopted, then we feed each vertex in V_{G_i} to *OTF* to compute its attraction set, after which we collect all clients that appear in at least one of the attraction sets. The drawback of this approach is that it requires multiple executions of *OTF*, which leads to inferior time efficiency. To remedy this drawback, we propose the *P-OTF* algorithm (in Figure 12) for computing the potential client set of a subgraph G_i. Given the graph G, *P-OTF* first creates a new vertex v_0 in G, and then constructs an edge between v_0 and each vertex in V_{G_i}, such that the edge has a length 0. After that, *P-OTF* invokes *OTF* *once* to compute the attraction set of v_0 in G. Observe that, if a client c is attracted by v_0, then there must exist a vertex in V_{G_i} that attracts c, and vice versa. Hence, the potential client set of G_i should be equal to the set of clients in the attraction set $\mathcal{A}(v_0)$ of v_0. Therefore, once $\mathcal{A}(v_0)$ is computed, *P-OTF* terminates by returning all clients in $\mathcal{A}(v_0)$. In summary, *P-OTF* computes the potential client set of G_i by invoking *OTF* only once, which incurs $O(n \log n)$ time and $O(n)$ space overhead.

Before closing this section, we discuss how we set the input parameter θ of *GPart*. In general, a larger θ results in smaller subgraphs, which in turn leads to tighter benefit upper-bounds. Nevertheless, the increase in θ would also lead to a larger number of subgraphs, which entails a higher computation cost, as we need to derive the potential client set for each subgraph. Ideally, we should set θ to an appropriate value that strikes a good balance between the tightness of the benefit upper-bounds and the cost of deriving the bounds. We observe that, when the potential client sets of the subgraphs are computed using *Blossom*, θ should be set to 1. This is because, *Blossom* derives potential client sets by computing the attraction sets of

[1]Note that we cannot apply *Blossom* on G_i directly, since a candidate location in G_i may attract a client outside G_i.

all vertices, regardless of the value of θ. As a consequence, the computation cost of the benefit upper-bounds is independent of θ. Hence, we can set $\theta = 1$ to obtain the tightest benefit upper-bounds without sacrificing time efficiency. On the the hand, if *P-OTF* is adopted, then the overhead of computing potential client sets increases with the number of subgraphs. To ensure that this computation overhead does not affect the overall performance, θ should be set to a small value. We suggest setting $\theta = 1‰$ across the board.

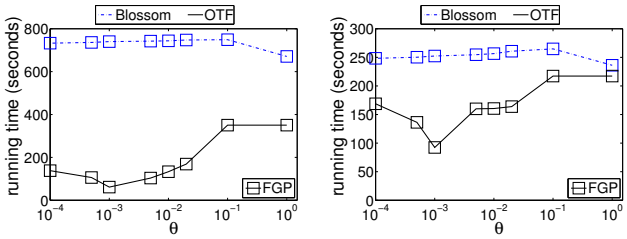
VII. Experiment

This section experimentally evaluates the proposed solutions. For each type of OL queries, we examine two approaches for traversing the edges in E_c, (i) the *Basic* approach that computes the local optimal locations on every edge in E_c before returning the final results and (ii) the *Fine-Grained Partitioning (FGP)* approach. For each approach, we combine it with two different techniques for deriving attraction sets, i.e., *Blossom* and *OTF*. We implement our algorithms in C++, and perform all experiments on a Linux machine with an Intel Xeon 2GHz CPU and 4GB memory. Our implementation uses the widely adopted road network representation proposed by Shekhar and Liu [23]. The running time reported in our experiments includes the cost of all steps, including the overhead for computing attractor distances using Erwig and Hagen's algorithm [21] (see Section V).

Datasets. We use two real road network datasets, *SF* and *CA*, obtained from the *Digital Chart of the World Server*. In particular, *SF* (*CA*) captures the road network in San Francisco (California), and contains 174,955 nodes and 223,000 edges (21,047 nodes and 21,692 edges). We obtain a large number of real building locations in San Francisco (California) from the *OpenStreetMap* project, and use random sample sets of those locations as facilities and clients on *SF* (*CA*). We synthesize the weight of each client by sampling from a Zipf distribution with a skewness parameter $\alpha > 1$.

Default Settings. We vary five parameters: (i) the number of facilities $|F|$, (ii) the number of clients $|C|$, (iii) the percentage $\tau = |E_c^\circ|/|E^\circ|$ of edges (in the given road network) where the new facility can be built, (iv) the input parameter θ of the *FGP* algorithm (see Figure 11), and (v) the skewness parameter α of the Zipf distribution from which we sample the weight of each client. Unless specified otherwise, we set $|F|$ =1000 and $|C|$ =300,000, so as to capture the likely scenario in practice where the number of clients is much larger than the number of facilities. We also set $\tau = 100\%$, in which case the OL queries are most computationally challenging, since we need to consider *every* point in the road network as a candidate location. The default value of θ is set to 100% (1‰) when *Blossom* (*OTF*) is used to derive attraction sets, as discussed in Section VI-B. Finally, we set $\alpha = +\infty$ by default, in which case all clients have a weight 1.

Effect of θ. Our first sets of experiments focus on competitive location queries (CLQ). Figure 13 shows the effect of θ on the



(a) *SF* (b) *CA*

Fig. 13. Running Time vs. θ (CLQ)

running time of our solutions that incorporate *FGP*. Observe that, when *Blossom* is adopted to compute attraction sets, our solution is most efficient at $\theta = 1$. This is consistent with the analysis in Section VI-B that $\theta = 1$ (i) leads to the tightest benefit upper-bounds, and thus, (ii) facilitates early termination of edge traversal. In contrast, when *OTF* is adopted, $\theta = 1$ results in inferior computational efficiency. This is because, when *OTF* is employed, a larger θ leads to a higher cost for deriving benefit upper-bounds, which offsets the efficiency gain obtained from the tighter upper-bounds. On the other hand, $\theta = 1‰$ strikes a good balance between the overhead of upper-bound computation and the tightness of the upper-bounds, which justifies our choice of default values for θ.

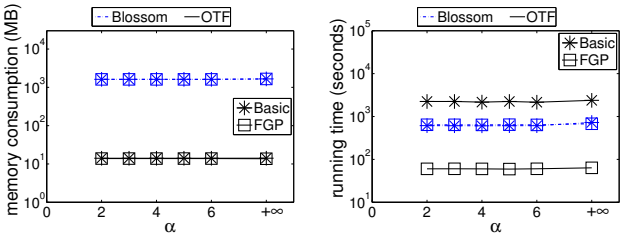
The *OTF*-based solution outperforms the *Blossom*-based approach in both cases. The reason is that, regardless of the value of θ, the *Blossom*-based approach requires computing the attraction sets of all vertices. In contrast, the *OTF*-based solution only needs to derive the attraction sets of the vertices in each *subgraph* it visits. Since the *OTF*-based solution visits only subgraphs whose benefit upper-bounds are large, it computes a much smaller number of attraction sets than the *Blossom*-based approach does, and hence, it achieves superior efficiency.

For brevity, in the following we focus on the *SF* dataset (since it is larger). The results on *CA* are qualitatively similar.

Effect of α. Figure 14 shows the effect of α on the performance of our solutions, varying α from 2 to $+\infty$. Evidently, the memory consumptions and running time of our solutions are insensitive to the clients' weight distributions.

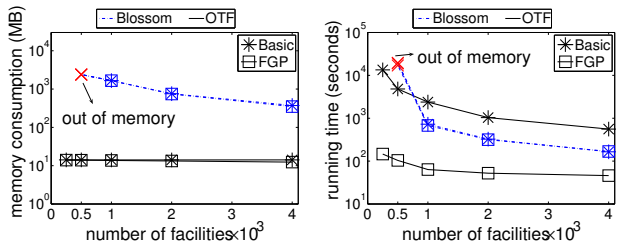
Effect of $|F|$. Next, we vary the number of facilities ($|F|$) from 250 to 4000. Figures 15(a) illustrates the memory consumptions of our solutions on *SF*. The methods based on *Blossom* incur significant space overheads, and run out of memory when $|F|$ <1000. This is due to the $O(n^2)$ space complexity of *Blossom*. In contrast, the memory consumptions of *OTF*-based methods are always below 20MB, since *OTF* incurs only $O(n)$ space overhead.

Figure 15(b) shows the running time of each of our solutions as a function of $|F|$. The methods that incorporate *FGP* outperform the approaches with *Basic* in all cases, since *FGP* provides much more effective means to avoid visiting the edges that do not contain optimal locations. In addition, when *FGP* is adopted, the *OTF*-based approach is superior to the *Blossom*-based approach, as is consistent with the results in Figure 13. Finally, the running time of all solutions decreases



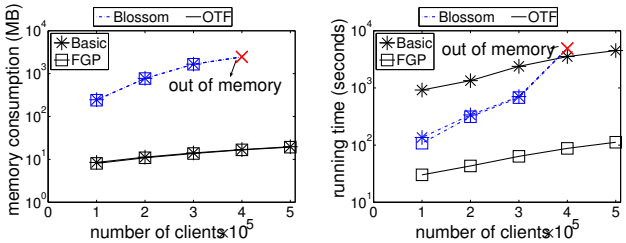
(a) Memory Consumption (b) Running Time

Fig. 14. Effect of α (CLQ on *SF*)



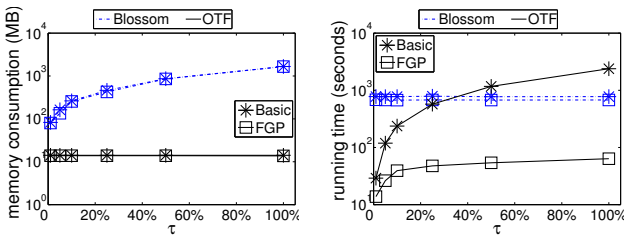
(a) Memory Consumption (b) Running Time

Fig. 15. Effect of $|F|$ (CLQ on *SF*)



(a) Memory Consumption (b) Running Time

Fig. 16. Effect of $|C|$ (CLQ on *SF*)



(a) Memory Consumption (b) Running Time

Fig. 17. Effect of τ (CLQ on *SF*)

with the increase of $|F|$. The reason is that, when $|F|$ is large, each client c tends to have a smaller attractor distance $a(c)$. This reduces the number of road network vertices that are within $a(c)$ distance to c, and hence, c should appear in a smaller number of attraction sets. Therefore, the attract set of each vertex in the road network would become smaller, in which case that *Blossom* and *OTF* can be executed in shorter time. Consequently, the overall running time of our solutions is reduced.

Effect of $|C|$. The next set of experiments investigate the scalability of our solutions by varying the number of clients, $|C|$, from 100,000 to 500,000. Figures 16(a) (16(b)) shows the space consumptions (running time) of our solutions as functions of $|C|$. As in the previous experiments, *Blossom*-based approaches consume enormous amounts of memory, while the method that incorporates both *FGP* and *OTF* consistently outperform all the other methods in terms of both space and time. The running time of all solutions increases with $|C|$, because a larger $|C|$ leads to (i) more edges in the transformed network G and (ii) more clients in each attraction set, both of which complicate the computation of optimal locations.

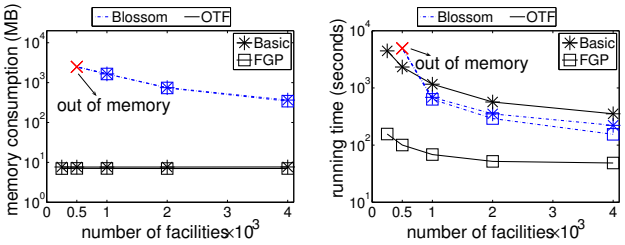
Effect of τ. Figure 17 illustrates the memory consumptions and running time of our solutions when τ changes from 1% to 100%. *Blossom*-based approaches require less memory when τ decreases, since (i) a smaller τ leads to fewer edges in E_c and (ii) *Blossom* stores only the attraction sets of the endpoints of the edges in E_c. In contrast, the space overheads of *OTF*-based methods do not change with τ, since they always compute attraction sets on the fly, regardless of the value of τ.

On the other hand, the running time of *Blossom*-based approaches is not affected by τ. This is because, *Blossom* computes attraction sets, by invoking Dijkstra's algorithm for each client $c \in C$, and putting c in the attraction set of every vertex $v \in V$ such that $d(c, v) \leq a(c)$. Observe that, even if we require only a single attraction set of a vertex $v_0 \in V$, *Blossom* still needs to execute Dijkstra's algorithm once for each client c; otherwise, it is impossible to decide whether $d(c, v_0) \leq a(c)$ holds or not. As a consequence, the efficiency of *Blossom*-based approaches does not improve with the decrease of τ. In contrast, *OTF*-based solutions incur much less computation overhead when τ is reduced, since they compute attraction sets by invoking *OTF* only on the vertices of the edges in E_c. The decrease in τ renders $|E_c|$ smaller, in which case the *OTF*-based solutions require fewer executions of the *OTF* algorithm, and hence, their running time decreases.

MinSum and MinMax Location Queries. The rest of our experiments evaluate the performance of our solutions for MinSum location queries (MinSumLQ) and MinMax location queries (MinMaxLQ). In general, the experimental results are mostly similar to those for competitive location queries. This is not surprising, because our solutions for the three types of queries follow the same framework, and adopt the same algorithmic components (e.g., *Blossom* and *OTF*).

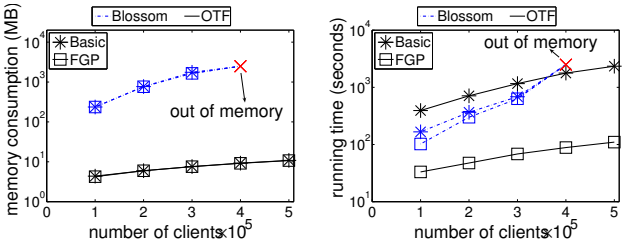
Figures 18 and 19 show the effects of $|F|$ and $|C|$ on the performance of our solutions for MinSum location queries. Again, the method that combines *FGP* and *OTP* achieves the best space and time efficiency in all cases. In addition, *Blossom*-based approaches entails excessively high memory consumption, especially when $|F|$ <1000 or $|C|$ >300,000.

Figures 20 and 21 plot the memory consumptions and running time of our solutions for MinMax location queries. The relative performance of each method remains the same as in Figures 18 and 19. Interestingly, each method incurs a higher computation time for MinMax location queries than for the other two types of OL queries. This is caused by the fact that, our solutions identify local MinMax locations on any given edge e, by computing the upper envelope of a set of WAD functions (see Section IV-C). This procedure is more costly than computing the local competitive (MinSum) locations on e.



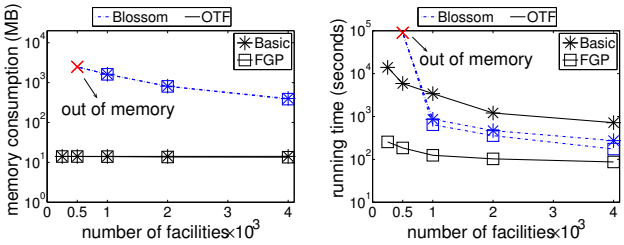
(a) Memory Consumption (b) Running Time

Fig. 18. Effect of $|F|$ (MinSumLQ on *SF*)



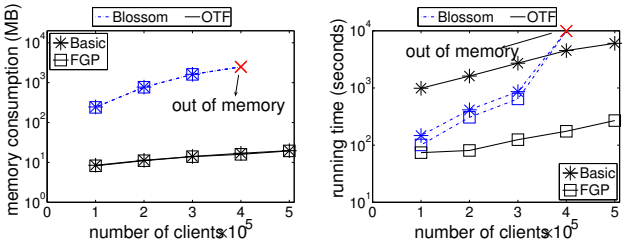
(a) Memory Consumption (b) Running Time

Fig. 19. Effect of $|C|$ (MinSumLQ on *SF*)



(a) Memory Consumption (b) Running Time

Fig. 20. Effect of $|F|$ (MinMaxLQ on *SF*)



(a) Memory Consumption (b) Running Time

Fig. 21. Effect of $|C|$ (MinMaxLQ on *SF*)

Summary. Our results show that the solutions incorporating *FGP* and *OTF* consistently achieve the best performance for all three types of OL queries, in term of both space and time. In particular, they require less than 20MB memory and 200 seconds to answer OL queries in a road network with 174,955 nodes, 223,000 edges, up to 500,000 clients, and down to 250 facilities (recall that the performance of the solutions improve with the number of facilities). The experimental results from the datasets we have tested suggest that setting $\theta = 1‰$ seems to provide a good performance.

VIII. Conclusion

This work presents a comprehensive study on optimal location queries in road network databases, closing the gap between previous studies and practical applications in road networks. Our study covers three important types of optimal location queries, and introduces a unified framework that addresses all three query types efficiently. Extensive experiments on real datasets demonstrate the scalability of our solution in terms of running time and space consumption. Interesting directions for future work include (i) the incremental monitoring of the optimal locations when the facility or the client set has been updated, and (ii) the optimal location queries for moving objects in road networks.

IX. Acknowledgment

Xiaokui Xiao was supported by the Nanyang Technological University under SUG Grant M58020016 and AcRF Tier-1 Grant RG 35/09. Feifei Li and Bin Yao were partially supported by NSF Grant IIS-0916488.

References

[1] D. Zhang, Y. Du, T. Xia, and Y. Tao, "Progressive computation of the min-dist optimal-location query," in *VLDB*, 2006, pp. 643–654.

[2] Y. Du, D. Zhang, and T. Xia, "The optimal-location query," in *SSTD*, 2005, pp. 163–180.

[3] S. Cabello, J. M. Díaz-Báñez, S. Langerman, C. Seara, and I. Ventura, "Reverse facility location problems," in *CCCG*, 2005, pp. 68–71.

[4] R. C.-W. Wong, T. Özsu, P. S. Yu, A. W.-C. Fu, and L. Liu, "Efficient method for maximizing bichromatic reverse nearest neighbor," *PVLDB*, vol. 2, no. 1, pp. 1126–1137, 2009.

[5] R. Z. Farahani and M. Hekmatfar, *Facility Location: Concepts, Models, Algorithms and Case Studies*, 1st ed. Physica-Verlag HD, 2009.

[6] S. Nickel and J. Puerto, *Location Theory: A Unified Approach*, 1st ed. Springer, 2005.

[7] A. Meyerson, "Online facility location," in *FOCS*, 2001, pp. 426–431.

[8] D. Fotakis, "Incremental algorithms for facility location and k-median," *Theor. Comput. Sci.*, vol. 361, no. 2-3, pp. 275–313, 2006.

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

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

[11] C. S. Jensen, J. Kolářvr, T. B. Pedersen, and I. Timko, "Nearest neighbor queries in road networks," in *GIS*, 2003, pp. 1–8.

[12] J. Sankaranarayanan, H. Samet, and H. Alborzi, "Path oracles for spatial networks," *PVLDB*, vol. 2, no. 1, pp. 1210–1221, 2009.

[13] J. Sankaranarayanan and H. Samet, "Distance oracles for spatial networks," in *ICDE*, 2009, pp. 652–663.

[14] M. L. Yiu, N. Mamoulis, and D. Papadias, "Aggregate nearest neighbor queries in road networks," *TKDE*, vol. 17, no. 6, pp. 820–833, 2005.

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

[16] H. Samet, J. Sankaranarayanan, and H. Alborzi, "Scalable network distance browsing in spatial databases," in *SIGMOD*, 2008, pp. 43–54.

[17] Z. Chen, H. T. Shen, X. Zhou, and J. X. Yu, "Monitoring path nearest neighbor in road networks," in *SIGMOD*, 2009, pp. 591–602.

[18] K. Deng, X. Zhou, H. T. Shen, S. Sadiq, and X. Li, "Instance optimal query processing in spatial networks," *VLDBJ*, vol. 18, no. 3, pp. 675–693, 2009.

[19] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*, 3rd ed. Springer-Verlag, 2008.

[20] J. Hershberger, "Finding the upper envelope of n line segments in $o(n \log n)$ time," *Inf. Process. Lett.*, vol. 33, no. 4, pp. 169–174, 1989.

[21] M. Erwig and F. Hagen, "The graph voronoi diagram with applications," *Networks*, vol. 36, pp. 156–163, 2000.

[22] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[23] S. Shekhar and D.-R. Liu, "CCAM: A connectivity-clustered access method for networks and network computations," *TKDE*, vol. 9, no. 1, pp. 102–119, 1997.